Centro de Investigación Científica y de Educación Superior de Ensenada

DESARROLLO DE UNA INTERFAZ Q3 PARA REDES ATM QUE CONTEMPLA LA GESTION DE FALLAS

TESIS
MAESTRIA EN CIENCIAS

LUIS ARTURO CALDERON VILLEGAS

Ensenada, Baja Cfa., Mexico. Abril de 1999.

TESIS DEFENDIDA POR

LUIS ARTURO CALDERÓN VILLEGAS

Y APROBADA POR EL SIGUIENTE COMITE

M.C. Jorge Enrique Preciado Velasco

Director del Comité

M.C. Alfonso Angeles Valencia

Miembro del Comité

M.C. Ernesto Eduardo Quiroz Morones

Miembro del Comité

Dr. Jesús Favela Vara

Miembro del Comité

Dr. José Luis Medina Monroy

Jefe del Departamento de Electrónica y Telecomunicaciones

Dr. Federico Graef Ziehl

Director de Estudios de Posgrado

19 de marzo de 1999

CENTRO DE INVESTIGACION CIENTIFICA Y
DE EDUCACION SUPERIOR DE ENSENADA

DIVISION DE FISICA APLICADA

DEPARTAMENTO DE ELECTRONICA Y
TELECOMUNICACIONES

DESARROLLO DE UNA INTERFAZ Q3 PARA REDES ATM QUE CONTEMPLA LA GESTION DE FALLAS

TESIS

que para cubrir parcialmente los requisitos necesarios para obtener
el grado de MAESTRO EN CIENCIAS presenta:

LUIS ARTURO CALDERON VILLEGAS

Ensenada, Baja California; México. Abril de 1999

RESUMEN de la tesis de Luis Arturo Calderón Villegas, presentada como requisito parcial para la obtención del grado de MAESTRO en CIENCIAS en ELECTRONICA Y TELECOMUNICACIONES. Baja California, Abril de1999.

Desarrollo de una interfaz Q3 para redes ATM que contempla la Gestión de Fallas

Resumen aprobado por:

M.C. Jorge Plecido Velasco
Director de Tesis

En los últimos años se ha suscitado una problemática referente a la gestión de redes de telecomunicaciones, esto se debe a la creciente complejidad y diversidad en las redes, así como a la demanda de mayor capacidad de anchos de banda en las mismas. La problemática de la gestión de redes ha creado una creciente necesidad de investigación dentro de esta área. La capacidad y la calidad de servicio en las redes requieren sin embargo no sólo de nuevas y mejores tecnologías básicas sino, de su gestión eficiente.

La Gestión de Red utiliza y coordina los recursos para planear, ejecutar, administrar, analizar, evaluar, diseñar y expandir las redes de comunicaciones para adaptarse al nivel de servicio requerido en todo momento, a un costo razonable y con capacidad óptima.

Las redes actuales se caracterizan por incorporar una variedad muy amplia de esquemas de cableado, métodos de acceso, protocolos, equipos y tecnologías de red. El crecimiento de estas redes y la diversidad de alternativas conducen a un incremento exponencial de la complejidad de la red.

El problema de interconexión, debido a la diversidad de redes de organizaciones, tanto en lo referente al hardware de sus equipos como a su arquitectura, se puede superar utilizando un sistema de Gestión de Red integrador, basado en una norma que sea independiente de equipos, sistemas y de arquitecturas propietarias. La solución que goza de mayor aceptación para llevar al cabo la gestión de redes con tecnologías y sistemas de gestión propietarios es la de integrarlos en una arquitectura común; esta arquitectura es conocida como TMN (Telecommunication Management Network). Esta arquitectura que contempla la coexistencia de distintos sistemas de gestión requiere sin embargo disponer de unos procedimientos de adaptación que tendrán que emplearse a diferentes niveles. Así por ejemplo, la adaptación a una interfaz normalizada podría realizarse a nivel de gestor de subred o incluso a nivel de gestor de red. La decisión depende de la naturaleza de los dispositivos a gestionar y de la complejidad del mecanismo de adaptación.

La ITU-T define la interfaz Q3 para TMN, la cual se encarga de comunicar un sistema de operación (agente para ATM) con un dispositivo de red (conmutador ATM). La interfaz Q3 tiene definidas funciones especificas dentro de la arquitectura TMN; algunas se encuentran en proceso de implantación, como la gestión de fallas.

En este trabajo se implanta parte de la funcionalidad de gestión de fallas en la interfaz Q3 desarrollada en la UPC (Universitat Politecnica de Catalunya) dentro del proyecto europeo MISA. Se desarrolló e integró un proceso de correlación como un primer paso para implantar la gestión de fallas en la Q3, con el cual es posible conocer el tipo de falla en la red, sobre la base de este conocimiento se puede realizar la recuperación de la falla.

Palabras Clave: *Interfaz Q3, Gestión de Fallas, TMN, CMIS/P, SNMP*

ABSTRACT of the Thesis of Luis Arturo Calderón Villegas, presented as a partial requirement to obtain the MASTER IN SCIENCES degree in ELECTRONICS AND TELECOMMUNICATIONS. Ensenada, Baja California, México. April 1999.

Development of a Q3 interface for ATM networks, including fault management procedures

Recently, the problems concerning the management of telecommunications networks have increased due to its complexity, diversity and increased capacity (bandwidth). This has created a growing need for more research in this area. The capacity and quality of service offered by modern networks require both newer and better basic technologies, more efficient management schemes.

Network Management uses and coordinates of resources to planing, execute, administer, analyze, evaluate, design, and expand communication networks to adapt to the demand level at all times, at reasonable cost and with optimum capacity.

Current networks are characterized by incorporating a wide variety of wiring schemes, access methods, protocols, equipment, and technologies. The growth in these networks and the diversity of alternatives lead to an exponential increase in the complexity of the network.

The interconnection problem, due the diversity of the networks in organizations including both the hardware of its equipment and its architecture, can be surpassed using a Network Management system integrator based on a standard that must be independent from the equipment, systems, and the proprietary architectures. The solution that has the greatest acceptance to carry out the management of networks, with a diversity of technologies and proprietary management systems, is the one that integrates them into a common architecture; this architecture is know as TMN (Telecommunication Management Network). Although TMN architecture contemplates the coexistence of different management systems, it requires adaptation procedures that they will be used at different levels. Thus, for instance, the adaptation to a standardized interface could even be made at the level of sub-network manager or at the level of network manager. The decision depends on the nature of the devices to manage and, the complexity of the adaptation mechanism.

The ITU defines the Q3 interface for TMN; this interface communicates an Operation System (agent) with a network device (ATM switch). The Q3 interface defines specific functions for the TMN architecture; some of these functions like fault management are still in the implementation process. In this work part of the fault management functions in Q3 interface are implemented for the MISA project, developed at UPC (Universitat Politecnica de Catalunya), a correlation process is developed and integrated, this is the first step to implement the fault management in Q3. Based on the correlation process, it is possible to know the type of network's faults in order to stablish the recovery mechanisms.

Keywords: *Q3 interface, fault management, TMN, CMIS/P, SNMP*

Dedicatoria

A *Jehová Dios*.
Por permitirme alcanzar esta meta.

A mis padres
José Calderón
Ana Villegas

Por enseñarme a vivir, por su amor, por su paciencia, por su apoyo incondicional, por todas las cosas valiosas que me han ofrecido.

A mis hermanos
Por todos los momentos maravillosos que hemos pasado juntos, por ese apoyo que siempre me han brindado.

Agradecimientos

Al M.C. Alfonso Angeles Valencia, mi director y amigo, quien me brindo todo su apoyo. Por la confianza que depositaste en mí para hacer posible este trabajo.

Al M.C. Jorge Preciado Velasco, mi tutor y amigo, por su paciencia y apoyo desinteresado.

Al M.C. Ernesto Quiroz miembro de mí comité de tesis y amigo por su apoyo y valiosas aportaciones para el desarrollo de este trabajo.

Al Dr. Jesus Favela Vara miembro de mi comité de tesis, por su valiosa cooperación en el desarrollo de este trabajo.

Al Dr. Joan Serrat mi amigo, quien fue el que propuso este tema de tesis y me brindó todos los recursos a su alcance cuando estuve en la UPC en Barcelona.

A Enric mi amigo, con quien trabajé en conjunto unos cuantos días en Barcelona en el desarrollo de este trabajo.

A mis compañeros de degeneración, sí a todos ustedes, por los momentos agradables y desagradables que pasamos juntos en nuestra estancia por CICESE.

Al CITEDI del IPN quien me adoptó como tesista, a todas la personas valiosas tanto profesores y estudiantes que me he encontrado aquí, por la confianza, por todos los recursos que pusieron a mi alcance.

Al CICESE, por brindarme la oportunidad de alcanzar una meta más en mi vida.

Al CONACYT por el apoyo financiero que me brindó para realizar mis estudios de posgrado.

Contenido

Lista de figuras

Lista de tablas

Desarrollo de una Interfaz Q3 para redes ATM que contempla la Gestión de Fallas.

Capítulo I

Introducción

I.1 Antecedentes

Cuando aparecieron las primeras redes de datos, no se planeó la gestión de las mismas ya que por su dimensión no se requería. Al ir popularizándose y en consecuencia aumentando su tamaño y complejidad, los problemas en éstas crecieron en la misma proporción que su dimensión. A estos problemas también se agregaron aquellos que surgen al usar equipos de distintos fabricantes. Las primeras formas de gestión de red se realizaron mediante protocolos propietarios, es decir, cada fabricante establecía la forma de gestionar sus dispositivos en la red, sin embargo, se hacía evidente la necesidad de un protocolo común. SNMP (Simple Network Management Protocol) aparece a finales de la década de los 80s, como respuesta a esta necesidad.

A pesar de que la gestión de las redes de telecomunicaciones no es un problema nuevo, considerando que los primeros esfuerzos para realizar gestión de red tienen casi unos 20 años de existencia, la gestión de redes es muy limitada, ya que sólo el protocolo SNMP está plenamente implantado en los dispositivos de las redes de telecomunicaciones y definido en las normas respectivas Request for Comments [RFC 1155, 1157, 1212]; ésto trae como consecuencia que la gestión esté muy limitada [Stalling William, 1996, pp. 195].

SNMP realmente es un protocolo de supervisión de red, y las acciones de gestión las realiza el encargado de la red manualmente. En redes donde se necesitan respuestas rápidas (comparadas con la respuesta humana), la intervención humana no es lo ideal. Dado lo anterior una de las metas de la gestión de redes es, precisamente, automatizar los procesos de gestión

La "Gestión de red" tiene el propósito de: utilizar y coordinar los recursos para planear, ejecutar, administrar, analizar, evaluar, diseñar y extender las redes de comunicaciones para adaptarse al nivel de servicio requerido en todo momento, a un costo razonable y con capacidad óptima. El costo razonable se considera de la siguiente forma: ¿qué resulta más barato? ¿Tener la red con o sin gestión?

La gestión de redes tiene diversas funciones de gestión, de las cuáles cinco están plenamente identificadas o reconocidas por TMN (Telecommunication Management Network), mismas que se mencionarán posteriormente en el capítulo IV.

Una de las soluciones más aceptadas para resolver el problema de la gestión de red es mediante la arquitectura TMN, la cual ha adoptado el protocolo de CMIS/P de OSI. Aunque TMN es la solución más aceptada, aún no se ha implantado completamente.

I.2 Planteamiento del problema

Entre la diversidad de funciones de gestión de red, la tratada en esta tesis es la que tiene que ver con la Gestión de Fallas en una red de telecomunicaciones ATM.

Las fallas que pueden aparecer en una red son muy diversas, y se tratan a diferentes niveles; en consecuencia, la solución a estas fallas también se trata a distintos niveles. No es conveniente tratar de resolver las fallas en una red mediante un Sistema de Operación

(ver capítulo IV.4). Este trabajo mostrará que hay casos en que no es necesario pasarlos al sistema de operación, sino que se pueden resolver antes.

La interfaz Q3 es la que comunica a un dispositivo de la red con el Sistema de Operación, la cual está definida para el protocolo CMIS/P, en consecuencia, es compatible con TMN. Esta puede realizar la tarea de recuperar las fallas e informarle únicamente al sistema de operación que la falla ha sido recuperado en forma de una notificación. Si la Q3 realiza las operaciones anteriores, se convertirá en una Q3 que contempla la gestión de fallas, lo cual constituye la base de este trabajo.

I.3 Objetivo

- Estudio e investigación de los distintos protocolos para gestión de redes.
- Estudio de la plataforma ISODE-OSIMIS
- Estudio de la interfaz Q3 y del funcionamiento de una red ATM.
- Desarrollo de la funcionalidad para la gestión de fallas de una interfaz Q3 para redes ATM.

I.4 Organización de la tesis

Los siguientes 2 capítulos de esta tesis describen los protocolos de gestión SNMP y CMIS/P, las normas y recomendaciones para realizar gestión de red con estos protocolos pueden encontrarse en [RFC 1155, 1157, 1212.][CCITT X.710, X.711]. El tercero cubre TMN, que es una arquitectura que contempla Gestión de red en diferentes niveles. El capítulo IV describe algunas plataformas de gestión. Éstas son una herramienta para la integración de Gestión de red. En nuestro caso ISODE-OSIMIS es la plataforma empleada

para realizar operaciones de gestión, por lo que se trata de una manera más detallada. El capítulo V primero nos introduce a ATM, para describir posteriormente el desarrollo de la funcionalidad que contempla la gestión de fallas en la Q3 para ATM. En el capítulo VI se presentan algunas pruebas realizadas a un conmutador FORE ATM. Estas pruebas presentan la secuencia de traps que ocurren cuando se genera una falla específica. El último capítulo presenta las conclusiones y aportaciones de este trabajo de tesis.

I.5 Contexto del proyecto

La idea de este proyecto de tesis surge del Dr. Joan Serrat, quien trabaja para la UPC (Universidad Politécnica de Cataluña) y fue coordinador en España del recientemente terminado proyecto europeo MISA, por lo que el presente trabajo fue realizado con la colaboración de la UPC, el CITEDI y el CICESE. La Q3 para ATM fue implantada dentro del programa *qatm* para el proyecto MISA, utilizando la plataforma ISODE-OSIMIS y programada en C++, por lo que *qatm* contiene clases definidas e implantadas para realizar entre otras actividades los trabajos de la interfaz Q3. A estas clases que componen *qatm*, mediante este trabajo de tesis, se pretende agregar una clase más que sea la que se encargue de realizar la gestión de las fallas.

I.5.1 El proyecto MISA

El proyecto MISA (Management of Integrated SDH and ATM networks) se inicia a finales de 1995 con el objetivo de diseñar y desarrollar un sistema de gestión de red que, integrando la gestión de subredes de distinta tecnología (ATM y SDH), proporcione un servicio de gestión de conexiones de banda ancha, en un ambiente multidominio y de forma totalmente transparente al usuario.

La oportunidad del proyecto es enorme ya que el uso de las comunicaciones de banda ancha es cada vez mayor, la desregulación de las telecomunicaciones favorece la entrada de nuevos participantes y, hasta el momento, no existen mecanismos de gestión automatizada de redes de banda ancha.

Para lograr los objetivos del proyecto se concibe una arquitectura distribuida plenamente basada en el concepto de TMN (Telecommunication Management Network) de la ITU (International Telecommunications Union). Así pues, el sistema de gestión estará constituido por un conjunto de sistemas de operación denominados MISA OS, cada uno de los cuales tendrá funcionalidad en el ámbito de "gestión de red" y en el ámbito de "gestión de servicio" y que tendrán que cooperar entre sí, con los sistemas de gestión preexistentes y con los usuarios finales. Esta cooperación con su entorno se realiza mediante unas interfaces especializadas. La figura 1 muestra la arquitectura del Sistema de Operaciones (OS) de MISA.

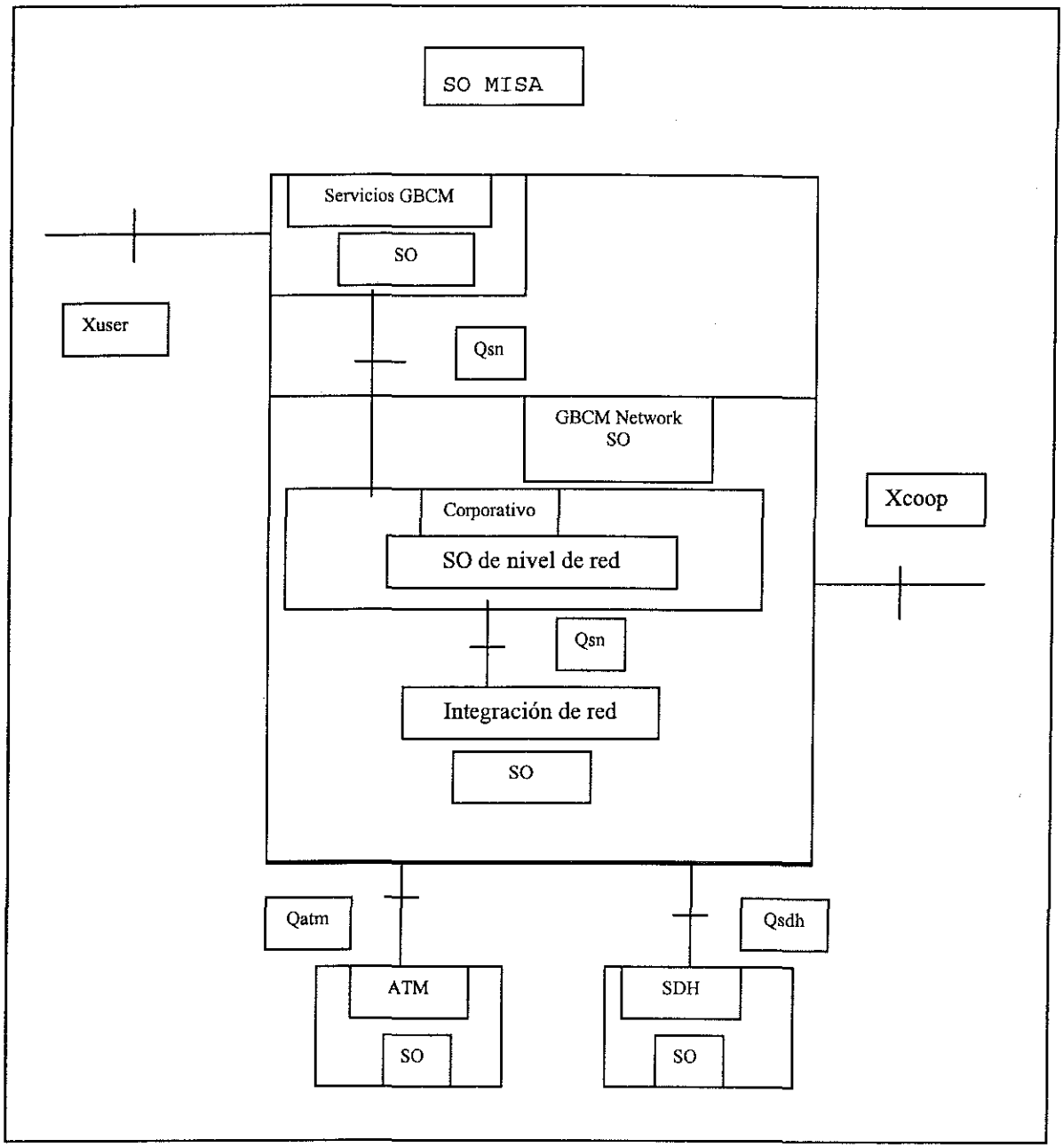


Figura 1. Arquitectura del Sistema de Operación MISA (MISA OS)

La funcionalidad se divide en los dos niveles antes mencionados; en realidad el MISA OS puede verse como dos sistemas de operación, uno a nivel de servicio y otro a nivel de red, interconectados mediante una interfaz Q (Qsn en la figura). Obsérvese asimismo las cuatro interfaces externas del sistema, dos de ellas de tipo X y las dos restantes de tipo Q. Las interfaces Xuser y Xcoop permitirán la cooperación del sistema con el usuario y con otros sistemas respectivamente. Las interfaces Qatm y Qsdh permitirán la cooperación con sistemas de operación preexistentes en redes basadas en tecnología ATM y SDH respectivamente.

Es importante notar que MISA no prescinde de los sistemas de operación que pudieran tener las redes ATM y/o SDH. Al contrario, el sistema MISA los utiliza para lograr sus objetivos. No sería técnica ni económicamente viable la implantación de un sistema de gestión que tuviera que prescindir del soporte que los fabricantes de equipos dotan a sus productos o de los sistemas que han tenido que desarrollar los operadores de redes para gestionar dichos equipos. Sin embargo, esta solución no está exenta de costos. Aunque los sistemas de operación de una determinada tecnología tengan funcionalidad a nivel de red (ATM NL-OS y SDH NL-OS) ésta no tiene que ser idéntica, ya que, ante la falta de normas suficientemente desarrolladas, dependerá principalmente del fabricante del producto y del operador que lo ha instalado. Por consiguiente, se hace necesario un proceso de adaptación que facilite a cualquier OS dependiente de tecnología (ATM o SDH) presentar la funcionalidad especificada en la interfaz Q correspondiente (Qatm o Qsdh). De su propia naturaleza se desprende que este proceso de adaptación es específico para cada sistema de gestión propietario que se desee conectarse a un OS de MISA.

I.6 Equipo utilizado para el desarrollo del proyecto

- Estación de trabajo SUN SPARC Ultra 1
- Sistema Operativo SOLARIS 2.5.1.
- Software: binutils 2.8.1, bison 1.25, gcc 2.7.2.2, libg++ 2.7.2, gawk 3.0.3.
- Conmutador ATM ASX200BX de ForeSystems.
- ISODE 8.0 y OSIMIS 4.0

Capítulo II

Protocolo simple de gestión de red (SNMP)

El término Simple Network Managment Protocol (Protocolo Simple de Gestión de Red, SNMP) es realmente usado para referirse a una colección de especificaciones para la Gestión de Red que incluye el protocolo mismo, las definiciones de las estructuras de los datos y los conceptos asociados.

El desarrollo de SNMP sigue un patrón histórico similar al desarrollo de toda la serie de protocolos de TCP/IP, de la cual es una parte.

II.1 Orígenes de TCP/IP

El punto de inicio de TCP/IP es en 1969, cuando el Departamento de Defensa de los Estados Unidos, a través de la Agencia de Proyectos de Investigación Avanzada, desarrollaron la primera red de conmutación de paquetes, ARPANET. El propósito de ARPANET fue el estudio de las tecnologías relacionadas con compartir los recursos de las computadoras. Muy pronto creció ARPANET, llegando a tener cientos de servidores y miles de terminales. Con equipo de distintos vendedores y el software necesario para cada vendedor, el problema de interconexión llega a ser tan grande como ARPANET.

El problema de interoperabilidad se solucionó, mediante un conjunto de protocolos que los investigadores de ARPANET desarrollaron y que llegaron a ser normas. Para 1970 se presentaron un conjunto de protocolos de TCP/IP, teniendo como base las normas militares del Departamento de Defensa de los Estados Unidos. Un inesperado desarrollo fue el crecimiento de aplicaciones no militares. Este crecimiento llegó a su fin aparentemente a

mediados de los 80s, cuando se hizo el esfuerzo por desarrollar un consenso internacional al rededor de OSI (Open System Interconnection, Interconexión de Sistemas Abiertos). A pesar de OSI, TCP/IP continuó creciendo rápidamente y su arquitectura es la norma más dominante actualmente.

La pregunta natural sería ¿por qué se prefirió una norma militar en lugar de una norma internacional?. La respuesta es simple: TCP/IP es un protocolo sencillo, maduro, que trabaja con un conjunto de protocolos que proporcionan interoperabilidad y alto nivel de desempeño, mientras que OSI provee mucho más riqueza de funcionalidad que TCP/IP, pero estas riquezas implican complejidad que hacen que su implantación sea más difícil que TCP/IP.

II.2 Los orígenes de la gestión de redes con TCP/IP

Cuando fue desarrollado TCP/IP, poco se pensó en la gestión de la red. En un principio, virtualmente todos los servidores y subredes conectadas a ARPANET se basaban en un ambiente que incluía programadores de sistemas y diseñadores de protocolos trabajando en algún aspecto de la investigación de ARPANET.

Durante la década de los 70s no hubo un protocolo de gestión como tal. La única herramienta utilizada para gestión fue el *Internet Control Message Protocol* (ICMP). ICMP provee un medio para transferir mensajes con control de enrutadores y servidores. ICMP esta disponible para todos los dispositivos que soportan IP.

El punto de inicio para proveer herramientas específicas para la gestión de redes fue el *Simple Gateway Monitoring Protocol* (SGMP) editado en noviembre de 1987. SGMP

provee un medio íntegro para la vigilancia de las pasarelas. Como la gestión de redes necesitó de herramientas de propósito general, tres propuestas de solución surgieron:

- *High-Level Entity Management System* (HEMS, Sistema gestor de entidad de alto nivel). Este tal vez fue la generalización del primer protocolo de Gestión de red utilizado en Internet.
- *Simple Network Management Protocol* (SNMP): Esta fue una versión mejorada de SGMP.
- *CMIP sobre TCP/IP (CMOT):* Este fue un intento para incorporar, con la máxima extensión posible, al protocolo CMIP (common management information protocol) a servicios y estructuras de base de datos normalizados por ISO para gestión de redes.

A principios de 1988, el *Internet Architecture Board* (IAB) revisó aquellas propuestas y aprovechó los desarrollos de SNMP como solución a corto plazo y CMOT como una solución a largo plazo. Se veía razonable que en un período de tiempo adecuado, las instalaciones de TCP/IP adoptarían los protocolos de OSI.

El tratar de juntar ambos protocolos resultó impráctico debido a la complejidad de los protocolos de OSI; por otro lado, SNMP es simple, éste no fue diseñado para trabajar con conceptos sofisticados, con lo que los protocolos siguieron desarrollándose independientemente.

II.3 Evolución de SNMP

Con el desarrollo de SNMP, al no tener la complejidad de OSI, muy pronto llegó a ser la norma *de facto* para gestión de redes. Mientras, la gestión de redes en OSI (CMIP)

continuó retrasándose.

El SNMP básico se usa ahora en un amplio espectro. Virtualmente todos los vendedores de computadoras, estaciones de trabajo, puentes, enrutadores y concentradores ofrecen en sus equipos agentes compatibles con SNMP. Se está trabajando y se ha progresado para lograr que SNMP se pueda usar en protocolos que no sean TCP/IP.

Han surgido numerosas propuestas para el mejoramiento de SNMP, tal vez la más importante de estas iniciativas sea el desarrollo de la capacidad de vigilancia remota en SNMP. La vigilancia remota (*remote monitoring* RMON) especifica definiciones adicionales básicas para la MIB (Management Information Base) de SNMP, así como funciones para explotar la nueva MIB (MIB de RMON). RMON da al gestor de la red la habilidad de supervisar subredes como un todo, más que como un dispositivo individual en la subred. Tanto vendedores como usuarios ven a RMON como una extensión de SNMP.

Hay un límite hasta donde SNMP puede ser extendido para simplificar nuevas definiciones y MIBs más elaboradas. Con el uso de SNMP en redes más grandes y sofisticadas, sus deficiencias llegan a ser más apreciables. Estas deficiencias tratan de subsanarse con SNMPv2.

II.4 Normas relacionadas con SNMP

El conjunto de especificaciones que definen SNMP y sus funciones relacionadas y bases de datos están comprendidas en las *Request for Comments* (RFCs), mismas que siguen un orden progresivo y continúan emitiéndose. Las RFCs relacionadas con SNMP

son las siguientes:

- Estructura e identificación de la gestión de información para redes basadas en TCP/IP (RFC 1155): Describe como son definidos los objetos gestionados contenidos en la MIB.
- Base de Información de Gestión (MIB) para redes basadas en TCP/IP Internets: MIB-II (RFC 1213): describe la gestión de objetos contenidos en la MIB.
- Protocolo Simple de Gestión de Red (SNMP) (RFC 1157): define el protocolo utilizado para la gestión de estos objetos.

II.5 Conceptos básicos de SNMP

SNMP es un protocolo diseñado para dar al usuario la capacidad para administrar remotamente una red de datos por encuesta y valores asignados en la terminal y eventos de supervisión de la red. En pocas palabras, SNMP trabaja de la siguiente manera: Éste intercambia información a través de mensajes conocidos como Unidad de Datos del Protocolo (Protocol Data Unit PDU). Un mensaje PDU puede ser visto como un objeto que contiene variables que tienen atributos y valores.

Hay 5 tipos de PDUs en SNMP empleados para supervisar una red: dos tratan con la lectura de los datos de la terminal (GetRequest y GetNextRequest), dos tratan con la escritura de datos en la terminal (SetRequest) y uno más llamado trap, el cual es usado para supervisar eventos en la red, tales como altas y bajas de elementos de red o fallas. SNMP está constituido por tres elementos, la MIB, el agente y el gestor.

II.5.1 Arquitectura de gestión de red TCP/IP

El modelo de Gestión de Red usado por TCP/IP incluye los siguientes elementos:

- Gestor.
- Agente gestionado o agente.
- Base de Información de Gestión (MIB) definida en ASN.1 (Abstract Sintax Notation One).
- Protocolo de gestión de red (SNMP).

El gestor es un dispositivo que funciona solo, pero éste puede implantarse en sistemas compartidos. En otro caso, el gestor sirve como la interfaz entre el administrador humano y el sistema administrador de la red.

El agente es un elemento activo del sistema de gestión de red. Plataformas claves tales como servidores, enrutadores y concentradores, pueden ser equipados con agentes SNMP tal que éstos puedan ser manejados por el gestor. El agente responde a las peticiones de información y acción del gestor.

Los recursos en la red pueden ser administrados por la representación de estos recursos como objetos de una manera abstracta. Cada objeto es, esencialmente, un dato variable que representa un aspecto del agente gestionado. A la colección de estos objetos se le conoce como base de información de gestión (managment information base MIB). La MIB de SNMP se define en ASN1 [Stalling William, 1996, pp. 425].

El gestor y los agentes interactúan mediante el protocolo de Gestión de Red. El protocolo utilizado para administrar redes TCP/IP es SNMP, el cual incluye las siguientes capacidades claves:

- get: habilita al gestor para recuperar el valor de los objetos del agente.
- set: habilita al gestor para poner valor a los objetos del agente.
- trap: habilita a un agente para notificar al gestor de eventos no deseados.

Para los servicios de get se utiliza el PDU GetRequest, este mismo PDU puede utilizarse para transmitir, tanto la primitiva GetRequest, así como las primitivas GetNextRequest y SetRequest. El PDU GetResponse, se encarga de la transmisión de la primitiva que lleva el mismo nombre de este. Los traps se transmiten mediante el PDU de trap (trap PDU en inglés).

II.5.2 Arquitectura del protocolo de gestión de red SNMP

SNMP fue diseñado para ser un protocolo que opera en la capa de aplicación, que es parte de las series de TCP/IP. Éste se propone para operar sobre un *User Datagram Protocol* (Protocolo de Datagrama de Usuario, UDP). La figura 2 muestra una configuración típica del protocolo SNMP.

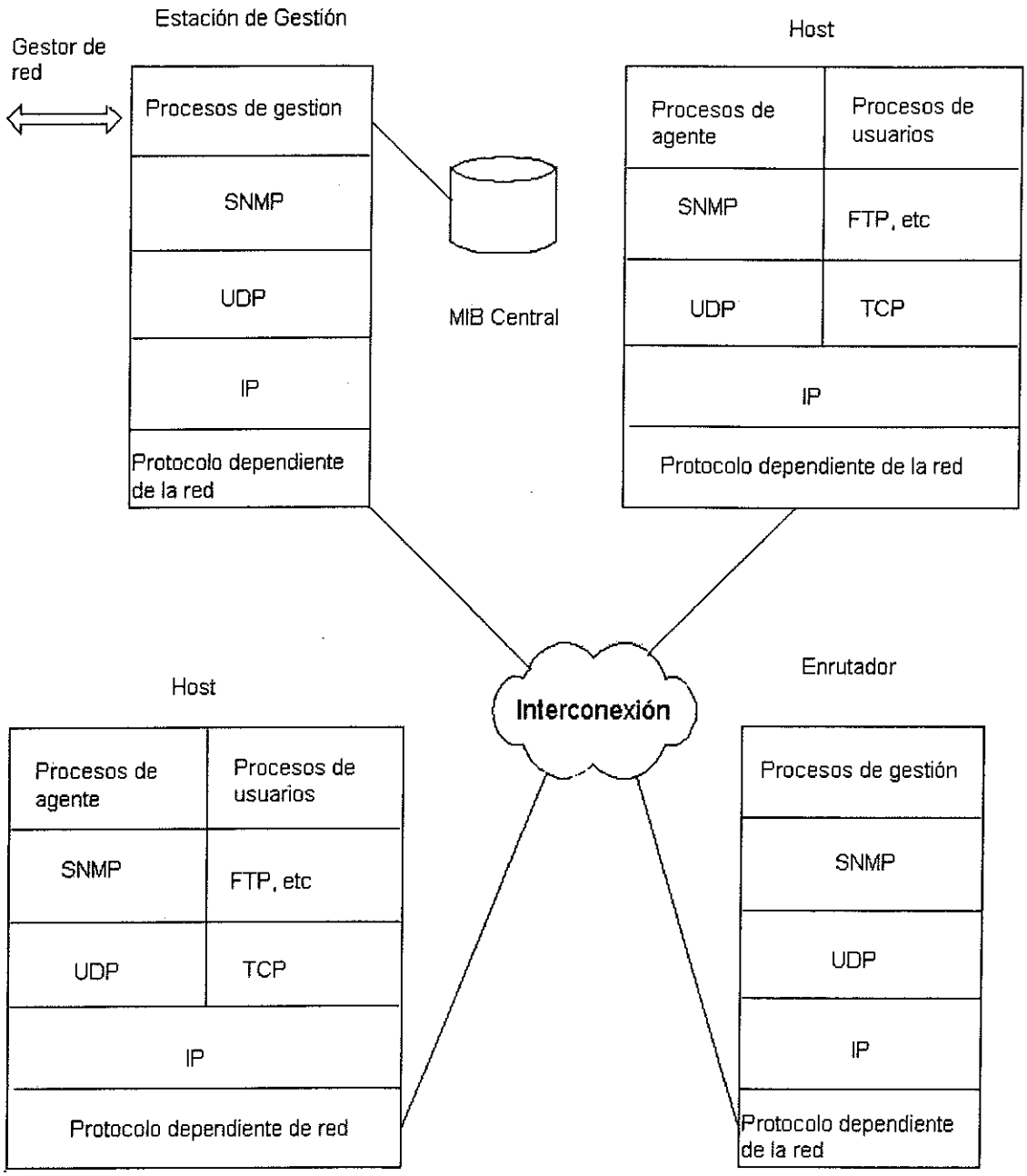


Figura 2. Configuración de SNMP

Para levantar o iniciar solo una estación de gestión, un gestor de procesos controla el acceso a la MIB central en la estación gestora y provee una interfaz para la gestión de la red. El gestor de procesos realiza la gestión de la red usando SNMP, el cual esta implantado en lo alto de UDP, IP y los protocolos relevantes dependientes de la red (ethernet, FDDI,

X.25, etc.).

Cada agente debe tener implantado también SNMP, UDP e IP. Además, un agente de procesos interpreta los mensajes SNMP y controla los agentes de la MIB, la figura 3 provee una visión del contexto del protocolo SNMP.

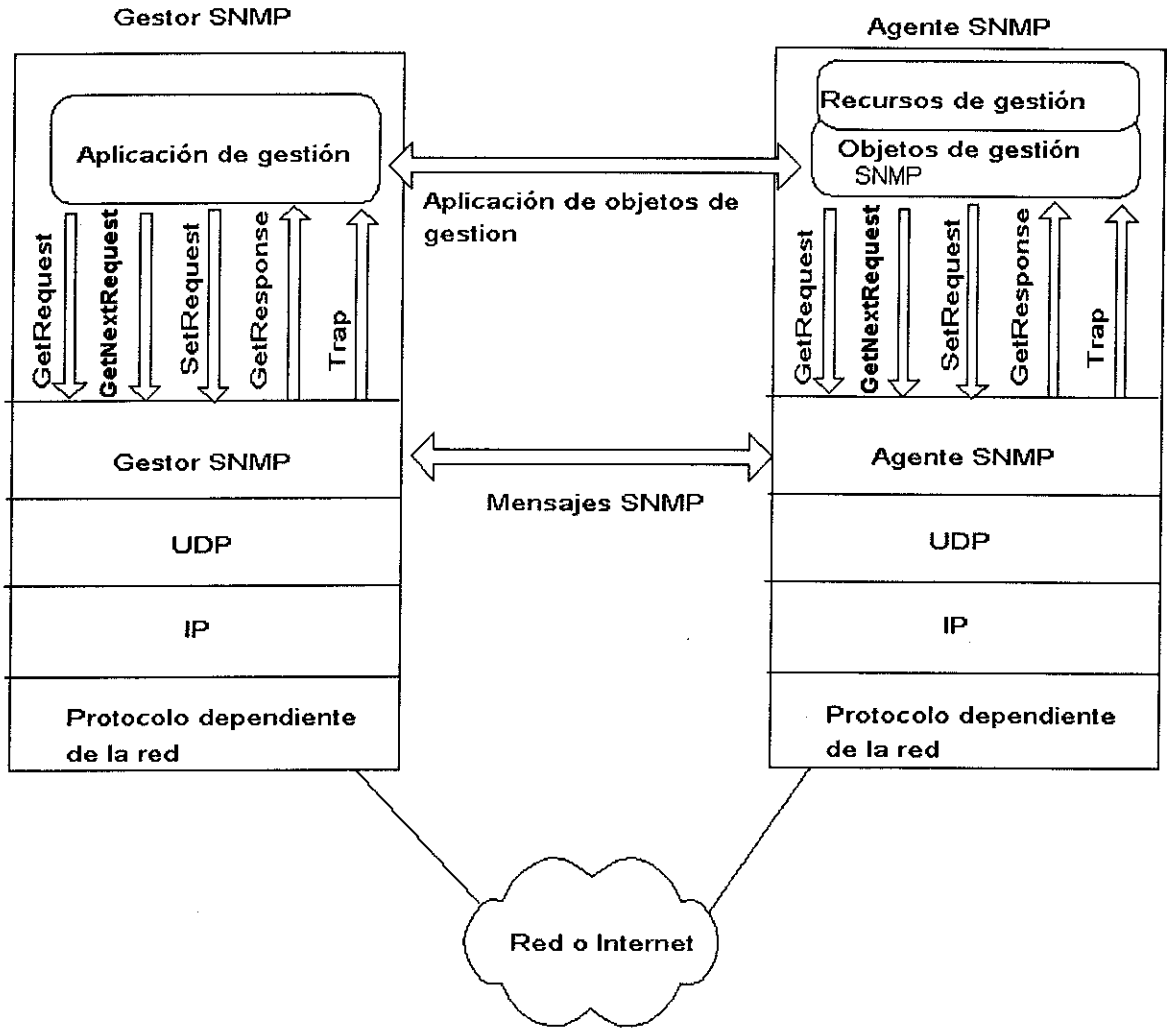


Figura 3.El papel de SNMP

Desde un gestor, se emiten tres tipos de mensajes: GetRequest, GetNextRequest y SetRequest. Los dos primeros son variaciones de la función get. Estos mensajes son recibidos por el agente, el cual debe responder con mensajes de conocimiento. Además, un agente puede emitir un mensaje de error (trap) en respuesta a un evento que afecte a la MIB

y al gestor de recursos principal.

II.5.3 Trap-encuesta directa.

Si un gestor es responsable de una gran cantidad de agentes, y cada agente mantiene una gran cantidad de objetos, puede llegar a ser impráctico regular para todos los agentes las encuestas a sus objetos gestionados. En lugar de eso, el SNMP y la MIB asociada utilizan una técnica referida como "*trap-directed polling*".

La estrategia es ésta: en un tiempo de inicio, y quizá a intervalos infrecuentes una vez al día por ejemplo, un gestor puede encuestar todo acerca de los agentes, las encuestas por parte del gestor no se estarían realizando en todo momento, sino en un periodo determinado, el responsable de informar el estado de los dispositivos o de los recursos sería el agente, este notificaría al gestor mediante traps, si algo inusual sucediera en la red.

La técnica de trap-encuesta directa puede resultar en un ahorro substancial en capacidad de la red y tiempo de proceso de agentes. En esencia, la red no carga a sí misma con información de gestión que no necesita.

II.5.4 Proxies

El uso de SNMP requiere que todos los agentes, así como las estaciones gestoras, puedan soportar una serie de protocolos comunes, tales como UDP e IP. Esto limita directamente la Gestión para tales dispositivos y excluye otros dispositivos, tales como algunos puentes o módems, que no soportan alguna parte de las series de protocolos TCP/IP. Además, hay numerosos y pequeños sistemas (computadoras personales, estaciones de trabajo) que no soportan TCP/IP, pero por otro lado no es deseable agregar

carga de SNMP, al agente lógico y a la MIB.

Para dispositivos que no tienen SNMP se desarrolló el concepto de *proxies*. Un proxie se encarga de traducir al protocolo SNMP si la petición viene de un dispositivo que no es compatible con SNMP, y también se encargará de traducir al formato propietario los comandos emitidos por el gestor SNMP.

Capítulo III

Gestión en OSI

III.1 Introducción

Él termino gestión OSI (Open System Interconection) es realmente utilizado para referirse a una colección de normas para la gestión de red que incluye a los servicios de gestión (Common Management Information Service CMISE) y al protocolo (Common Management Information Protocol CMIP).

Las normas para la gestión de red OSI, son extensas y numerosas [Stalling William, 1993, pp.371]. La primera norma relacionada con gestión de red, editada por ISO (International Organization for Standarization) fue ISO 7498-4, la cuál especifica la trama de gestión para el modelo de OSI.

Las normas para la gestión de red se han venido desarrollando por ITU-T (antes CCITT) y OSI desde principios de los años 80s, estas caen en 5 categorías generales:

1. Trama de gestión OSI [CCITT X.700, X.701]: Estas normas proveen una introducción a los conceptos de gestión, dentro de estos conceptos de gestión se definen las áreas funcionales de gestión tales como: Gestión de Fallos, Gestión de Configuración, Gestión de la Contabilidad, Gestión del Desempeño y Gestión de Seguridad.
2. CMIS/P [CCITT X.710, X.711]. Servicio de Información de Gestión Común (CMIS de sus siglas en inglés), provee servicios de gestión OSI, y Protocolo de Información de Gestión Común (CMIP), provee la información para el intercambio de los servicios CMIS.

3. Funciones de gestión de sistemas [CCITT X.730 hasta X.739]: Define las funciones específicas de la gestión de sistemas de OSI.
4. Modelo de información de gestión [CCITT X.720, X.721 y X.722]: Define la base de información de gestión (MIB), la cual contiene una representación de todos los objetos sujetos a ser gestionados dentro del ambiente OSI.
5. Gestión de capas [ISO 10733, ISO 737]: Define la información de gestión, servicios y funciones relacionadas para capas específicas OSI.

III.2 Modelo de gestión ITU-T/OSI

ITU-T/OSI utiliza el paradigma cliente servidor, en el que el cliente se denomina sistema gestor, el servidor sistema gestionado y una MIB (ver figura 4), recuérdese que CMIS/P son los encargados de realizar la gestión en ITU-T/OSI.

El sistema gestor, también conocido como gestor es el encargado de emitir los comandos de gestión hacia el agente, así como recibir notificaciones de éste. El sistema gestionado asume el papel de agente, lo que implica la recepción de notificaciones de gestión, operaciones y envío de notificaciones hacia y desde la MIB, que es la que contiene una representación de los recursos del dispositivo asociado.

Los elementos Agente - Gestor - MIB (ver figura 4) también son utilizado por SNMP, por lo menos en cuanto a arquitectura, SNMP y CMIS/P son iguales.

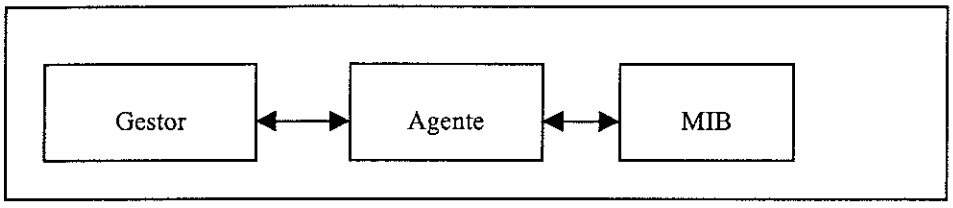


Figura 4. Elementos de gestión en OSI

Entre las diferencias más evidentes están:

- La MIB de SNMP se define con una pequeña parte de ASN.1, mientras que la MIB de CMIS/P [Stalling William 1993, pp. 389-428] se define con GDMO (Guidelines for the Definition of Managed Objects), que utiliza toda la potencialidad de ASN.1.
- CMIS/P esta diseñado para realizar las operaciones de gestión de red sobre la pila de OSI, mientras que SNMP sobre TCP/IP.

III.3 CMIS (Servicio de información de gestión común)

La función fundamental de la gestión en sistemas OSI es el intercambio de la información de gestión entre dos entidades (agente y gestor). Esta funcionalidad dentro de los sistemas de gestión OSI (Open system interconection), se refiere a *common management information service element* (CMIS, elemento de servicio de información de gestión común). CMISE se especifica en dos partes.

La interfaz con el usuario, provee específicamente los servicios. Este es llamado CMIS.

El protocolo, especifica el formato de la unidad de datos del protocolo (PDU) y sus procedimientos asociados, conocido como CMIP.

La figura 5, muestra el contexto de CMIS y CMIP. CMIS provee 7 servicios para realizar las operaciones de gestión, en la forma de primitivas de servicio. Para los servicios de operaciones de gestión, CMIS emplea CMIP para el intercambio de las PDUs (protocol data unit). CMIP activo cuenta con los servicios de elementos de servicio-operación-remota.

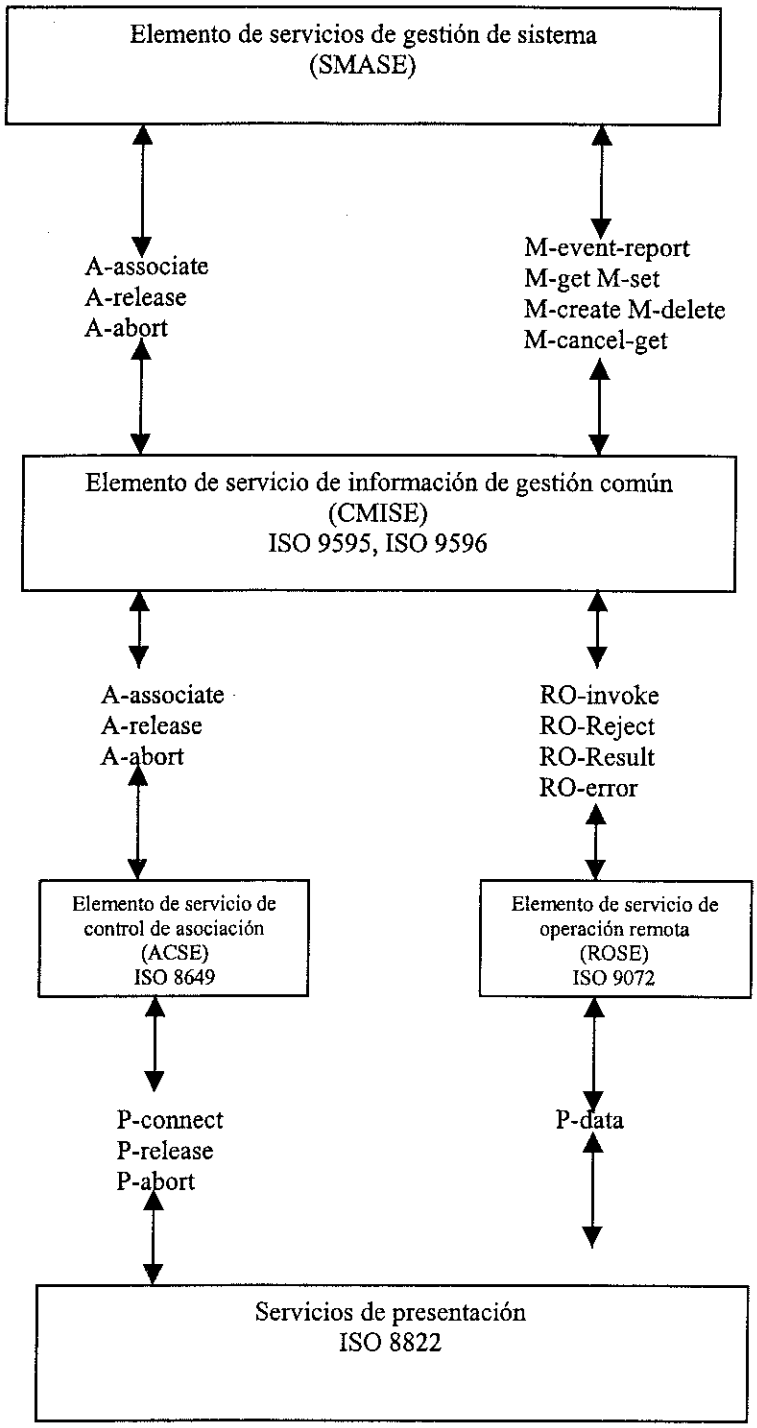


Figura 5. Servicios provistos por CMISE y servicios utilizados por CMISE

El servicio de información de gestión común define los servicios provistos por el sistema de gestión de OSI [ISO 9595]. Estos servicios están envueltos en los procesos de gestión para comunicación remota.

Los servicios CMIS son expresados en formas de primitivas que pueden ser vistos como comandos o procedimientos con parámetros. Estas se listan en la tabla 1. Los servicios son de dos tipos: peticiones de servicios confirmados, que requieren una confirmación de éxito o falla por parte de a quien se solicito el servicio, el otro servicio es el no confirmado el cual no requiere respuesta.

Tabla 1. Servicios CMISE

a)Servicios de notificación de gestión

Servicio	Tipo	Definición
M-EVENT-REPORT	Confirmado/no confirmado	Reporta eventos de gestión de objetos, para usuarios de servicio CMISE

b)Servicios de operación de gestión

Servicio	Tipo	Definición
M-Get	Confirmado	Solicitudes para recuperar información de gestión para un punto de servicio CMISE
M-Set	Confirmado/no confirmado	Solicitud de modificación de información de gestión para un punto de servicio CMISE
M-Action	Confirmado/no confirmado	Solicitud que un punto de servicio CMISE desempeñe una acción.
MCreate	Confirmado	Solicitud de que un punto de servicio CMISE, cree una instancia de un objeto de gestión.
M-Delete	Confirmado	Solicitud de que un punto de servicio CMISE, borre una instancia de un objeto de gestión
M-Cancel-Get	Confirmado	Solicitud de que un punto de servicio CMISE, cancele una solicitud previa.

Hay tres categorías de servicio relevantes en CMIS:

1. Servicio asociado: Los usuarios CMIS necesitan establecer una aplicación asociada para comunicación. Ellos cuentan con elemento de servicio de control asociado, para el control de las asociaciones aplicadas.
2. Servicios de notificación de gestión. Este servicio es utilizado para cubrir la información de gestión aplicable a una notificación. La definición de notificación y el consecuente comportamiento de las entidades de comunicación es dependiente de las especificaciones de los objetos de gestión que generan la notificación y su salida fuera de CMIS.
3. Servicios de operaciones de gestión. Hay seis servicios que son utilizados para cubrir información de gestión aplicable a operaciones de gestión de sistemas. La definición de la operación y el consecuente comportamiento de las entidades de comunicación son dependientes de las especificaciones de los objetos de gestión a los cuales la operación es direccionada y esta fuera del alcance de CMIS.

III.4 CMIP (Protocolo de información de gestión común)

El diseño de CMIP es similar a SNMP, por lo cual los PDUs son variables empleadas para vigilar una red. CMIP sin embargo contiene 11 tipos de PDUs comparados con los 5 de SNMP.

Mientras CMIS define los servicios para operaciones de gestión, CMIP define:

1. Los procedimientos para la transmisión de la información de gestión.
2. La sintaxis para los servicios de gestión CMIS.

CMIP esta definido en términos de la unidad de datos de protocolos CMIP que es el intercambio entre los puntos comunes de elementos de servicios de información de gestión (CMISEs) para transportar los servicios CMIS.

Para entender la operación CMIP, es necesario estar en el contexto de CMIS. La figura 5 muestra las relaciones entre CMIP y CMIS, como se ha discutido en la sección anterior, CMIS provee siete servicios para realizar las operaciones de gestión, en forma de primitivas de servicio. Además, los usuarios CMIS deben de ser capaces de establecer asociaciones para realizar las operaciones de gestión. Estos últimos servicios los provee ACSE (asociation control service element) y se provee para CMIS, aquí no interviene CMIP. Para los servicios de operación de gestión CMIS emplea a CMIP para el intercambio de PDUs. CMIP utiliza ROSE (remote operations service element) para transferir las PDUs. Tanto ACSE y ROSE se ocupan de los servicios de presentación [Stalling William 1993, pp 451-472].

La tabla II muestra los 11 PDUs que existen en CMIP. Hay tres tipos de información que transporta cada unidad de datos. Los argumentos, los resultados y los errores. Los argumentos son transportados en la PDU, estos son enviados por las primitivas CMISE. Los resultados y errores contienen información del desempeño de las operaciones de gestión. Estos valores son enviados de la primitiva de respuesta CMISE.

Tabla II. Unidades de datos CMIP.

Unidad de datos CMIP	Argumentos	Resultados	Errores
m-EventReport	**managedObjectClass, managedObjectInstance, everTime, everType, eventInfo**	———	———
m-EventReport-Confirmed	**managedObjectClass, managedObjectInstance, everTime, everType, eventInfo**	**managedObjectClass, managedObjectInstance, currentTime, eventReply**	**invalidArgumentValue, noSuchObjetClass, noSuchObjecInstance, processingFailure.**
m-Get	**base managedObjectClass, basemanagedObjectInstane, accessControl, synchronization, scope, filter, atributeIdList**	**managedObjectClass, managedObjectInstance, curretTime, atributeList**	**accessDenied, classInstanceConflict, complexityLimitation, getListError, invalidFilter, invalidScope, noSuchObjetClass, noSuchObjecInstance, operationCanceled, processingFailure, syncNotSupported**
m-Linked-Reply	**getResult, getListError, setResult, setListError, actionResult, processingFailure, deleteResult, actionError, deleteError**	———	———
m-Set	**base managedObjectClass, basemanagedObjectInstane, accessControl, synchronization, scope, filter, atributeIdList**	———	———
m-Set-Confirmed	**base managedObjectClass, basemanagedObjectInstane, accessControl, synchronization, scope, filter, atributeIdList**	**managedObjectClass, managedObjectInstance, currentTime, attributeList**	
m-Action	**base managedObjectClass, basemanagedObjectInstane, accessControl, synchronization, scope, filter, actionInfo**	———	———
mAction-Confirmed	**base managedObjectClass, basemanagedObjectInstane, accessControl, synchronization, scope, filter, actionInfo**	**managedObjectClass, managedObjectInstance, currentTime, actionReply.**	**accessDenied, classInstanceConflict, invalidFilter, invalidScope, noSuchObjetClass, noSuchObjecInstance, operationCanceled, processingFailure, syncNotSupported, noSuchAction, invalidArgumentValue, complexityLimitation, noSuchArgument**
m-Create	**managedObjectClass, objectInstance, accessControl, referenceObjectInstance, attributeList.**	**managedObjectClass, managedObjectInstance, currentTime, attributeList**	**accessDenied, classInstanceConflict, duplicateManagedObjectInstance, invalidAtributeValue, invalidObjectInstance, missingAttributeValue, noSuchObjectInstance, noSuchReferenceObject, processingFailure**
m-Delete	**baseManagedObjectClass, baseManagedObjectinstance, accessControl, synchronization, scope, filter**	**managedObjectClass, managedObjectInstance, curretTime.**	**accessDenied, classInstanceConflict, complexityLimitation, invalidFilter, invalidScope, noSuchObjectClass, noSuchObjectInstance, processingFailure, syncNotSupported.**
m-Cancel-Get-Confirmed	**getInvokedId**	———	**mistypedOperation, noSuchInvokeId, processingFailure.**

La tabla III muestra la transformación entre las primitivas CMISE y las unidades de datos de CMIP. Nótese que las unidades de datos de CMIP incorporan otros parámetros para las primitivas M-GET, M-SET, M-ACTION y M-DELETE de CMIS.

Tabla III Correspondencia entre las primitivas CMIS y las unidades de datos CMIP

Primitiva CMIS	Modo	Unidad de dato CMIP
M-EVENT-REPORT /request/indicaction	No confirmado	m-EventReport
M-EVENT-REPORT /request/indicaction	Confirmado	m-EventReport-Confirmed
M-EVENT-REPORT /request/indicaction	No aplicable	m-EventReport-Confirmed
M-GET request/indication	no nfirmado	m-Get
M-GET request/indication	Confirmado	m-Get
M-GET request/indication	No aplicable	m-Linked-Reply
M-SET request/indication	No confirmado	m-Set
M-SET request/indication	Confirmado	m-Set
M-SET request/indication	No aplicable	m-Linked-Reply
M-ACTION request/indication	No confirmado	m-Action
M-ACTION request/indication	Confirmado	m-Action
M-ACTION request/indication	No aplicable	m-Linked-Reply
M-CREATE request/indication	Confirmado	m-Create
M-CREATE request/indication	No aplicable	m-Create
M-DELETE request/indication	Confirmado	m-Delete
M-DELETE request/indication	No aplicable	m-Delete
M-DELETE request/indication	No aplicable	m-Linked-Reply
M-CANCEL-GET request/indication	Confirmado	m-Cancel-Get-Confirmed
M-CANCEL-GET request/indication	No aplicable	m-Cancel-Get-Confirmed

Capítulo IV

Red de gestión de las telecomunicaciones

IV.1 Introducción

La TMN (Red de gestión de las telecomunicaciones) ha sido especificada por los cuerpos normativos desde principios de los ochenta, viéndose como una posible solución al complejo problema de operación, gestión, mantenimiento y aprovisionamiento (OAMP) de las redes y los servicios de telecomunicaciones dentro y fuera del ambiente actual de múltiples proveedores.

La necesidad de gestionar equipo heterogéneo ha requerido la implantación de alguna forma de normativa; para poder integrar tecnologías heterogéneas se han desarrollado interfaces generales y flexibles. Para asegurar que las interfaces tengan consistencia suficiente para permitir un cierto nivel de gestión integrada, se han desarrollado un conjunto de recomendaciones guía [CCITT M.3020, M.3100, M.3010]. Este conjunto de recomendaciones constituye los principios de la TMN.

La intención primaria ha sido la de tener una red de sistemas de gestión enlazados entre ellos y con las diferentes redes de telecomunicaciones, todo lo cual, en conjunto, forma la TMN. Esta deberá supervisar y sintonizar constantemente las redes de telecomunicaciones y, en general, eliminar en lo posible la intervención del factor humano. Las interfaces están normalizadas, de tal manera que incluir equipo de nuevos proveedores no representa ningún problema, al menos en lo que a OAMP concierne. De igual manera, las nuevas tecnologías se podrán integrar con un mínimo de adaptaciones.

La Red de Gestión de las Telecomunicaciones (TMN) provee funciones y servicios

de gestión para redes de telecomunicaciones, y ofrece comunicación entre esta, la red de telecomunicaciones y los servicios. En este contexto se supone que una red de telecomunicaciones consiste tanto de equipo digital y analógico como del equipo de soporte asociado. Un servicio de telecomunicaciones en este contexto consiste de un número determinado de capacidades provistas para el usuario.

La función básica de TMN es proveer una arquitectura para realizar la interconexión entre varios tipos de Sistemas de Operación (OS) y/u otros equipos de telecomunicaciones para el intercambio de información de gestión utilizando una arquitectura acordada con interfaces normalizadas que incluyen protocolos y mensajes.

Entre las aplicaciones de TMN, tenemos las siguientes:

- Redes públicas y privadas, incluyendo tanto ISDN como BISDN, redes móviles, redes privadas de voz, redes privadas virtuales y redes inteligentes.
- La misma TMN.
- Terminales de transmisión (multiplexores, conectores de cruce, equipos de translación de canal, SDH, etc.).
- Sistemas de transmisión digital y analógica (cable, fibra, radio, satélite, etc.).
- Redes de área, local, amplia, metropolitana.
- Redes con conmutación de circuitos y de paquetes.
- Sistemas de operación y sus periféricos.
- Sistemas de soporte asociado (pruebas de módulo, sistemas de potencia, unidades de aire acondicionado, sistemas de alarmas de edificios, etc.).

TMN identifica cinco áreas de gestión estas son:

- Gestión de Desempeño.
- Gestión de Fallas.
- Gestión de Configuración.
- Gestión de Contabilidad
- Gestión de Seguridad.

Dentro de la arquitectura general de TMN hay tres aspectos básicos de la arquitectura los cuales pueden ser considerados por separado cuando se planea y diseña una TMN. Estos tres aspectos son:

- Arquitectura funcional TMN.
- Arquitectura de información TMN.
- Arquitectura física TMN.

IV.2 Arquitectura funcional TMN

TMN provee el medio para transportar y procesar la información relacionada con la gestión de las redes de las telecomunicaciones. La arquitectura funcional TMN está basada en los bloques funcionales que se muestran en la figura 6:

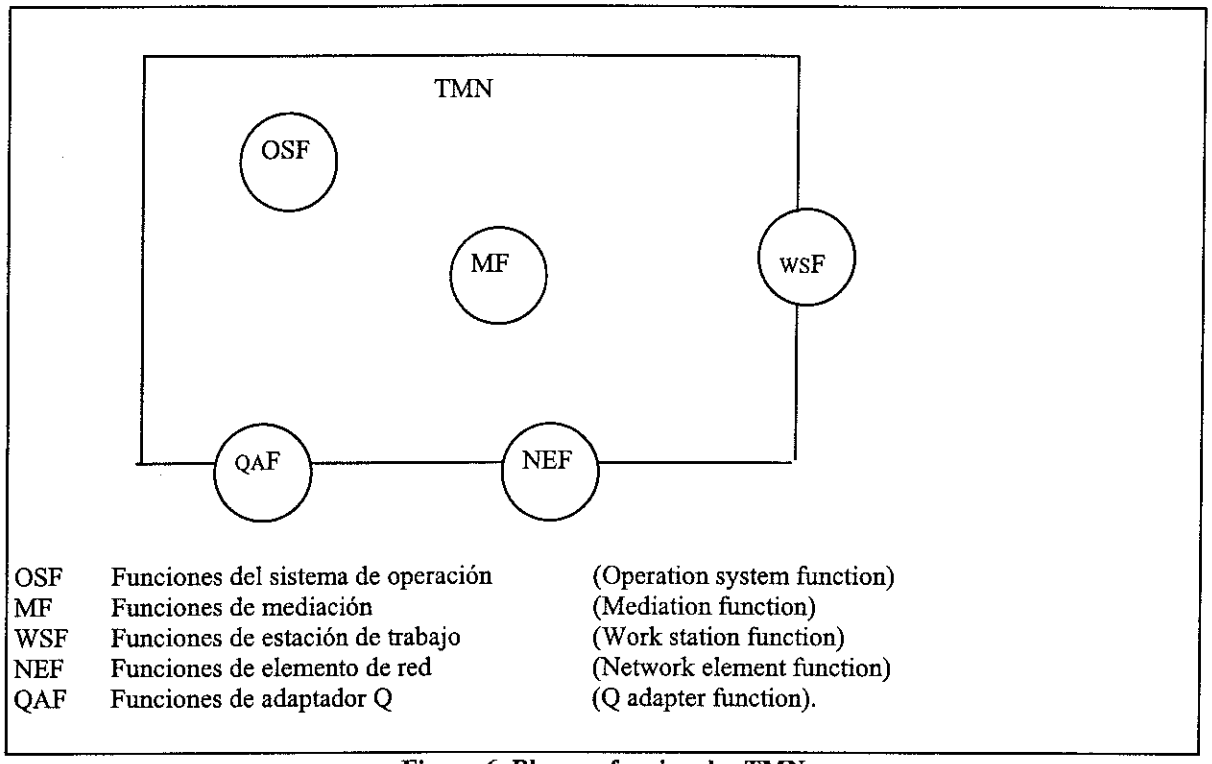


Figura 6. Bloques funcionales TMN

Los bloques funcionales proveen las funciones generales TMN las cuales habilitan las funciones de gestión. Una función de comunicación de datos (DCF) se utiliza para transferir la información entre los bloques funcionales TMN. A continuación se presenta una breve descripción de cada uno de los bloques funcionales.

- *OSF Función del sistema de operación (Operation system function).* El OSF procesa información relacionada a la gestión de las telecomunicaciones para los propósitos de supervisar, coordinar y/o controlar las funciones de telecomunicaciones incluyendo funciones de gestión.
- *MF Función de mediación (Mediation function).* Los bloques MF actúan sobre la información que pasa entre el OSF y NEF (o QAF). Esto puede ser necesario debido a que puede haber diferentes tipos de comunicación entre los mismos puntos de

referencia. Los bloques de mediación pueden almacenar, adaptar, filtrar, limitar y condensar la información.

- *WSF Función de estación de trabajo (Work station function).* La WSF provee el medio para interpretar la información TMN para la información de gestión del usuario. La WSF incluye soporte para la interfaz para el usuario humano. Tales aspectos de soporte no se consideran parte de TMN.
- *NEF Función de elemento de red (Network element function).* El NEF es un bloque funcional el cual se comunica con la TMN para propósitos de supervisión y/o control. NEF provee funciones de telecomunicaciones y de soporte, las cuales son necesarias para la gestión de las redes de telecomunicaciones. El NEF incluye funciones de telecomunicaciones las cuales son el sujeto de la gestión. Estas funciones no son parte de TMN pero están representadas para TMN por NEF.
- *QAF Función de adaptador Q (Q adapter function).* El bloque QAF se usa para conectar parte de TMN con entidades que no son TMN. La responsabilidad de QAF es trasladar la información entre los puntos de referencia TMN y los no TMN.

Los bloques funcionales están separados por puntos de referencia. La siguiente tabla muestra los bloques funcionales lógicos en términos de los puntos de referencia y su relación entre ellos.

Tabla IV Relaciones entre los bloques funcionales lógicos expresados como puntos de referencia.

	NEF	OSF	MF	QAF_{q3}	QAF_{qx}	WSF	No-TMN
NEF		q3	qx				
OSF	q3	q3, x[a]	q3			f	
MF	qx	q3	qx		qx	f	
QAF_{q3}		q3					m[b]
QAF_{qx}			qx				m[b]
WSF		f	f				
No-TMN				m[b]	m[b]	g[b, c]	

a) El punto de referencia x solo aplica cuando cada OSF esta en diferente TMN.
b) Los puntos m y g no son puntos de referencia de TMN.
c) El punto g está entre la WSF y el usuario humano.

Típicamente diferentes bloques funcionales pueden tener diferente grado de restricción en el alcance de la implantación del mismo punto de referencia. Las funciones provistas por los bloques funcionales TMN se pueden describir en términos de los componentes funcionales que comprenden éstos.

La tabla V muestra los bloques funcionales y sus componentes funcionales:

Tabla V Bloques funcionales y sus componentes

Bloque funcional	Componentes funcionales	Funciones de comunicación de los mensajes asociados
OSF	MIB, OSF-MAF(A/M), HMA	MCF_x, MCF_{q3}, MCF_f
OSF subordinado	MIB, OSF-MAF(A/M), ICF,HMA	MCF_x, MCF_{q3}, MCF_f
WSF	PF	MCF_f
NEFq3	MIB, NEF-MAF (A)	MCF_{q3}
NEFqx	MIB, NEF-MAF (A)	MCF_x
MF	MIB, MF-MAF (A/M), ICF, HMA	MCF_x, MCF_{q3}, MCF_f
QAFq3	MIB, QAF-MAF (A/M), ICF	MCF_{q3}, MCF_m
QAFqx	MIB, QAF-MAF (A/M), ICF	MCF_{q3}, MCF_m

PF Funciones de presentación
MCF Funciones de comunicación de mensajes
MIB Base de Información de Gestión
MAF Funciones de aplicaciones de gestión
ICF Funciones de conversión de información
A/M Agente/Gestor
HMA Maquina adaptadora para el usuario.

IV.3 Arquitectura de información TMN.

Dentro de esta arquitectura se considera una solución orientada a objetos para el intercambio de la información. Conceptos tales como agente y gestor, tal como se desarrollaron en la gestión OSI cae dentro de la arquitectura de información TMN.

Gestor: Es un programa de software situado dentro de una estación de gestión de red. El gestor puede preguntar a agentes, recibir respuestas de estos y actualizar variables, mediante comandos de gestión.

Agente: Es un programa de software situado en un dispositivo de red (ordenador, terminal, enrutador etc.). Un agente almacena datos de gestión, accesible a los agentes y manipula vía comandos de gestión para dar información de gestión de red.

La información de gestión puede ser considerada desde dos puntos de vista

a) *El modelo de la información de gestión.* El modelo de la información de gestión presenta una abstracción de los aspectos de gestión de los recursos de la red y las actividades de soporte de gestión. El modelo determina el alcance de la información que puede ser intercambiada en una manera normal (estándar). Esta actividad para el soporte del modelo de información toma lugar en el nivel de aplicación e involucra una variedad de funciones de aplicación de gestión tales como almacenamiento, recuperación y procesamiento de información. Las funciones involucradas en este nivel son referidas como "bloques funcionales TMN".

b) *El intercambio de la información de gestión.* El intercambio de la información de gestión involucra las funciones de comunicaciones de datos (DCFs), tales como una red de comunicaciones en una interfaz dada. Este nivel de actividad solo involucra

mecanismos de comunicación tales como la pila de protocolos (OSI).

IV.4 Arquitectura física TMN

La figura 7 muestra un ejemplo de la arquitectura simplificada para TMN.

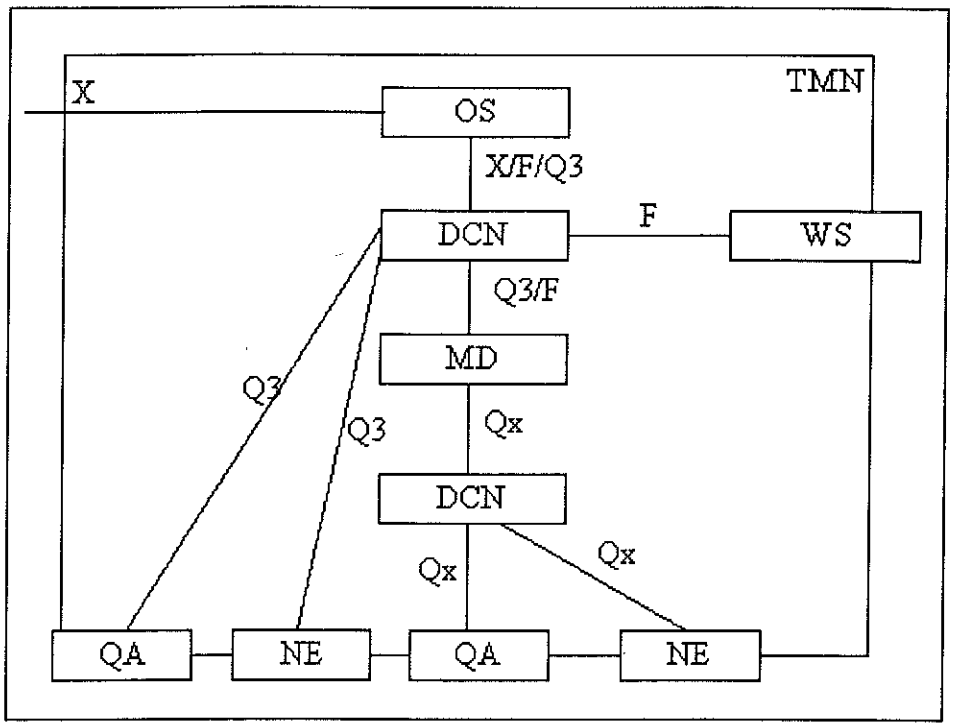


Figura 7. Arquitectura física TMN

Los componentes incluyen sistemas de operación (OS), redes de comunicación de datos (DCN), estaciones de trabajo (WS), elementos de red (NE), dispositivos de mediación (MD) y adaptadores Q (QA).

El sistema de operación (OS) constituye el sistema de supervisión o de control de la TMN. Provee administración de servicio para la supervisión y control de la red gestionada, interconexión con elementos de la red para reunir información y emitir comandos que puedan afectar el comportamiento de la red.

Los dispositivos de mediación (MD) son un aspecto nebuloso de la TMN. Se puede

asegurar que no se había desarrollado ninguno hasta 1996, por lo que, con frecuencia, lo que se ofrece en el mercado son en realidad adaptadores Q.

La misión de los adaptadores Q (QA) es la de conectar un sistema TMN con un sistema no TMN. Estos elementos son la esperanza de poder integras las redes existentes en la TMN.

El elemento de red (NE) es el único nodo ubicado en la red gestionada, es decir, la red de telecomunicaciones, y su principal tarea es el control de tráfico. Es, a su vez, el último origen o destino de la supervisión y el control de gestión.

La estación de trabajo (WS) es donde el usuarios interactúa con la TMN. Proporciona la función de presentación a aquel. Nótese que la WS como nodo en la TMN no coincide con el concepto de estación de trabajo del mundo informático.

Los nodos mencionados anteriormente se comunican a través de la red de comunicación de datos (DCN), que es el medio de transporte de TMN.

La interfaz más importante de la TMN es la Q3 [Angeles Valencia et al.,1995]. Es la única cuyas especificaciones están completas. Conecta un OS con un NE, o con un QA, o con un MD, o con otro OS en la misma TMN.

La QX es una interfaz Q3 subdesarrollada. Es como una Q3, pero con menor funcionalidad, y se utiliza cuando los costos o los requerimientos no justifican el uso de una Q3, aunque no existe un consenso acerca de que tipo de funciones pueden obviarse.

La interfaz X se usa para comunicar OSs en diferentes TMNs, o un OS de una TMN y un OS de una no TMN compatible con una interfaz orientada a TMN. Estas interfaces son de interés general, pero, dada su complejidad, muy pocas han sido especificadas.

La interfaz F conecta a la WS con los demás nodos. Muy poco esfuerzo se ha

realizado en la especificación de este tipo de interfaz.

IV.5 SNMP y CMIS/P

Existen dos protocolos principales para Red de Gestión de las Telecomunicaciones, uno es SNMP y el otro CMIS/P, siendo CMIP el que se escogió para TMN.

SNMP trabaja en modo orientado a no-conexión, siendo una buena opción para la gestión de redes donde los mecanismos de transporte son el medio primario de la administración del transporte de tráfico.

CMIP, por otro lado, sacrifica la simplicidad de SNMP, para proporcionar servicios más sofisticados por los agentes. El gestor puede hacer simples peticiones a los agentes que pueden causar que éstos ejecuten una sofisticada secuencia. Los eventos gestores son manejados a través de un conjunto de normas para eventos de notificación. Relaciones entre gestor y agente pueden operar tanto en el modo orientado a conexión como en el no-orientado a conexión.

En la tabla VI se muestra un resumen de diferencias entre ambos protocolos de gestión.

Tabla VI CMIS/P vs SNMP.

Modelo CMIS/P (TMN)	SNMP
Las clases de objetos son colecciones de propiedades asociadas con un recurso y son reutilizables	Los tipos de objetos son datos atómicos o tablas y no son redefinidos.
Las clases de objetos se pueden especializar usando el concepto de herencia	No existe el concepto de herencia
Diseñado para correr sobre la pila de OSI	Diseñado para correr sobre TCP/IP
Las clases pueden tener atributos opcionales y coexistir con atributos por omisión	Todas las variables dentro de un grupo de objetos (tales como una tabla), sus atributos son por omisión
No hay restricción en los tipos ASN.1 para el uso especifico del intercambio de información	Construcciones simples de ASN.1 y se restringe al tipo básico por la definición de la sintaxis.

Capítulo V

Plataformas de gestión de red

V.1 Introducción

Desde sus comienzos, la gestión de la red ha pasado de gestionar la red física, cables, puentes, enrutadores y concentradores, a proporcionar una multitud de servicios, como administración de la red, diagnósticos y control, rendimiento de la red, análisis de tendencias, gestión de la configuración y seguridad. La instalación de una plataforma de gestión de red de un fabricante de hardware puede garantizar la gestión fiable de este e informar del rendimiento de la red. Esto se consigue mediante el protocolo de gestión propietario. El problema de los protocolos propietarios es que sólo gestionan equipos específicos del fabricante. La integración y gestión de equipos de trabajo en red de otros fabricantes en estas plataformas puede ser difíciles, tal vez hasta imposible. En algunos casos, esto obliga al administrador de red a comprar varias plataformas para gestionar un entorno de trabajo en red heterogéneo.

La industria apuesta por una arquitectura de sistema abierto. Un entorno de computación cliente/servidor distribuido por red ha contribuido a la evolución de las redes, que se han convertido en la piedra angular de la infraestructura de computación colectiva. Las redes de empresa, vistas ahora como una parte clave del entorno de computación colectiva, provocan más de una complicación a los gestores de la red para controlar, diagnosticar y analizar sus redes. La necesidad de ver los problemas en el momento de producirse ha dado paso a una aspiración mayor, identificar los problemas y corregirlos automáticamente, y más aún, minimizar los problemas potenciales. Entre otros objetivos

clave se incluye la posibilidad de establecer tasas de umbral en los segmentos LAN y WAN para generar informes de las violaciones, ver las cargas de tráfico de un vistazo, recoger y archivar los datos para el análisis de tendencias. En este escenario también está incluida la integración de un sistema de notificación que avisaría al personal de mantenimiento por medio de correo electrónico o teléfono automáticamente cuando se produjeran problemas, cargas excesivas u otras actividades que necesitaran de la intervención del gestor.

Puede que parezca mucho pedir a una plataforma de gestión de red, pero con la necesidad constante de información instantánea y las crecientes demandas de ancho de banda, esto plantea una tarea necesaria y monumental a los gestores de errores actuales.

Para enfrentar a los retos anteriores la industria ha creado plataformas de gestión de red, que se ven ahora como solución de gestión de empresa. Estas plataformas de gestión utilizan una arquitectura de sistemas abiertos y protocolos de gestión basados en normas para integrar hardware y sistemas multivendedor. Dos de estos protocolos de gestión son SNMP y CMIP, existen además una variante de CMIP que se ejecuta sobre TCP/IP llamada CMOT (Common Management Protocol Over TCP/IP).

En los últimos años se han hecho grandes progresos en las plataformas de gestión de red. Aunque no se han definido todas las especificaciones de gestión de la red, los usuarios tienen la imperiosa necesidad de implementar una solución; una que evolucione con las normas según se vayan definiendo y pueda emigrarse completamente a las nuevas normas. Conseguirlo todo será un compromiso importante para cualquier vendedor. Así, una solución completa de gestión de empresa puede venir de una multitud de otros vendedores de software que se integre con la plataforma principal de gestión para realizar funciones específicas. Estos paquetes de software de otros vendedores pueden utilizar o no

los protocolos de gestión normalizados. Si no lo hacen, la plataforma de gestión de empresa debe encontrar un medio alternativo de comunicarse con estas aplicaciones por medio de pasarelas de protocolo externas o Interfaces de programación de aplicaciones (API). Las verdaderas plataformas deben poder incorporar estos otros vendedores.

Al pensar en una plataforma de gestión se debe de asegurar que esta sea una solución que se integre completamente con los sistemas elegidos. Por ejemplo, si se dispone de una plataforma de gestión que pueda identificar los problemas de la red e informar sobre ellos, puede elegirse un sistema de etiquetado de problemas que automáticamente avise a un reparador cuando se detecte un problema. También se podrían integrar una base de datos de gestión de activos, que podría incluir posiciones de los dispositivos, números de serie, números de revisión del software, conexiones de red y cualquier otro dato pertinente. Al emitir la etiqueta del problema, todos estos datos se imprimirían automáticamente.

Otras funciones que deben buscarse son las herramientas de gestión del sistema que permitan que la estación de gestión supervise la utilización de la memoria y el espacio en disco, la salud del procesador en la CPU, las aplicaciones disponibles y las violaciones de seguridad en los sistemas. Estas herramientas permitirán establecer umbrales y generar informes sobre ellos, y llevar al cabo acciones automatizadas. Una plataforma de gestión de red que incorpore la mayoría de estas funciones puede parecer demasiado cara; no obstante, con el aumento del valor de los recursos de información, no se permite no optimizar la gestión de estos recursos. El tiempo medio de indisponibilidad de una red en un entorno de gran negocio se estima que cuesta entre $5,000 y $6,000 dólares por hora productiva pérdida [Sheldom Tom, 1997], lo anterior justifica el pensar en una plataforma de gestión.

Existen muchas soluciones en el mercado actual. Algunas de las plataformas de gestión de red líderes en la industria: Gestor de nodos de red de Open View de Hewlett Packard; Polycenter de Digital Equipment; Spectrum de Cabletron Systems; NetView/6000 de IBM; Gestor de SunNet de SunConnect, todas las plataformas anteriores se pueden conseguir por algunos miles de dólares, una plataforma de dominio público y que será la que se describa con detalle posteriormente es OSIMIS.

Una plataforma de gestión de red va a ser una herramienta con la que se puede contar para mantener la integridad de una red. Una plataforma de red puede ser cara, pero a cambio ofrece la flexibilidad de implementar las soluciones en fases (el tiempo de indisponibilidad de la red supone perder ingresos). Las plataformas de gestión de red se han creado porque los gestores de red necesitan de estas herramientas para mantener sus crecientes redes heterogéneas activas y en funcionamiento. La gestión de la red es más que un seguro; es la inversión en el mantenimiento de la infraestructura de la red.

V.2 Las plataforma ISODE y OSIMIS

OSIMIS (OSI Management Information Service) es una plataforma de gestión orientada a objetos basada en el modelo OSI [CCITT X.701] e implementada principalmente en C++ [Pavlou George et al. 1995]. Esta provee un ambiente para el desarrollo de aplicaciones de gestión ocultando detalles de los principales servicios de gestión a través de los APIs (Applicattion Program Interfaces) orientados a objetos, permitiendo diseñar e implementar de una forma más sencilla pero más potente los mecanismos de acceso a los servicios y protocolos de gestión.

En ella se utiliza el modelo gestor/agente y la noción de objetos gestionados. Sin embargo, las aplicaciones de gestión pueden llevar al cabo tanto funciones de agente como de gestor. También suministra diferentes métodos para interaccionar con los recursos reales. Además, ofrece la posibilidad de combinar la potencia de la gestión OSI con elementos de red basados en el protocolo SNMP de Internet, ampliamente extendido. Para ello, implementa un agente "proxy" que es capaz de traducir las peticiones CMIP a SNMP.

OSIMIS utiliza ISODE (International Standardization Organization Development Environment) como mecanismo de comunicaciones implícito.

ISODE es una plataforma para el desarrollo de servicios OSI y sistemas distribuidos [Pavlou et al. 1995]. Proporciona las capas más altas de la pila OSI siguiendo las recomendaciones más relevantes de ISO/ITU-T e incluye herramientas para la manipulación de ASN.1 y para la generación de operaciones remotas. Además, incluye dos aplicaciones OSI fundamentales: la realización del *Directory Service* y del *File Transfer* (FTAM). ISODE está implementado en lenguaje de programación C y puede trabajar sobre la mayoría de las versiones de sistemas operativos UNIX. ISODE no proporciona ningún protocolo de red ni inferiores (como X.25, CLNP), pero descansa en las implantaciones para estaciones de trabajo basadas en UNIX a las cuales tiene acceso a través del Kernel. Los protocolos de las capas más altas implantados son los protocolos de transporte, sesión y presentación del modelo de 7 capas de OSI. También se proporcionan Elementos de Servicio de la capa de aplicación (Application layer Service Elements ASEs) en forma de bloques construidos para los servicios de niveles más altos, como pueden ser Elemento de servicio de control asociado, elemento de servicio de operación remota y elemento de servicio de transferencia segura. (Association Control Service Element, Remote

Operations Service Element y Reliable Transfer Service Elements o sus siglas en inglés: ACSE, ROSE y RTSE). Éstos, juntamente con el soporte al ASN.1, se utilizan para implementar los servicios de los niveles más altos. Desde el punto de vista de la ingeniería, la pila de ISODE es un conjunto de bibliotecas unidas a aplicaciones que las utilizan.

La manipulación de ASN.1 es muy importante en aplicaciones distribuidas OSI. La solución de ISODE para interfaces programables (API's) se basa en una abstracción fundamental conocida como *Presentation Element* (PE). Se trata de una estructura de C genérica capaz de describir de forma recursiva cualquier tipo de dato ASN.1. También proporciona un compilador de ASN.1 conocido como pepsy, que produce representaciones concretas, por ejemplo estructuras en C que corresponden a tipos en ASN.1 y también codifica/descodifica rutinas que las convierten en PE's y viceversa. La capa de presentación convierte PE's a cadenas de datos de acuerdo con las reglas de codificación (BER). Se debe observar que X/Open ha definido una API para la manipulación de ASN.1 conocida como XOM (XOpen) que, aunque similar en principio a la de ISODE, es muy diferente sintácticamente. Es posible la traducción entre las dos y tal tipo de solución se ha utilizado para el desarrollo de aplicaciones OSIMIS sobre XOM /XMP.

Esta plataforma ISODE proporciona:

- API's orientadas a objetos de alto nivel implementadas en forma de bibliotecas.
- Herramientas proporcionadas de forma separada que soportan las API's anteriores.
- Aplicaciones genéricas: visualizador, pasarela, servicios de directorio.

Los servicios que soporta ISODE son:

- Los protocolos de las capas de transporte, sesión y presentación, incluyendo una versión de éste último, la RFC 1006, que puede operar directamente sobre TCP/IP.

- Los elementos ACSE y ROSE (Association Control y Remote Operation Service Element).
- La FTAM (File Transfer Access and Management) y el DASE (Directory Access Service Element).
- La herramienta pepsy, que es un compilador de ASN.1 a C.
- La herramienta rosy, que es un generador de operaciones remotas.
- Un servicio FTAM para el sistema operativo UNIX.
- Una implantación completa de servicios de directorio, incluyendo un agente DSA (Directory Service Agent) así como varios DUA's (Directory User Agents).
- Un puente (bridge) de servicio de transporte que permite interoperar con aplicaciones de diferentes tipos de redes.

V.2.1 Componentes y Arquitectura de OSIMIS

La plataforma OSIMIS utiliza los siguientes componentes de ISODE:

- El compilador de ASN.1 *pepsy*, que ofrece una API equivalente a la de X/Open XOM. El uso del compilador es necesario porque en la gestión OSI, se utiliza ASN.1 para describir las *Protocol Data Unit* (PDU's) de CMIP y para describir la sintaxis de los valores de atributos, acciones y notificaciones de los objetos gestionados.
- Las API's *Association Control / Remote Operation Service*, que se utiliza para implementar el CMIP.
- La implantación del *Directory Service* de OSI que también incorpora ISODE (QUIPU) para el enrutamiento de aplicaciones. Es opcional, por lo que en el caso por omisión utiliza las tablas locales.

- Utiliza también parte del código de QUIPU que manipula sintaxis ASN.1, a través del uso de tablas de sintaxis e identificadores de objetos.

Por otra parte, los servicios que ofrece OSIMIS son:

- Implantación de la interfaz CMIS/P utilizando el ACSE, él ROSE y las herramientas de tratamiento de ASN.1 de ISODE.
- Implantación del protocolo SNMP de Internet sobre UDP de UNIX utilizando las herramientas de tratamiento de ASN.1 de ISODE.
- Soporte de alto nivel que encapsula las sintaxis ASN.1 en los objetos C++.
- Un meta-compilador de ASN.1 orientado a objetos que utiliza el compilador *pepsy* de ISODE. Compila de ASN.1 a C.
- Un mecanismo de coordinación que permite estructurar una aplicación en el sentido que funcione con el mecanismo de conducción de eventos (*event-driven*).
- Un servicio de soporte a la presentación que es una extensión del mecanismo de coordinación para utilizar mecanismos basados en X-Windows.
- El sistema gestionado genérico (GMS, Generic Management System) que es un agente OSI y que ofrece API's de alto nivel para implementar nuevas clases de objetos gestionados, una biblioteca de atributos genéricos, notificaciones y funciones de gestión de sistemas.
- Un compilador de GDMO (Guidelines for the Definition of Managed Objects) que complementa al GMS.
- Las MIB's Remota y Oculta que son API's con funciones de gestor.

- El soporte de directorio que ofrece servicios de enrutamiento de aplicaciones así como de *Location Transparency.*
- Una pasarela genérica entre aplicaciones SNMP y CMIS realizado mediante un traductor de MIB's de Internet a OSI.
- Diversas aplicaciones de gestión genéricas: visualizador de MIBs entre otros.

Los servicios y la arquitectura de OSIMIS se muestran en la figura 8:

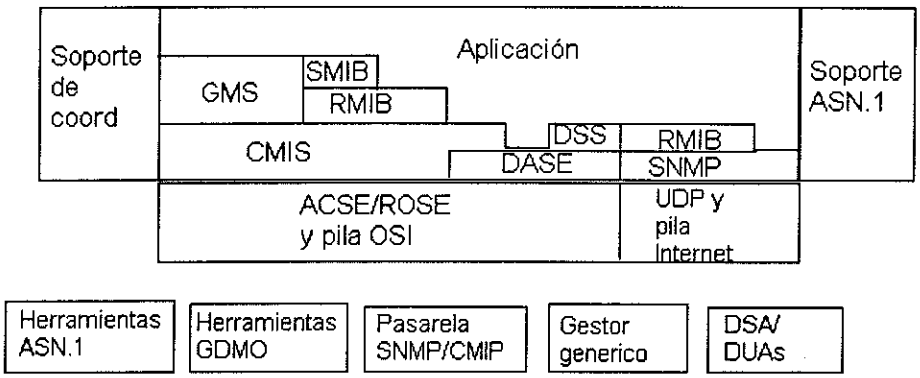


Figura 8 Arquitectura de las capas de OSIMIS y aplicaciones genéricas

La figura 8 muestra aplicaciones (GMS, SMIB, RMIB, etc.) que son programas y bibliotecas (ACSE/ROSE y pila OSI, UDP y pila internet).

La parte baja de la figura son las aplicaciones genéricas; de estos las herramientas ASN.1 y GDMO son esenciales para implementar el soporte de nuevas MIBs. GMS, SMIB, RMIB, CMISE, DASE, DSS y SNMP son los programas (APIs) que pueden ser utilizados por cualquier aplicación. En la práctica la mayoría de las aplicaciones utiliza Sistema de Gestión Genérico (GMS) y la MIB remota cuando actúan en el papel de agente o gestor.

OSIMIS ofrece los servicios más altos de la pila de OSI, es decir, los de la capa de aplicación. La plataforma de ISODE esta por abajo de estos servicios, ISODE ofrece los servicios en los niveles de la capa de presentación y aplicación; en la figura 9 se muestra la ubicación de OSIMIS e ISODE con respecto al modelo de referencia de OSI.

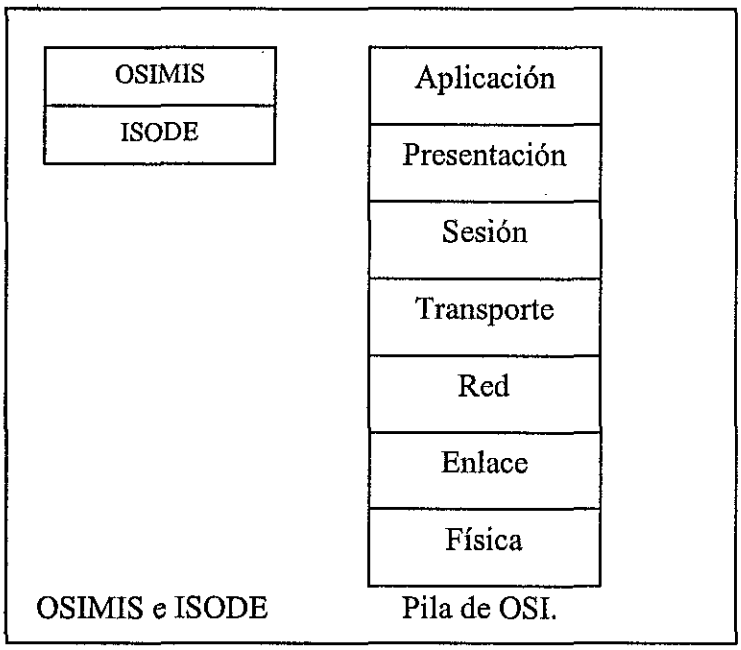


Figura 9. Ubicación de OSIMIS e ISODE en la pila de OSI.

Capítulo VI

Desarrollo de la Interfaz Q3 que contempla la gestión de fallas.

En este capítulo se describirá como se desarrolló la funcionalidad para gestionar las fallas en una red de ATM mediante la Q3, además de una descripción más detallada del problema a resolver, pero antes de hacer esta descripción se describirá el funcionamiento del modo de transferencia asincrono ATM.

VI.1 ATM

VI.1.1 Introducción

El desarrollo de la tecnología ATM (Asynchronous Transfer Mode), así como su acelerada adopción en el mercado de las Telecomunicaciones se debe principalmente a importantes cambios en las estrategias de negocio de organizaciones y a los avances tecnológicos ocurridos en la década de los 90.

El ámbito de los negocios, el aplanamiento de los organigramas dentro de las empresas sometidas a un proceso de Reingeniería, reemplazando la estructura jerárquica de varios niveles, traen como consecuencia la sustitución de arquitecturas centralizadas basadas en grandes computadoras por el procesamiento distribuido de una LAN. Estos cambios organizacionales influyen en los sistemas de información de empresas. Conceptos tales como arquitectura cliente-servidor, groupware, Intranet, videoconferencia de escritorio, bases de datos distribuidas, aplicaciones multimedia y el correo electrónico, entre otros, forman parte de esta tendencia.

Lo anterior genera la necesidad de aumentar el ancho de banda de las redes de Telecomunicaciones para lo cual el uso de ATM es muy atractivo ya que esta tecnología transmite información a velocidades que van del orden de los Mbps a los Gbps, entre otras ventajas

Otro efecto del cómputo distribuido es el considerable aumento en el número de miembros dentro de una organización con la necesidad de conectarse a la red corporativa. Por tal motivo es necesario el uso e interconexión de redes LAN, redes para Campus (CAN), redes Metropolitanas (MAN), redes de Banda Amplia (WAN) o una Red Global (GAN).

Además, dentro de la tendencia del procesamiento distribuido, es posible que varios miembros de la organización requieran acceso remoto desde su casa para realizar su trabajo, o sea personal en constante movimiento con igual necesidad de acceso a la información de la compañía, para la cual también se podría utilizar ATM, ya que permite una integración natural con la mayoría de las tecnologías de red existentes y es capaz de soportar gran cantidad de tráfico generado por los múltiples usuarios de la organización.

La capacidad de integración de voz, datos, imágenes y video, junto con la asignación dinámica del ancho de banda de la tecnología ATM, la colocan como una opción muy atractiva para aprovechar al máximo la infraestructura de fibra óptica con la que cuentan las compañías telefónicas y los proveedores de servicios en el mundo (carriers), permitiéndoles ofrecer nuevos servicios digitales de alta velocidad, como videoconferencia o conectividad LAN a 100Mbps.

En el caso de redes privadas, las características de la tecnología ATM mencionadas permiten consolidar el tráfico de una organización, eliminando los gastos operativos y la complejidad de sostener una red especializada para cada tipo de tráfico.

VI.1.2 Definición

El modo de Transferencia Asíncrono (ATM) se define como una tecnología para la transferencia de información entre redes de datos. Esta tecnología, relativamente nueva, tiene algunas características que hacen que se vislumbre como la tecnología del futuro; tecnología que ha de sustituir paulatinamente a las utilizadas actualmente en redes de cobertura amplia.

Pero, ¿es ATM realmente la panacea que viene a resolver los problemas de interconectividad con que nos encontramos actualmente, o es sólo la tecnología de moda de la que todo mundo habla y que después desaparece bajo la sombra de la nueva?

La tabla VII muestra el ancho de banda típico para algunas aplicaciones cliente-servidor, donde se aprecia claramente la necesidad de grandes anchos de banda para transmisión de información.

Tabla VII Aplicaciones y sus anchos de banda [GS Communications, 1996]

Tipo de aplicación	Funciones típicas	Longitud típica del mensaje	Tiempo de respuesta (seg)	Ancho de banda requerido
Automatización de oficina	Analizar y recuperar datos alfanuméricos	1.2 a 4.3Kbytes	1 a 3	6 a 70Kbits por segundo
Oficinas virtuales (para seguridad, estado real, etc.)	Analizar a escala de grises e imágenes a color para base de datos.	Escala de grises: 30 a 60Kbytes Color:250 a 500 Kbytes	1 a 5	1 a 8Mbits por segundo
Transmisión de imágenes (para publicidad)	Transmisión de fotografías	Mas de 1 Mbytes	10	800 Kbits a 5 Mbits por segundo
Médica o CAD/CAM	Transmisión de rayos X y otras imágenes a altas resoluciones	10Mbytes	2	40Mbits por segundo
Financiera	Transmisión a escala de grises	35 a 75 Kbytes por chequeo	0.025 por imagen	10 a 24 Mbits por segundo
Científica	Visualización	80Kbytes a 3 Mbtyes	0.03 a 1	600 Kbites por segundo a 800 Mbits por segundo.

VI.1.3 Elementos de ATM

ATM funciona con base en la conmutación y multicanalización de celdas; un método similar a la conmutación de paquetes en X.25 o conmutación de tramas en Frame Relay. La celda es la unidad principal en ATM y ha sido definida con un tamaño fijo de 53 bytes (424 bits). Al igual que en otras tecnologías basadas en conmutación de paquetes existen celdas de propósito especial que dan lugar a la aparición de distintos tipos de celdas que se mencionan a continuación:

1.- Celdas no utilizadas

a) Ajuste de velocidad de transferencia del medio.

b) Sincronización del medio físico.

c) No pasan a la capa ATM.

2.-Celdas no asignadas

a) Contiene VPI/VCI

b) No soporta datos.

3.-Celdas VP/VC

a) Datos del usuario.

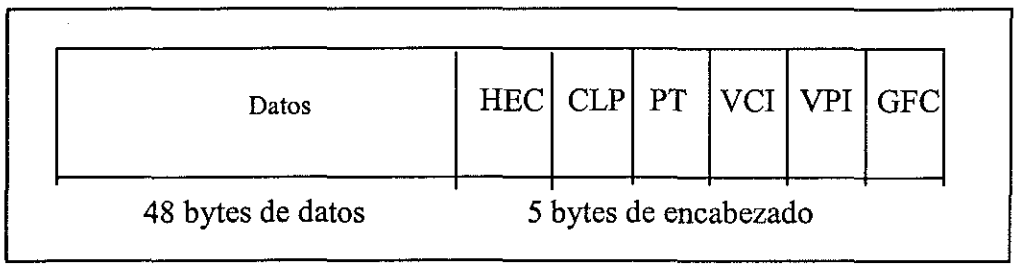
b) Señalización de (broadband) ancho de banda.

c) VC OAM.

d) SMDS.

e) ILM.

Existen dos codificaciones estándar para la estructura de la celda: la UNI (User to Network Interface), que se muestra en la figura 10.



- GFC Control Genérico de Flujo 4bits (Generic Flow Control)
- VPI Identificador de Ruta Virtual 1 byte (Virtual Path Identifier)
- VCI Identificador de Canal Virtual 2 bytes (Virtual Channel Identifier)
- PT Tipo de Información 3 bits (Paiload Type)
- CLP Prioridad de la Celda 1 bit (Cell Loss Priority)
- HEC Verificación de Errores de Encabezado 1 byte (Header Error Check)

Figura 10 Estructura de una celda ATM

Un camino de transmisión ATM, puede agrupar diversos caminos (rutas) virtuales y cada camino virtual puede transportar diversos canales virtuales, tal y como se muestra en la figura 11:

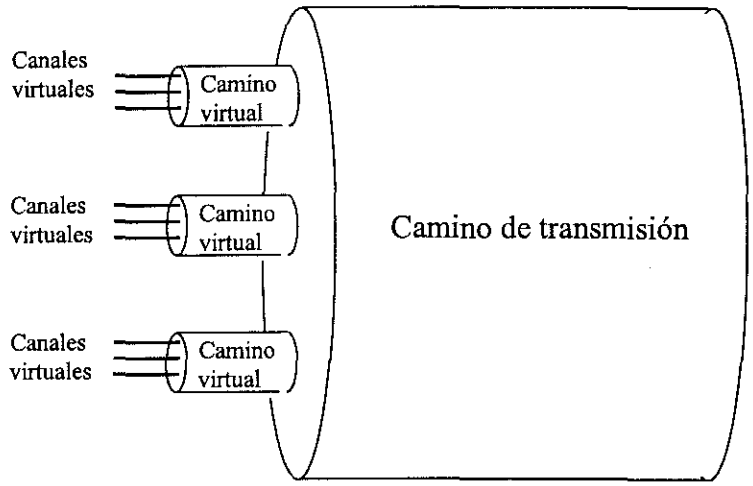


Figura 11. Agrupación de Canales y caminos virtuales en ATM

Cualquier tipo de información que vaya a ser transportada en una red ATM se divide en "pedazos' de 48 bytes, y a cada uno de estos pedazos se le agrega un encabezado de 5 bytes que incluye los campos mencionados, de modo que los nodos de conmutación de la red ATM sólo se encargan del manejo de estas celdas con base en la información que lleva en el encabezado.

A diferencia de los paquetes de X.25, en las celdas ATM sólo se verifican errores en el encabezado (mediante el campo HEC), dejando la detección de errores y corrección de errores en la información a las capas más altas en los equipos del usuario.

Una vez que la celda llega a su destino, se le retira el encabezado anexo y se vuelven a reunir reconstruyendo de esta manera la información original.

La dirección de la celda está contenida en los campos VCI y VPI; estos indican la dirección hacia donde se dirige la celda. Funciona igual que los DLCIs (identificadores de Conexión de enlace en Frame Relay); es decir, cuando un conmutador ATM recibe una celda, el VPI y el VCI dicen la procedencia de la celda, después se los cambia a la celda con base en una tabla de enrutamiento almacenada en su base de datos, y la envía por el siguiente enlace al próximo nodo. Por ésto los VCI y VPI solo tienen significado local en ATM, dado que direccionan la celda al nodo próximo, pero la ruta completa se establece con base en la configuración de las tablas de enrutamiento de los conmutadores.

En ATM están definidos 2 tipos de interfaz, dependiendo si ésta conecta con un nodo de red NNI o con un nodo de Usuario a Red (UNI). La interfaz de UNI soporta hasta 256 rutas virtuales (VPIs), la interfaz de Red a Red (NNI) soporta hasta 4096 rutas virtuales y cada ruta virtual UNI o NNI, puede contener hasta 65,536 canales virtuales (VCIs).

El primer campo de la celda GFC le permite al conmutador ATM controlar la velocidad de un equipo de usuario que va a comunicarse a través de la red de acuerdo a las condiciones de está; por ejemplo si está congestionada.

El campo PT le indica al conmutador la clase de información que forman los datos de la celda. Existen celdas con datos de señalización y de mantenimiento.

El CLP es el bit que le indica al conmutador si la celda es prioritaria o no, si tiene prioridad se descartará como ultima instancia en caso de congestión; las celdas sin prioridad son las primeras que se descartan durante los episodios de congestión.

Cada aplicación tiene diferentes requerimientos de comunicación. Por ejemplo, un enlace de voz o un enlace de videoconferencia requieren que la información llegue a su destino a una velocidad fija para operar correctamente, de ahí que se consideren aplicaciones de velocidad fija CBR (constant bit rate applications). En este tipo de enlace no importa si se pierde uno que otro bit en el enlace, ya que sólo se percibirá como una interferencia momentánea en la imagen o en la voz. Por otro lado, una aplicación de datos (el correo electrónico), requiere que la información llegue completa; no es aceptable la pérdida de un sólo bit en archivos de aplicaciones críticas de diseño o investigación científica, pero en este tipo de aplicación no importa si el archivo tarda 2 segundos en llegar y el siguiente tarda 10. Estas se conocen entonces como aplicaciones de velocidad variable VBR (variable bit rate aplications).

La ITU-T definió 4 clases de servicio nominadas A, B, C y D con las siguientes características:

Clase A: Servicio de velocidad constante (CBR), orientado a conexión con señal de reloj de extremo a extremo.

Clase B: Servicio de velocidad variable, orientado a conexión sin señal de reloj requerida.

Clase C: Servicio de velocidad variable, orientado a conexión sin señal de reloj requerida.

Clase D: Servicio de velocidad variable orientado a no-conexión y sin señal de reloj requerida.

ATM tiene capas definidas para cada función. Estas capas se comparan con las capas del modelo OSI como se muestra en la figura 12:

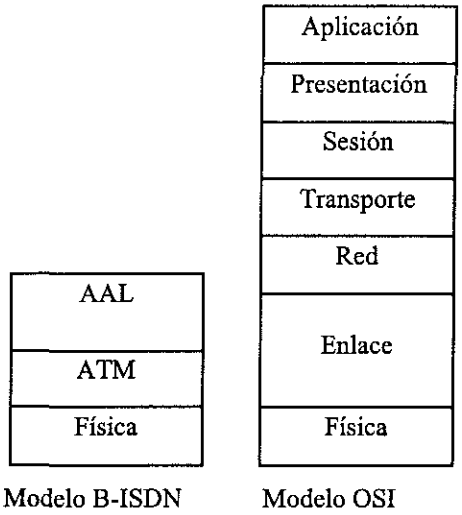


Figura 12 Comparación de ATM con el modelo de referencia OSI [GS Communications, 1996].

EL ITU-T, ANSI y Foro de ATM seleccionaron el modo de transferencia asíncrono como parte de las especificaciones de B-ISDN que provee la convergencia, la multicanalización y conmutación de celdas. En otras palabras, el ATM está basado en el modelo B-ISDN.

Como se muestra en la figura anterior, ATM funciona principalmente en las 2 primeras capas del modelo OSI, aún cuando algunas funciones de enrutamiento corresponden por sus funciones a la capa 3 del modelo de referencia. El hecho que la conmutación de celdas se realice en estos niveles es lo que permite que en ATM se puedan manejar tan altas velocidades de conmutación.

- Nivel físico: la función en general del nivel físico de ATM es vertir celdas ATM en el medio físico para enviarlas a otro nodo y recuperar las celdas recibidas desde otro nodo.

El nivel físico se divide en 2 partes PMD (physical medium dependent), tren de bits encargados de generar la señal física inyectada al medio, también se define en este nivel el tipo de interfaz (E3, DS3, E4, etc.), así como la velocidad; y TC (transmisión convergence) que se encarga de convertir las celdas ATM, provenientes del nivel superior, en una corriente estable de bits para entregarlo a PDM y de reagrupar en celdas ATM el tren de bits que recibe desde PMD, para entregarlo a la capa ATM. Otra función importante realizada por TC es el desacoplamiento de velocidad según la recomendación I.321 de ITU-T; cuando se utiliza una interfaz cuya velocidad definida no es múltiplo de 53 bytes se realiza el desacoplamiento de velocidad insertando celdas vacías y retirándolas durante la recepción.

- Nivel ATM: la función básica de este nivel es mantener las rutas y canales virtuales (VPs y VCs) esto lo realiza interpretando y asignando los identificadores de ruta y de canal (VPIs y VCIs) a las celdas que pasan por él.
- Nivel AAL (ATM Adaptation Layer): Posiblemente se trata del nivel más importante de ATM [Prycker Martin, 1995] dado que es el que permite ofrecer las diferentes clases de servicio para llevar al cabo la conexión a la red ATM de los diferentes equipos que la requieren, y comunicarse con otros equipos de su categoría. Es en esta capa donde se reciben los datos de las capas superiores en forma de PDUs (unidades de datos del protocolo). Normalmente un PDU es mucho mayor en tamaño que la parte de datos de la celda ATM, así que estas tramas se cortan en segmentos de 48 bytes para formar las celdas ATM que viajan por la red. Cuando estas celdas llegan a su destino se les retira el encabezado y se volverán a reunir para recuperar el PDU en su forma original. Existen niveles AALs específicos para cada tipo de servicio. Por ejemplo, existe un

Nivel de Adaptación ATM (AAL) para llevar al cabo el transporte de tramas TCP/IP sobre ATM, y ya hay definidos niveles AALs para otros servicios.

En una red ATM se distinguen 2 tipos de nodos: los de conmutación, que sólo reciben celdas ATM en una interfaz y las conmutan entre otras interfaces de acuerdo a sus tablas, y los nodos finales o de acceso, los cuales realizan funciones en la capa AAL al convertir la información de usuario en celdas ATM y viceversa.

VI.2 La Q3

La interfaz Q3, se encuentra como un elemento en la arquitectura de la TMN (ver figura 7), la interfaz Q3 es el elemento más desarrollada dentro de esta arquitectura. Es la única cuyas especificaciones están completas [Angeles Valencia et al. , 1995 pp.127]. Conecta un OS (sistema de operación) con un NE (elemento de red), o con un QA (Adaptador Q), o con un MD (dispositivo de mediación) o con OS en la misma TMN.

El desarrollo de la Q3 de nuestro caso, comunicará un OS con un NE, el elemento de red, es un conmutador ATM, por otro lado la Q3 podría contemplar cualquiera de las siguientes 5 funcionalidades reconocidas por TMN [CCITT, X.700](ver figura 13):

- *Gestión de configuración*, se encarga de proveer los recursos y servicios en una red. Esta identifica, ejerce control, colecciona datos y provee estos a la red, preparándola para su inicialización, para proveer las operaciones y servicios terminales. La gestión de configuración tiene que ver con servicios, o redes particulares tales como redes de telefonía pública o privada.

- *Gestión de fallas* incluye la gestión de averías, recuperación de fallas, y mantenimiento preventivo que permita la autodetección de fallas. La gestión de averías correlaciona las alarmas para servicios y recursos, inicializa pruebas, diagnostica fallas, dispara servicios de restauración, y desempeña actividades necesarias para reparar la falla diagnosticada. El mantenimiento preventivo responde a los primeros indicios de falla que degradan la confiabilidad de la red y pueden eventualmente resultar en un impacto en los servicios. Las rutinas de mantenimiento tales como pruebas para la detección o corrección de problemas se deben de realizar antes de que ocurra una avería en la red.
- *Gestión de desempeño* dirige los procesos que aseguran la más eficiente utilización de los recursos de la red. Esta evalúa y reporta el comportamiento de los recursos de la red y al mismo tiempo asegura los picos de desempeño, así como los servicios de envío de voz, datos o video.
- *Gestión de contabilidad* procesa, manipula servicios y registros de la utilización de recursos, generando reportes y facturas por todos los servicios prestados. Esta establece cambios e identifica costos para el uso de los servicios y los recursos de la red.
- *Gestión de seguridad* controla el acceso a la red y protege tanto a esta como a los sistemas de gestión de la red contra abusos intencionales o accidentales, acceso no autorizado y comunicaciones perdidas. Los mecanismos de seguridad deberían ser construidos con la flexibilidad de adecuarse a la variedad de modos de acceso.

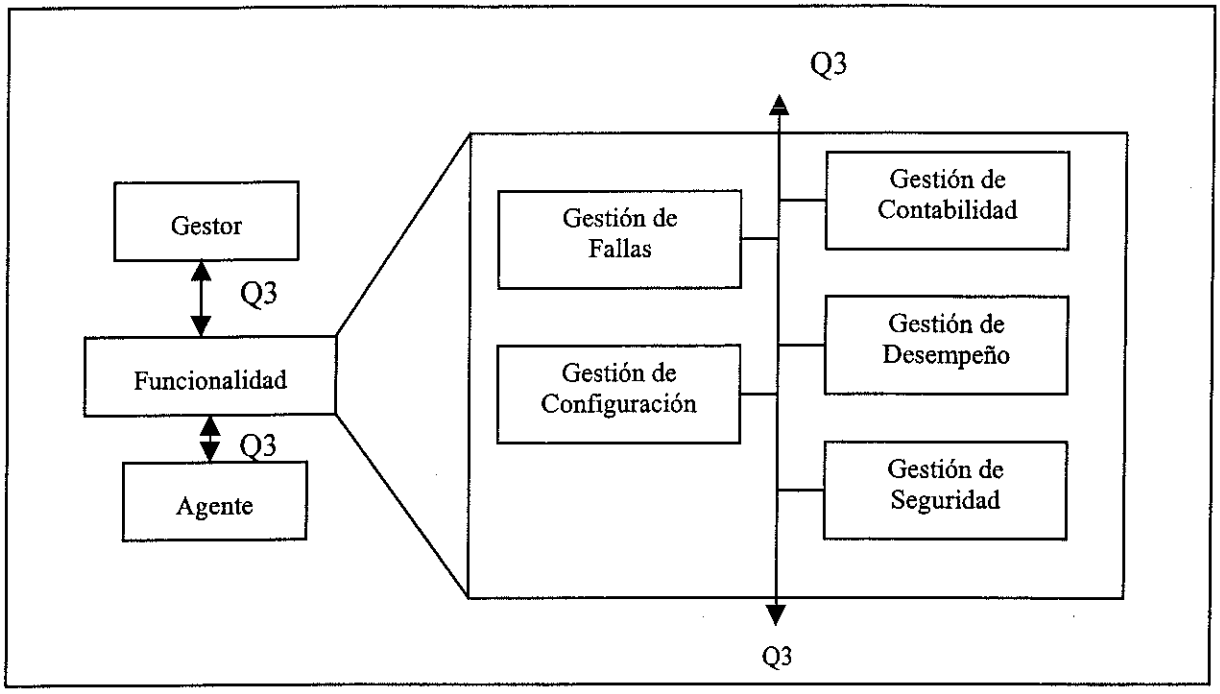


Figura 13. La Q3 y sus posibles funcionalidades.

En este trabajo se contempla la funcionalidad de Q3 para gestión de fallas, que lleve a tomar decisiones por parte de esta interfaz cuando ocurra una falla en la red.

Las fallas en una red de telecomunicaciones son bastante diversas, las más comunes son fallas en los medios de transmisión, congestión en los enlaces, alto procesamiento en los enrutadores lo que hace se puede caer el enrutador, elevación de temperatura de los equipos, pérdida de rutas tanto en equipos que utilizan el protocolo de ipx como de ip, lazos de enrutamiento, fallas en el hardware de los equipos, etc.

VI.3 Modelo de gestión

Para poder llevar al cabo la gestión de fallas y el desarrollo de la Q3 que la contemple, es necesario definir como estará integrada dicha gestión, se tienen los siguientes elementos en el modelo de gestión:

El conmutador ATM y su MIB, el agente y el gestor.

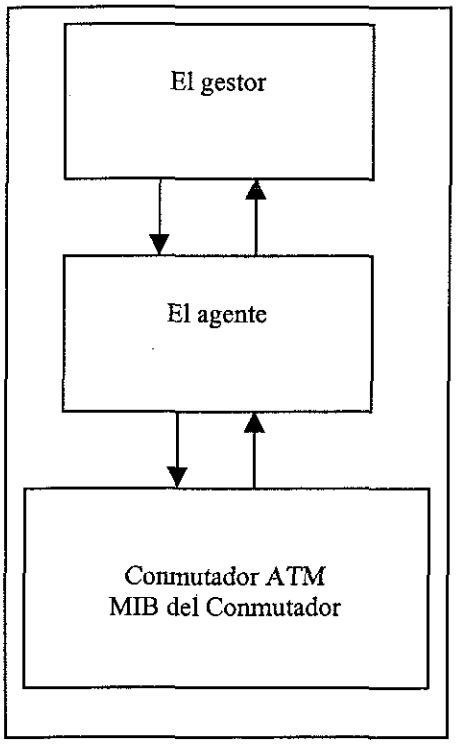


Figura 14. Modelo empleado para la implantación de la Q3

El conmutador tiene su MIB, a está accederá el agente y el gestor emitirá órdenes al conmutador mediante el agente. El agente recibe la información del conmutador en SNMP y le entrega la información al gestor en CMIS/P, para nuestro caso la Q3 estará situada en el agente, la Q3 solo sirve de puente entre el agente y el gestor, contemplando la gestión de fallas como ya se ha comentado.

Utilizando la plataforma OSIMIS se ha desarrollado el agente para el conmutador, este agente debido a que recibe la información del conmutador en SNMP y entrega la información en CMIS/P al gestor, es un agente proxy (ver figura 15), que traduce de SNMP a CMIS/P y viceversa.

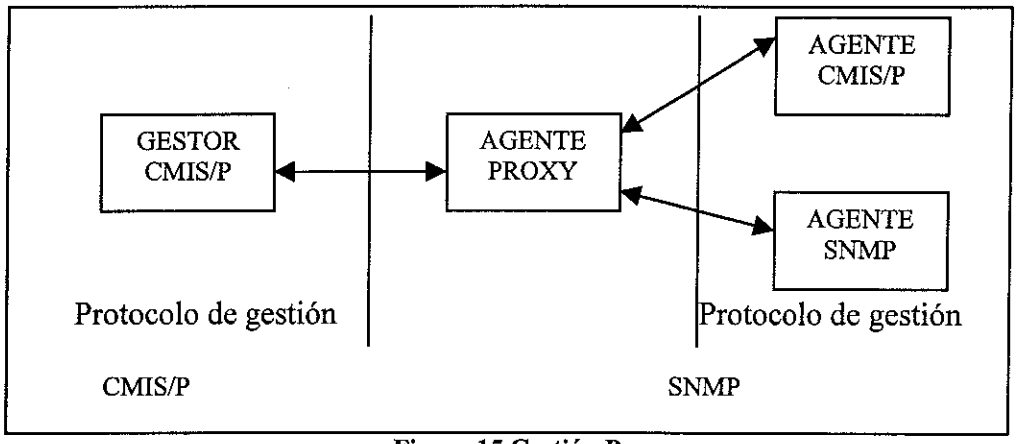


Figura 15 Gestión Proxy

Un trap es una notificación por parte del agente hacia el gestor cuando ocurren eventos inusuales en la red, se genera un trap cuando ocurre alguna falla en los dispositivos en la red, los traps que emite el conmutador de Fore estan definidos en SNMP, un trap se propaga a través de la red en las PDUs (Protocol Data Unit), la plataforma OSIMIS provee soporte para las PDU's de cualquier tipo, entre estos tipos de PDUs están los traps; para el manejo de los traps OSIMIS define la siguiente estructura en Lenguaje C:

```
struct type_SNMP_Trap__PDU {
  OID    enterprise;

  struct type_SNMP_NetworkAddress *agent__addr;

  integer   generic__trap;
#define        int_SNMP_generic__trap_coldStart  0
#define        int_SNMP_generic__trap_warmStart 1
#define        int_SNMP_generic__trap_linkDown 2
```

```
#define        int_SNMP_generic__trap_linkUp    3
#define        int_SNMP_generic__trap_authenticationFailure    4
#define        int_SNMP_generic__trap_egpNeighborLoss 5
#define        int_SNMP_generic__trap_enterpriseSpecific6

   integer    specific__trap;

   struct type_SNMP_TimeTicks *time__stamp;

   struct type_SNMP_VarBindList *variable__bindings;
};
```

La estructura anterior define los campos que componen a un trap, de estos campos *specific__trap*, es el que nos indica el número del trap específico, este número nos servirá para asociarle un nivel de severidad a cada trap, como se verá en la siguiente sección

VI.4 La gestión de fallas en la Q3

Para realizar la gestión de las fallas se tiene previamente que realizar un proceso de correlación y posteriormente, algún método o forma para la recuperación de la falla.

Una red de ATM no se escapa de las posibles fallas, con la Gestión de Redes es posible, detectar una falla, así como la toma de decisiones, para tratar de resolver la falla que se presente en la red. La detección de una falla se realiza mediante encuesta a la MIB del dispositivo, o bien mediante los traps que emiten estos, tanto las MIBs como los traps están dentro del protocolo SNMP. La encuesta de las MIB y la vigilancia de los traps quedan a cargo del administrador de la red en cuestión, esta persona es la que tiene que tomar las decisiones cuando se detecta un trap o la MIB indica fallas en el dispositivo, las fallas en una red podrían ser desde una advertencia (warning) hasta fallas críticas.

Se tienen las fallas, y se tiene la Q3, el problema a resolver será el siguiente: se parte de que se tiene una red ATM, es decir una serie de conmutadores ATM (ver figura 16), es claro que cualquier conmutador podría tener fallas, la Q3 deberá identificar el tipo de falla.

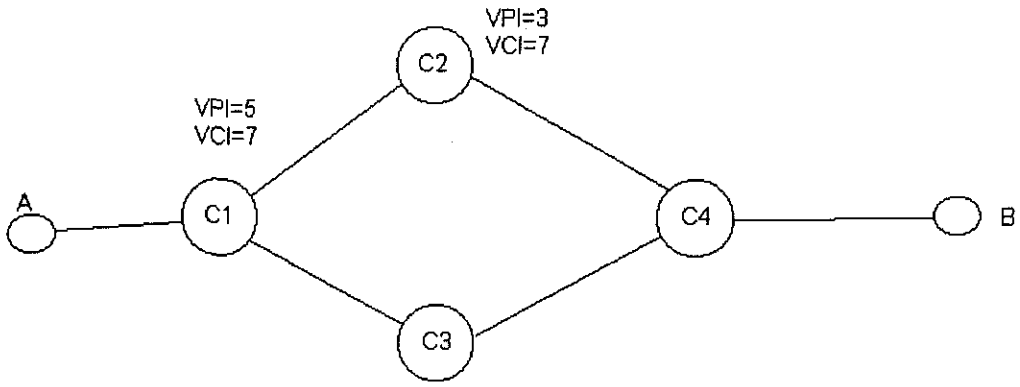


Figura 16. Arquitectura de red propuesta

En cuanto se detecten fallas la interfaz Q3 deberá realizar un proceso de correlación para determinar la gravedad de la falla, si la falla es severa, como se explicará mas adelante, enrutar el tráfico por otra trayectoria disponible, es por ello también que se parte de una determinada red, una vez que se enruta nuevamente el tráfico, se emitirá una notificación.

Es conveniente mencionar que la Q3 desarrollada en la Universidad Politécnica de Cataluña (UPC), para el proyecto Europeo MISA, no contempla la gestión de fallas. Esta interfaz Q3, esta inmersa en el conjunto de programas que constituyen al agente proxy que gestiona al conmutador, este programa se llama *qatm*. Es en esta interfaz Q3 para ATM donde se ha agregado la funcionalidad para contemplar la gestión de fallas. El agente proxy

se implantó en C++, con la ayuda de la plataforma OSIMIS, mientras que la plataforma OSIMIS se instaló en una SUN SPARC Ultra 1 con sistema operativo SUN-SOLARIS.

El conmutador empleado fue un conmutador ATM de la familia de Fore ASX200BX, con el cual se hicieron determinadas conexiones para simular una red de ATM, para evitar uso de varios conmutadores.

Los problemas en una red, específicamente en ATM, pueden ser en la conexión o en los mismos conmutadores. Nuestra interfaz Q3, que está asociada a un agente, deberá de ser capaz de detectar el tipo de falla; cada conmutador detectará una determinado falla y emitirá un trap, este trap a su vez desencadenará una serie de traps, por lo que el problema en principio tendrá que centrarse en identificar cual es el trap primario, o trap principal, es decir, aquel trap asociado a la falla real.

Se ha comentado las fallas de una forma general, pero el problema real va mas allá. Esto debido, por un lado, al tipo de traps que genera el conmutador Fore de ATM que pueden ser desde simples alertas hasta fallas críticos, y por otro lado, que cuando se repara alguna falla, se emite un trap de re-establecimiento que indica que la falla ha sido arreglada, nuevamente se tiene que emplear algún criterio para escoger entre todo el conjunto de traps que se generan al producirse dicha falla, este criterio conduce a la correlación de traps.

VI.4.1 La correlación

Cuando ocurre una falla en la red, siempre habrá un primer conmutador en la red que emitirá un trap como consecuencia de esta falla; este trap a su vez desencadenará una avalancha de traps, y en la red no tendremos un solo trap, sino un conjunto de traps. De este conjunto de traps es necesario identificar cual fue el posible trap que generó al conjunto de traps, el mecanismo que permitirá seleccionar tal trap, se conoce como correlación de traps.

Para realizar la correlación se tiene que asociarle un nivel de severidad a cada falla y en consecuencia a cada trap; por otro lado el conmutador Fore de ATM, tiene su propia MIB y emite determinados traps, cada trap tiene un número asociado en el conmutador Fore, a cada uno de estos traps emitidos por el conmutador, se tiene que asociarle un nivel de severidad para poder efectuar la correlación.

La siguiente tabla muestra algunos de los traps que se han identificado en el conmutador de Fore:

Tabla VIII. Los traps y su asociación de nivel de severidad, en un Fore ASX200BX [Serrat Fernández et al. 1998]

ATM ASX200BX(Nombre del trap)	N.trap	Severidad	ATM ASX200BX(Nombre del trap)	N.trap	Severidad
AsxSonetLOFDetected	130	2	AsxSonetLOFCleared	131	5
AsxSonetLOSDetected	36	1	AsxSonetLOSCleared	37	5
AsxSonetLineAISDetected	40	4	AsxSonetLineAISCleared	41	5
AsxSonetPathAISDetected	134	4	AsxSonetPathAISCleared	135	5
AsxVPAISDetected	1037	4	AsxVPAISCleared	1038	5
AsxSonetLineRDIDetected	132	4	AsxSonetLineRDICleared	133	5
AsxSonetPathRDIDetected	140	4	AsxSonetPathRDICleared	141	5
AsxVPRDIDetected	1039	4	AsxVPRDICleared	1040	5
AsxSonetPathLOPDetected	136	2	AsxSonetPathLOPCleared	137	5
AsxSonetPathLabelDetected	38	2	AsxSonetPathLabelCleared	39	5
AsxSonetPathUNEQDetected	138	3	AsxSonetPathUNEQCleared	139	5
AsxSonetAtmLCDDetected	142	2	AsxSonetAtmLCDCleared	143	5

Obsérvese que hay traps que indican la ocurrencia de la falla y traps que indican que la falla ha sido reparada (detected y cleared respectivamente).

La tercera columna de la tabla se refiere al nivel de severidad asociado al trap. El trap que tenga el menor valor en severidad es el trap de mayor nivel de severidad. Sobre la base de la tabla VIII es posible realizar la correlación con los traps. Recuérdese que el proceso de la correlación tiene que realizarse para detectar cual es el trap que generó al conjunto completo de traps. Los pasos para realizar la correlación tomando como referencia el documento [Deliverable D4. 1997] son los siguientes:

1. Almacenar los traps capturados por el agente en algún lugar de memoria.
2. Identificar si existen traps en parejas, es decir el aviso de la falla y la reparación del mismo.
3. Si se identifican traps en pareja, eliminar esa pareja de traps.
4. Seleccionar los traps de mayor severidad, de acuerdo a la tabla VIII. Si se encuentra un único trap de mayor severidad la correlación se ha terminado y el trap encontrado es el trap primario.
5. Si al realizar la selección de traps de mayor severidad estos se encuentran repetidos, identificar si es un punto de acceso o una conexión. Un trap será de mayor severidad si es una conexión.
6. Si después de la consideración de mayor severidad e identificación de punto de acceso o conexión, se tienen varios traps con el mismo nivel de severidad se selecciona alguno de ellos como el primario.

Puede darse el caso que todos los traps, se presenten en pareja en este caso no hay un trap primario.

Se ha descrito el proceso de correlación, otra consideración que se debe de hacer para realizar la correlación es tomar una ventana de tiempo, porque la correlación no puede estarse ejecutando en todo momento, por ello es necesario definir cada cuanto tiempo se debe de ejecutar la correlación. Al tiempo transcurrido entre un proceso de correlación y otro se le llama ventana de tiempo [Deliverable D4. 1997], en este período de tiempo los traps se almacenan si es que se generó alguno para realizar la correlación posteriormente. Hay dos factores cualitativos que influyen en el valor de la ventana de tiempo:

- El tiempo necesario para detectar las fallas y el tiempo de propagación en la red de estas. Tal tiempo típicamente está en el intervalo de milisegundos [Deliverable D4. 1997].
- El tiempo de procesamiento. Tal tiempo no es sencillo de predecir puesto que depende de la carga y de la capacidad de procesamiento de la máquina donde se ejecute la correlación, y del estado y las características de los elementos de red con sus gestores y este último con los gestores superiores. Tal tiempo no deberá de exceder el umbral de 1 segundo, que para conexiones en tiempo real, es el caso del contexto del proyecto MISA (servicios de voz e imagen digitales) [Deliverable D4. 1997].

Es claro que, una ventana mayor de tiempo es la mas adecuada para la correlación. Sin embargo, una ventana menor de tiempo significa un menor tiempo de espera para el siguiente proceso (posible solicitud de desconexión y conexión).

La ventana de tiempo se debe de escoger y configurar de tal forma que maximice la probabilidad de que en esta aparezca el trap primario con todos los traps inducidos por este, mientras minimice el tiempo de espera para el siguiente proceso. Por esta razón la ventana de tiempo debería de quedar como un parámetro de entrada, que se ajuste a distintas necesidades. Lo anterior se puede evitar si se escoge una ventana de tiempo por omisión, sin que la ventana de tiempo pierda la flexibilidad de ser modificada [Deliverable D4. 1997].

Una vez que se ha identificado el trap primario, de acuerdo a su gravedad se tomará una acción, esta acción puede ser la de mandar solo una notificación, o bien re-enrutar el tráfico por otra trayectoria (de acuerdo a la arquitectura de red propuesta) y mandar la notificación.

Una vez que se ha determinado hacer el re-enrutamiento del tráfico, se parte de que ya se tiene un algoritmo eficiente para establecer una nueva vía, además, con la ayuda de la topología de red propuesta es posible ahora solicitar la desconexión y solicitar una nueva conexión del enlace, con esto el problema inicialmente planteado queda resuelto.

La implantación de la correlación se realizó en Lenguaje C++, y se integró en programa *qatm*, realizada en la UPC, la correlación se definió como una clase, la definición de la clase en C++ se realizó en un archivo llamado corre.c, cuyo listado es:

```
#include<stdio.h>
#include "GenList.h"
#include "GenericKS.h"
#include "SNMP-types.h"
```

```
class TrapList :public List {};


class Correlacion : public KS
{
 private:

   static TrapList _lista;
   int *tabla,*trap;
   int n,i,j,k,traps, detecta;
   int inditrap,ve;

 public:

   int wakeUp(char *);


   void step1();
   void construye();
   Correlacion ();
   ~Correlacion();
};
```

El listado anterior presenta la definición de la clase *Correlacion*, la clase esta constituida por un conjunto de variables y métodos (procedimientos). Esta clase se integró al resto de los procedimientos para la construcción de la Q3 para ATM. Con la integración de esta clase, esta interfaz cuenta con gestión de fallas, que es el objetivo principal del proyecto de tesis. Un listado de la implantación de esta clase se presenta en los apéndices, así como una de las bibliotecas que utiliza la clase. El conjunto de todos los archivos se pueden encontrar en una máquina de la UPC de España, específicamente en la máquina *misa2.upc.es*, dentro del directorio */export/home/apps/osimis-4.0/osimis/aplicacions/luis/qatm.*

En esta máquina funciona *qatm* con el conmutador FORE ATM, en la máquina del CITEDI *cucapah.citedi.ipn.mx*, en el directorio */usuarios/estudia/lcaldero/qatm*, se tiene una copia del programa *qatm.*

VI.4.2. Recuperación de fallas

Para ofrecer conexiones confiables a los clientes de una red, esta debe tener implantados mecanismos adecuados y procedimientos para la recuperación de conexiones en caso de fallas en la red.

Se pueden definir dos distintos tipos de mecanismos de recuperación los de conmutación protegida y los de restauración [Silva y Paiva. 1998]. El primero se enfoca en restablecer conexiones mediante el ahorro de recursos específicamente reservados para un ámbito, mientras los de restauración implican el re-enrutamiento de la conexión afectada con recursos disponibles compartidos que se buscan en tiempo real cuando ocurre la falla. Ambos mecanismos pueden ser realizados de forma centralizada o distribuida

Al proceso de restablecer una conexión debido a una falla se le conoce como, recuperación de fallas, la recuperación de fallas tiene su propia dificultad, en el caso de estudio se está partiendo de que se tiene ya una red perfectamente bien definida, por otro lado lo que sé esta aplicando en esta topología es posible extrapolarlo a otras topologías más complejas.

El proceso de restablecer una conexión implica solicitar la desconexión actual y solicitar una nueva conexión, los procesos de conexión y desconexión son descritos por la recomendación [ITU-T Q.2931, 1994], estos pasos consisten en lo siguiente:

Cancelación de llamada

- RELEASE — Inicializa la cancelación de la llamada.
- RELEASE COMPLETE — La llamada ha sido cancelada.

Establecimiento de llamada

- SETUP — Inicializa el establecimiento de llamada.
- CALL PROCEEDING — La llamada ha sido establecida.
- CONNECT — La llamada ha sido aceptada.
- CONNECT ACKNOWLEDGE — Reconocimiento de que la llamada se ha establecido.

Las primitivas anteriores son posibles realizarlas mediante el agente y el gestor, el gestor emitirá las órdenes y el agente se encargará de realizar las órdenes del gestor.

Capítulo VII

Pruebas y resultados

VII.1 Introducción

En este capítulo se presentan algunas pruebas hechas al conmutador Fore Modelo SX200BX ubicado en campus norte de la UPC de Barcelona. Las pruebas que se presentan se realizaron con la finalidad de identificar los distintos mensajes (traps) que genera el conmutador cuando se presentan situaciones de falla. En este caso las fallas son provocadas por desconexiones intencionales en los enlaces y se han dividido en 4 partes, las cuales se detallan en la sección VII.2. Con estas pruebas se mostrará que cuando ocurre una falla, ésta genera un trap primario, lo cual provoca una serie de traps como consecuencia de la falla. La conexión que se hizo consistió en hacer varias interconexiones de un puerto a otro del conmutador para simular una red, o bién varios conmutadores, ya que solo se contaba con un conmutador ATM.

El conmutador se conectó físicamente a la estación de trabajo (SUN-SPARC ULTRA misa3.upc.es), en donde se encuentra la plataforma OSIMIS la cual proporciona un gestor CMIS/P. En esta estación de trabajo también esta ubicado el agente desarrollado en el laboratorio de gestión de red de la UPC mediante la plataforma OSIMIS. Este agente es el que realiza las operaciones de gestión sobre el conmutador, la ubicación del programa en está máquina es la siguiente:

/home/soft/osimis-4.0/osimis/aplicacions/DemoSantorini/qatm

Entre otras funciones, este programa (agente) recibe los traps del conmutador y mediante él, el gestor envía las peticiones u órdenes al conmutador.

Otra prueba realizada fue la de gestionar al conmutador de Fore de la UPC, desde una SUN-Sparc (cucapah.citedi.mx) del CITEDI en Tijuana BC. México, esta cuenta también con la plataforma ISODE-OSIMIS, la cual proporciona al gestor, se colocó al agente "*qatm*" en la SUN-Sparc del CITEDI en el siguiente directorio:

/usuarios/estudia/lcaldero/qatm

Desde *cucapah* se emitieron órdenes de gestión al conmutador, las cuales se llevaron a cabo con éxito. Esta pequeña prueba se realizó con la finalidad de verificar si era posible la gestión remota del conmutador.

En este capítulo también se presentan pruebas y resultados hechas a un programa que implanta la correlación (llamado *correlacion*), debido a que no se logró la integración completa de la clase *Correlacion* al programa *qatm.*

VII.2 Pruebas al conmutador de FORE

En esta sección se presentan las pruebas hechas al conmutador de FORE ASX200BX, estas se realizaron con la finalidad de observar el comportamiento de los traps que emite el conmutador cuando se crea una situación de falla, esta consiste en desconectar la fibra que une a los puertos del conmutador.

La forma en que se conectó el equipo se indica en la Figura 17.

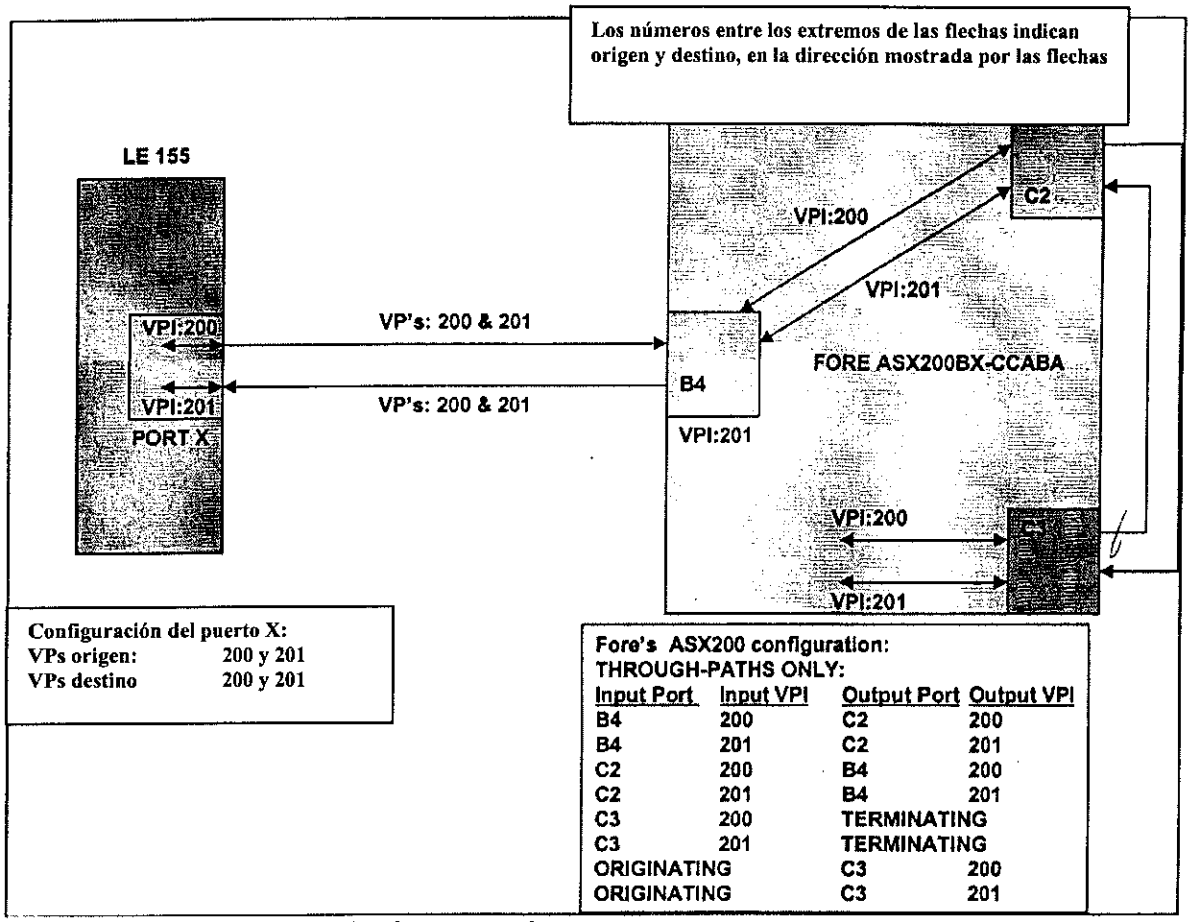


Figura 17. Configuración del conmutador FORE ASX200

La figura anterior presenta los dos conmutadores involucrados en la prueba con los VPs y tabla de enrutamiento para el conmutador de Fore. El LE155 es utilizado como una fuente de tráfico mientras el ASX200 es el conmutador bajo prueba. Ambos son enlazados

con dos fibras que llevan los VPs. En el ASX200, el puerto C2 es enlazado con C3 por medio de dos fibras también.

El objetivo de estas pruebas es observar cuales traps son generados al desconectar y conectar inmediatamente cada una de las fibras. Para cada uno de los casos de prueba se desconectó la fibra que une a dos puertos del conmutador, en las figuras esta desconexión se indica mediante una X.

Primer caso

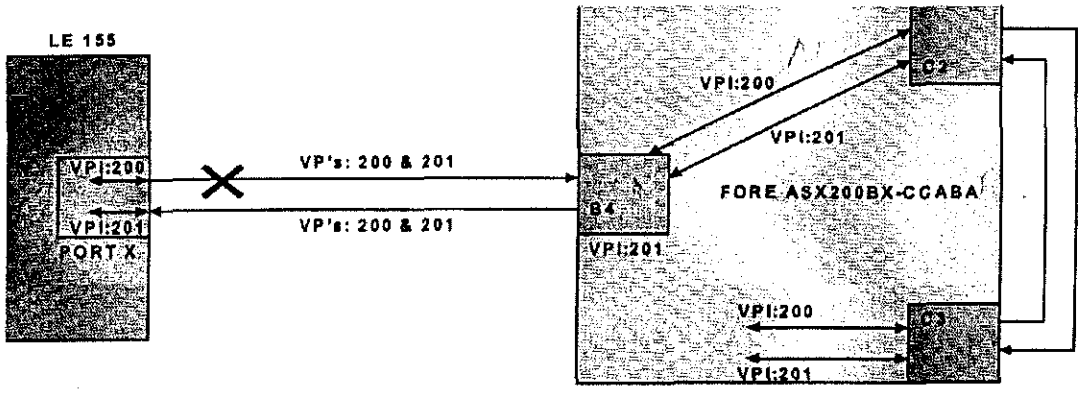


Figura 18 Primer caso de prueba

La X significa que la fibra fue desconectada, en este caso se abre la conexión entre el LE155 y el puerto B4 del conmutador. El agente recibe los traps en SNMP (agente SNMP), mientras que el gestor es proporcionado por OSIMIS y este es un gestor en CMIS/P, por lo que el agente, es un agente proxy.

Los traps generados por la desconexión de la fibra, se muestra en la figura 18.

Tabla IX. Traps generados por la desconexión de la fibra

Puerto 1B4	Puerto 1C3
	AsxVPAISDetected; VPI=200 AsxVPAISDetected; VPI=201
AsxSonetLOSDetected	
	AsxVPAISCleared; 200 AsxVPAISCleared; 201
AsxSOnetLOSCleared	

Obsérvese que en la tabla IX hay 6 traps, estos son generados como consecuencia de conectar y desconectar la fibra, el conmutador genera un primer trap asociado al puerto B4 cuando la fibra se desconectada, en este caso AsxSonetLOSDetected es el trap primario, sin embargo al agente llegan otros traps, ¿por qué?, sería la pregunta. Los traps asxVPAISDetected para VPI=200 y VPI=201 son generados como consecuencia de la falla ocurrida o reportada por el puerto B4.

El trap AsxSonetLOSDetected se genera como consecuencia de la desconexión de la fibra, este se detecta por el puerto B4 y con VPIs 200 y 201. En el puerto C3 aparecen 2 traps también con VPIs 200 y 201, pero como traps asxVPAISDetecteds para cada VPI respectivamente, esto indica necesariamente que la falla no se encuentra entre el puerto C2 y C3, la conexión B4 y C2 es ofrecida por el conmutador internamente, la falla tiene que estar entre el LE155 y el puerto B4. Cuando se conecta la fibra, se genera el trap AsxSonetLOSClreared, lo que indica que se ha restablecido el enlace, obsérvese que también en el puerto C3 aparecen nuevamente 2 traps como consecuencia del trap AsxSonetLOSCleared.

Al agente llegan los 6 traps, y lo único que realiza este es enviar todos los traps (SNMP) como notificaciones (CMIS/P) al agente superior de MISA, si se quiere realizar la

gestión de las fallas a este nivel y no dejarla al agente de MISA, la correlación tendría que ser parte del agente que gestiona al conmutador, utilizando el proceso de correlación en el agente y realizando enrutamiento del tráfico, por otro lado, el agente solo tendría que enviar una notificación de falla y restablecimiento.

Al desconectar cada una de las fibras se generaron: un trap primario y los traps secundarios como consecuencia del primario, como se muestra en las siguientes configuraciones.

Segundo caso

La configuración para este caso es la siguiente:

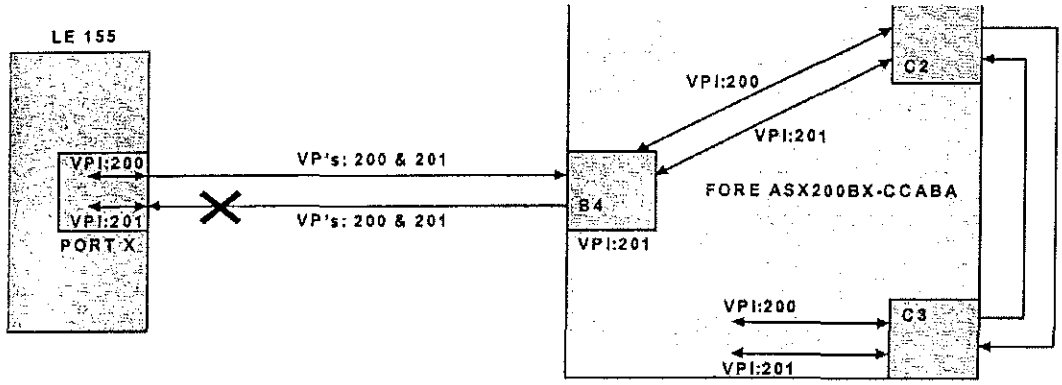


Figura 19. Segundo caso de prueba

Nuevamente la X indica que la fibra fue desconectada, para este caso se abre la conexión del puerto B4 al LE 155, la lista de traps generados para este caso los muestra la tabla X.

Tabla X. Traps generados por la desconexión de la fibra, segundo caso

Puerto 1B4	Puerto 1C3
AsxVPRDIDetected;VPI=0	AsxVPRDIDetected;VPI=200 AsxVPRDIDetected;VPI=201
AsxSonetLineRDIDetected AsxSonetPathRDIDetected AsxVPRDICleared; VPI=0	
	AsxVPRDICleared; VPI=200 AsxVPRDICleared; VPI=201
AsxSonetLineRDICleared AsxSonetPathRDICleared	

Para este caso nuevamente se sigue teniendo un conjunto de traps, y no un solo trap. En este caso no se genera ninguna falla critica (AsxSonetLOSDetected) como en el primer caso, esto es debido a que se desconectó B4 y el LE155, el trap crítico se generaría en el LE155, el puerto B4 sigue enviando, por decirlo así, la información sin problemas, es por ello que no hay un trap crítico generado. En caso de querer detectar la falla primaria para este caso, el LE155 tendría que ser gestionado también por el agente *qatm* de esta manera sería posible detectar la falla primaria.

Tercer caso

La configuración para este caso se muestra a continuación:

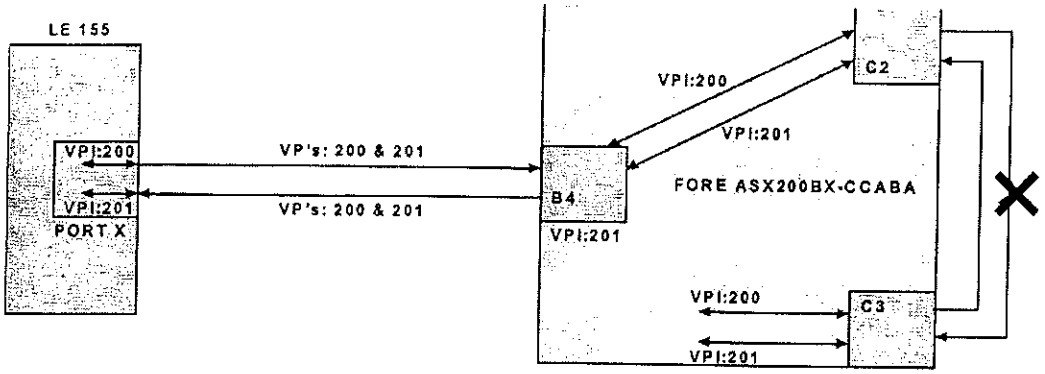


Figura 20 Tercer caso de prueba

En esta ocasión la prueba consiste en abrir la fibra que comunica al puerto C2 con el C3 como lo muestra la X de la figura 20 e identificar cuales traps se generan en este caso.

Los traps correspondientes a este caso son:

Tabla XI Traps generados por la desconexión de la fibra, tercer caso

Puerto 1C2	Puerto 1C3
AsxVPRDIDetected; VPI=0 AsxSonetLineRDIDetected AsxSonetPathRDIDetected	
	AsxSonetLOSDetected
AsxVPRDICleared:VPI=0	
	AsxSonetLOSCleared
AsxSonetLineRDICleared AsxSonetPathRDICleared	

Para esta ocasión se tiene también un conjunto de traps como consecuencia de abrir y cerrar la conexión entre los puertos C2 y C3 del conmutador de FORE ASX200BX. El problema sigue siendo encontrar al trap primario en el conjunto de traps que se generarón.

Cuarto caso

La configuración para este caso se muestra a continuación:

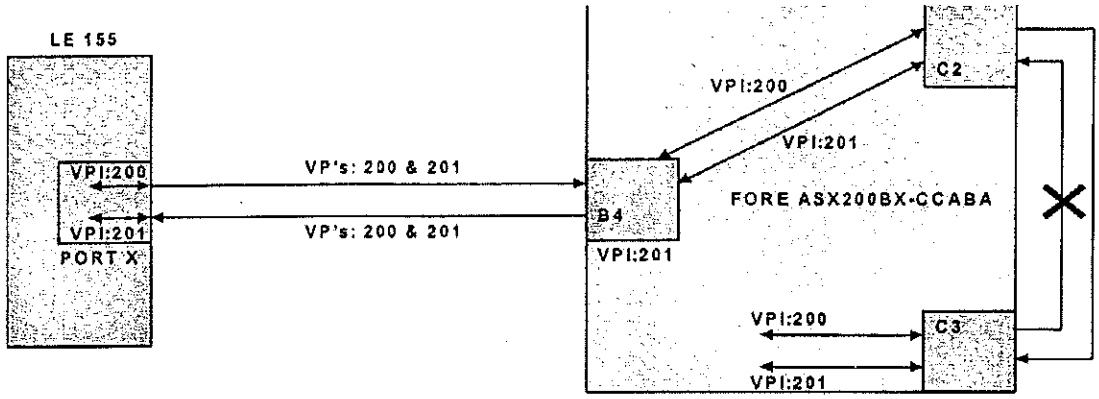


Figura 21 Cuarto caso de prueba

La prueba para esta ocasión consistió en abrir la fibra que comunica a los puertos C3 a C2 del conmutador, la X en la figura indica que la fibra fue abierta, nuevamente esta prueba tiene la finalidad de ver el comportamiento de las notificaciones de falla o traps del conmutador.

Los traps generados por esta desconexión son los siguientes:

Tabla XII Traps generados por la desconexión de la fibra, cuarto caso

Puerto 1C2	Puerto 1C3
	AsxVPRDIDetected; VPI=0 AsxVPRDIDetected; VPI=200 AsxVPRDIDetected; VPI=201
AsxSonetLOSDetected	
	AsxSonetLineRDIDetected AsxSonetPathRDIDetected
	AsxVPRDICleared; VPI=0
	AsxVPRDICleared; VPI=200 AsxVPRDICleared; VPI=201
AsxSonetLOSCleared	
	AsxSonetLineRDCleared AsxSonetPathRDCleared

En este último caso, se tiene nuevamente que se genera un conjunto de traps como consecuencia de conectar y desconectar la fibra, para poder realizar la gestión de fallas es necesario identificar el trap primario, si es que este existe.

Con estos experimentos debería de ser claro, entender que una falla no sólo genera un trap, sino un conjunto de traps y que necesitamos realizar algún proceso para realizar la gestión de fallas, el proceso propuesto es la correlación de traps.

Al integrar la clase *Correlacion* al agente *qatm*, para el primer caso de prueba la correlación indicaría que no hay falla, ya que ocurrió un trap critico y un trap de restablecimiento (clear) del trap crítico y así algo similar en los siguientes casos, al integrar esta clase, solo se pasaría la notificación de lo sucedido es decir que ocurrió una falla y que fue restablecida.

VII.3 Pruebas al programa correlación.

Para probar el proceso de correlación de traps por separado del programa *qatm*, se realizó un programa en lenguaje C llamado *correlacion* bajo el ambiente de UNIX. Este programa implanta la correlación de traps tomando como base al documento [Deliverable D4. 1997].

Las pruebas para este programa consistieron en introducir los traps que se generaron en las pruebas al conmutador de FORE (casos I a V de las pruebas al programa). El resto de los casos de prueba al programa consistieron en solicitarle al programa la generación aleatoria de traps (casos VI-IX), esto con la finalidad de observar que sucede si se aumenta el número de traps, esto simularía un aumento en la ventana de tiempo o bien el aumento en la complejidad de la red.

Como se comento en el capítulo VI la ventana de tiempo tiene que ser lo suficientemente amplio para capturar a todo el conjunto de traps cuando se genere una falla, además de que está y el próximo proceso (correlación y posible solicitud de desconexión y re-conexión) no deben ser mayor del umbral de 1 seg. Aumentar el número de traps en el programa *correlacion* permite simular incrementos en la ventana de tiempo. Aumentar el número de traps también dará una idea de hasta que número de traps es posible procesar, ya que entre mayor sea el número de traps mayor será el tiempo de procesamiento, por lo cual el número de traps aparentemente también influye en la ventana de tiempo.

La figura 22 muestra la pantalla principal del programa *correlacion*.

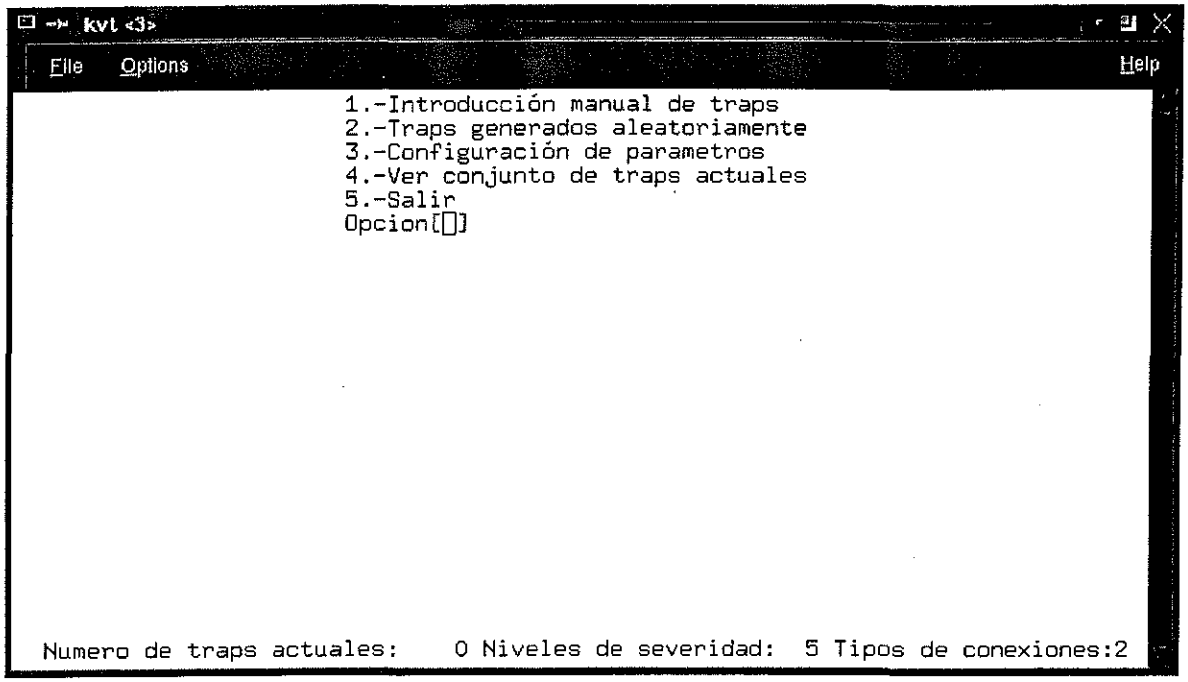


Figura 22. Pantalla principal del programa *correlacion*.

El programa correlación presenta 5 opciones, como puede observarse en la Figura 22. Este programa permite tanto la introducción manual de traps, así como generarlos aleatoriamente. En cuanto se tiene un conjunto de traps, sea de forma manual o aleatoria, el

programa realiza el proceso de correlación para entregar únicamente el trap de mayor severidad de acuerdo a los criterios mencionados en el capítulo VI o ningún trap si es que se presentan todos en pareja.

El programa *correlacion* genera los traps aleatoriamente entre 1 y 2000, a estos traps este les asocia una severidad entre 1 y 5 que se genera aleatoriamente, siendo posible la modificación de estos niveles de severidad. El programa aleatoriamente le asocia al trap un tipo de conexión, que puede ser 1 o 2, el 1 significa que es un punto de acceso, mientras que el 2 es una conexión, el programa permite aumentar los tipos de conexiones. En el capítulo VI, se mencionó que el tipo de conexión se emplea para determinar al trap de mayor severidad en caso de que se tengan traps del mismo nivel de severidad.

El programa *correlacion*, fue probado en una PC con un procesador de Intel de 166MHz y 48Mb en RAM con sistema operativo LINUX (leviatan.citedi.mx) y en una SUN SPARC Ultra I con UNIX (kumiai.citedi.mx). También se ejecutó este programa en DOS en la misma PC, en LINUX se logró medir el tiempo, siendo este de milésimas de segundo para la mayoría de los casos.

VII.4 Resultados

El resultado de las pruebas al conmutador FORE muestra la necesidad de realizar el proceso de correlación si se quiere saber cual fue el trap primario. En este tipo de experimentos por conectar y desconectar la fibra del conmutador, la correlación indica que no hubo trap primario, ¿por qué? si se generaron 6 traps.

Para el primer caso apareció el trap AsxSonetLOSDetected, que significa pérdida de la señal; éste apareció como el trap más crítico (ver tabla VIII), sin embargo también apareció el trap asxSonetLOSCleared (su pareja) lo que indica que ya no hay falla. La clase que implementa la correlación detectaría esto, por lo que no habría ya un trap primario. En cuanto a los otros traps, estos también aparecen en pareja. Por esta razón, para la correlación no hay trap primario.

Algo similar sucede en los casos 3 y 4 en donde sí aparece una falla crítico, sin embargo, también aparece acompañado de su trap de "clear" correspondiente.

En el caso 2, no aparece ningún trap asociado a una falla crítica, y además los traps de baja severidad aparecen acompañados con su pareja de "clear". Por esta razón para la correlación no existe falla.

Las fallas reales no se restablecen por si solas, por lo que el proceso de correlación es indispensable para poder realizar la gestión de fallas de manera automática.

Por ejemplo para el primer caso, al mantener la fibra desconectada, esto es, mantener la situación de falla, los traps que tendríamos se muestran en la tabla XIII.

Tabla XIII. Traps generados manteniendo la fibra desconectada

Puerto 1B4	Puerto 1C3
	AsxVPAISDetected; VPI=200 AsxVPAISDetected; VPI=201
AsxSonetLOSDetected	

En este nuevo caso, la correlación entregaría al trap AsxSonetLOSDetected como la falla crítica, sobre la base de este conocimiento el siguiente paso para la recuperación de la falla sería solicitar la desconexión y la reconexión por otra trayectoria (ver figura 16).

Con las pruebas anteriores se demuestra la importancia de la correlación para la gestión de fallas. Debe ser claro que es necesario este proceso para poder implantar la gestión de fallas en la Q3.

El proceso de correlación fue implantado por la clase *Correlacion*, esta clase fue descrita en el capítulo anterior, las pruebas a esta clase se hicieron por separado del programa *qatm* que se desarrolló en la UPC, en principio, para probar su funcionamiento individual. Una vez que se probó su correcto funcionamiento el siguiente paso fue la integración de esta clase al programa *qatm*.

El programa *qatm* como ya se ha mencionado fue desarrollado en la UPC, y no contempla la gestión de fallas; al no contemplar esto, el conjunto de clases que integran al programa *qatm* no está listo para interactuar con la clase *Correlacion*.

Como se mencionó en el capítulo anterior, la correlación se dispararía o pondría a trabajar cada cierto tiempo, el tiempo de disparo de la correlación tendría que ser del orden de milisegundos, aunque para las pruebas de esta con el programa *qatm* que se realizaron se dejo que la ventana de tiempo fuera de 10 segundos para tener el tiempo de generar las fallas. Al iniciarse el proceso de correlación, esta clase necesita del conjunto de traps que se hayan generado en ese intervalo de tiempo. Por otro lado, *qatm* solo pasa las notificaciones al sistema de operación MISA, pero lo hace individualmente, y al hacerlo el trap es desechado, por lo que es necesario modificar esta parte para que *qatm* deje almacenados los traps generados para la clase correlación. El inconveniente para realizar esta parte de *qatm* es que se debe tener un conocimiento detallado de la programación de *qatm* o bien realizar nuevamente un conjunto de procedimientos para capturar los traps cuando éstos se generan. Es más sencilla la primera proposición, aunque ambas tienen su

grado de dificultad. Lo anterior resultó en la necesidad de probar la clase que implanta la correlación por separado. Los resultados que se obtuvieron al probar el programa correlación se presentan a continuación.

CASO I

Este caso consistió en introducir los traps que se generaron en el Primer caso de pruebas al conmutador de Fore. Después de introducir los traps, el programa realiza la correlación. El resultado de esta correlación se muestra en la figura 23.

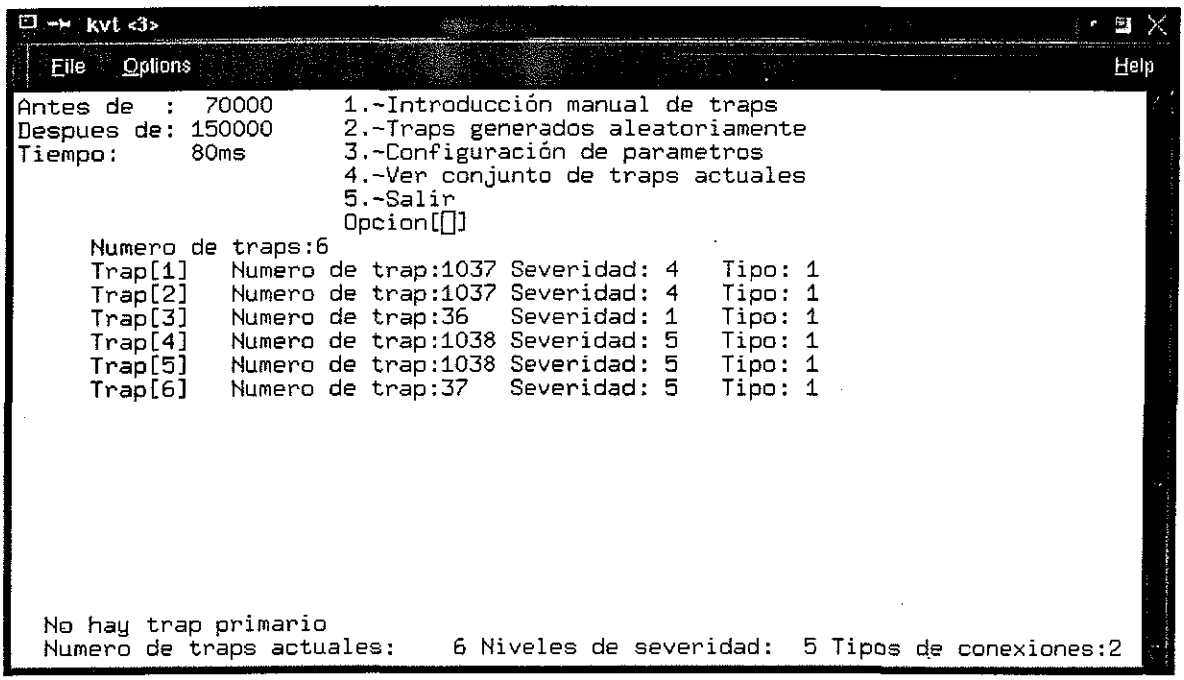


Figura 23. Resultados del programa correlación para el caso I.

El programa aplica la correlación sobre el conjunto de los 6 traps introducidos e indica que "No hay trap primario", esto ya se esperaba ya que todos los traps aparecen en

pareja (ver tabla VIII), es decir aparece el trap de falla y el trap de su re-establecimeinto. El programa es capaz de identificar cuando los traps aparecen en pareja. Nótese también que en la figura 23 todos los traps tienen tipo de conexión 1, como todos los traps son una conexión (no hay puntos de acceso) no importa si es 1 o 2 el número asociado al trap, siempre y cuando todos tengan el mismo número. El tiempo de proceso para la correlación es de 80ms.

CASO II

Este caso consistió en introducir los traps presentados en la tabla X al programa *correlacion*, los resultados se presentan en la figura 24.

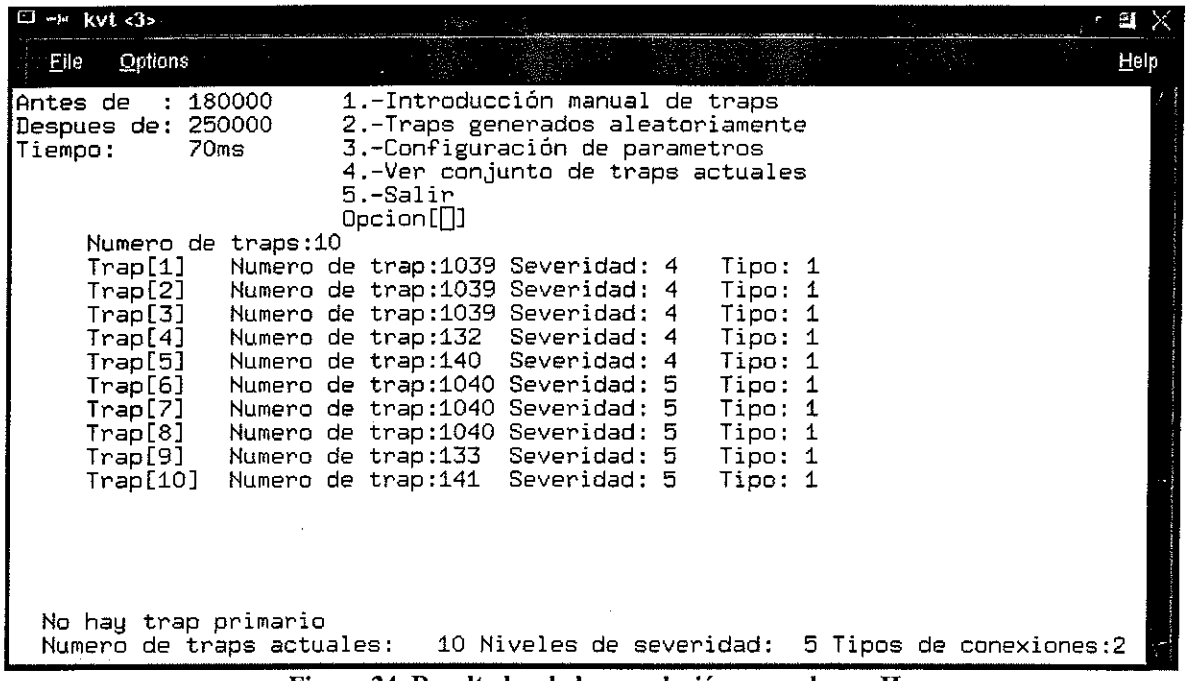


Figura 24. Resultados de la correlación para el caso II

Como se esperaba, el programa que implanta la correlación vuelve a indicar que "No hay trap Primario", esto se debe que aparecen los traps en parejas, el asociado a la falla y el asociado a su re-establecimeinto. El tiempo de procesamiento de la correlación es de 70ms.

CASO III

Este caso consistió en introducir los traps presentados en la tabla XI al programa *correlacion*, los resultados se presentan en la figura 25.

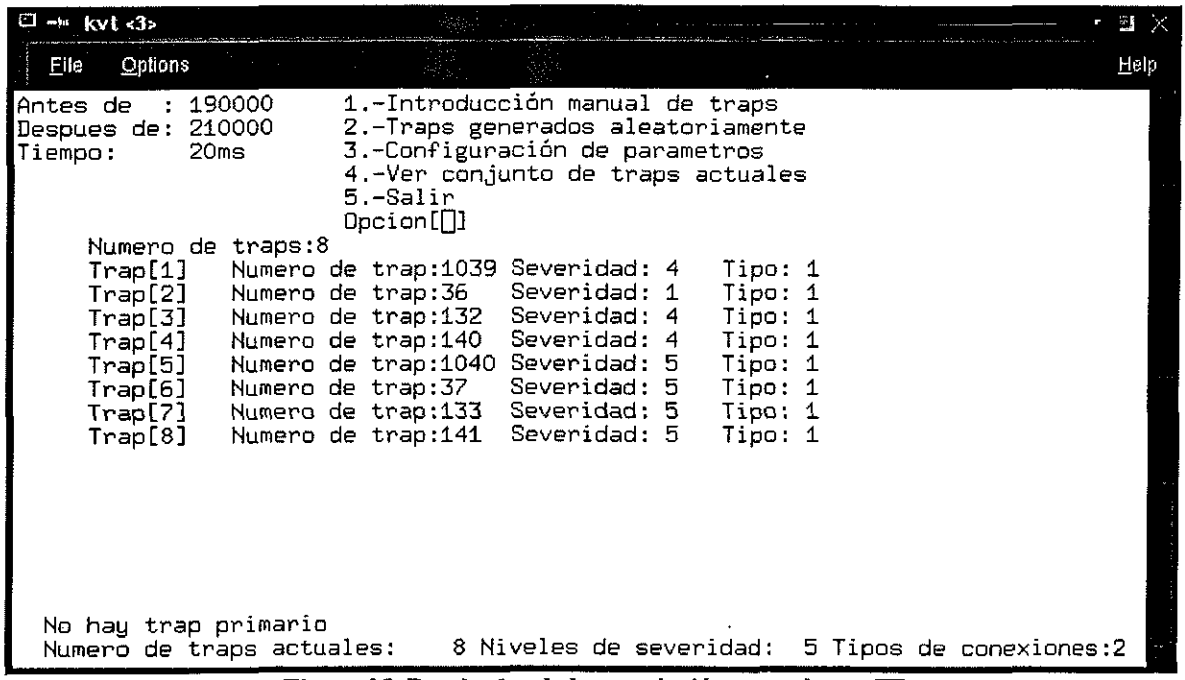


Figura 25. Resultados de la correlación para el caso III

Para este caso el programa *correlacion* sigue indicando, "No hay trap primario", esto ya se esperaba, ya que los traps que aparecen tiene su pareja respectiva, lo cual indica que no hay falla. El tiempo de procesamiento de la correlación es de 20ms.

CASO IV

El presente caso consistió en introducir los traps presentados en la tabla XII al programa *correlacion*, los resultados se presentan en la figura 26.

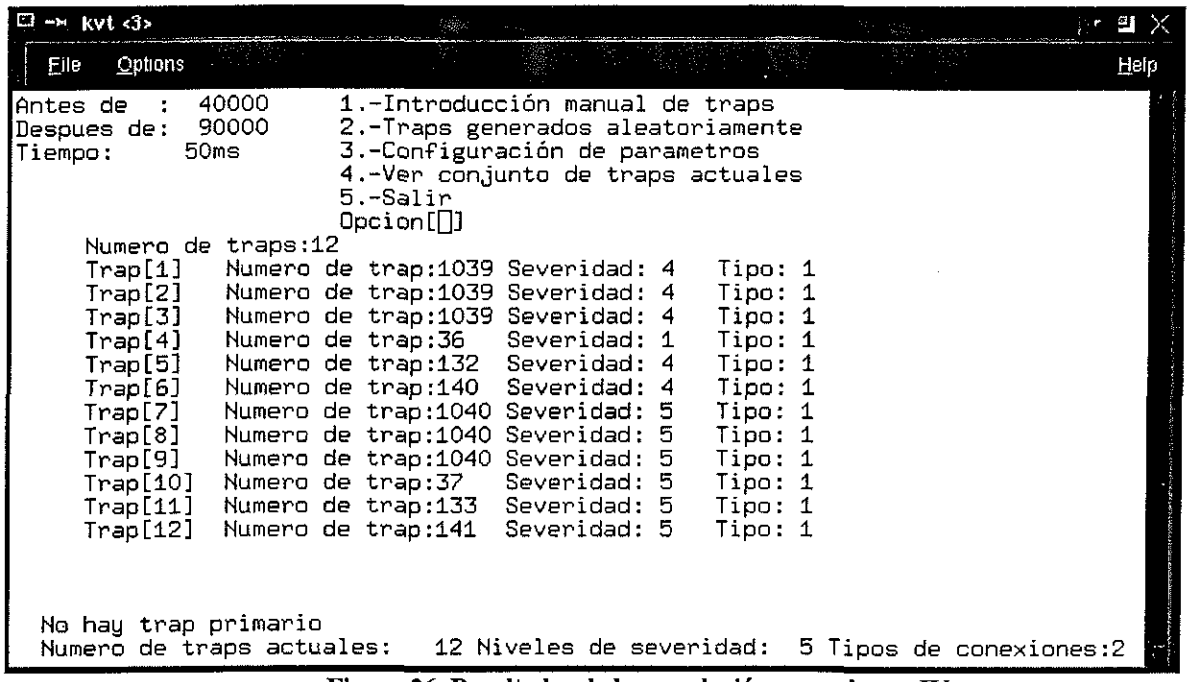


Figura 26. Resultados de la correlación para el caso IV.

En este caso la correlación da como resultado "No hay trap primario", los traps vuelven a aparecer con su pareja respectiva por lo que al terminar la correlación, no hay trap primario. El tiempo de proceso para este caso con 12 traps es de 50ms.

Caso V

Este caso consistió en introducir al programa *correlacion*, los traps presentados en la tabla XIII, los resultados fueron los que se muestran en la figura 27.

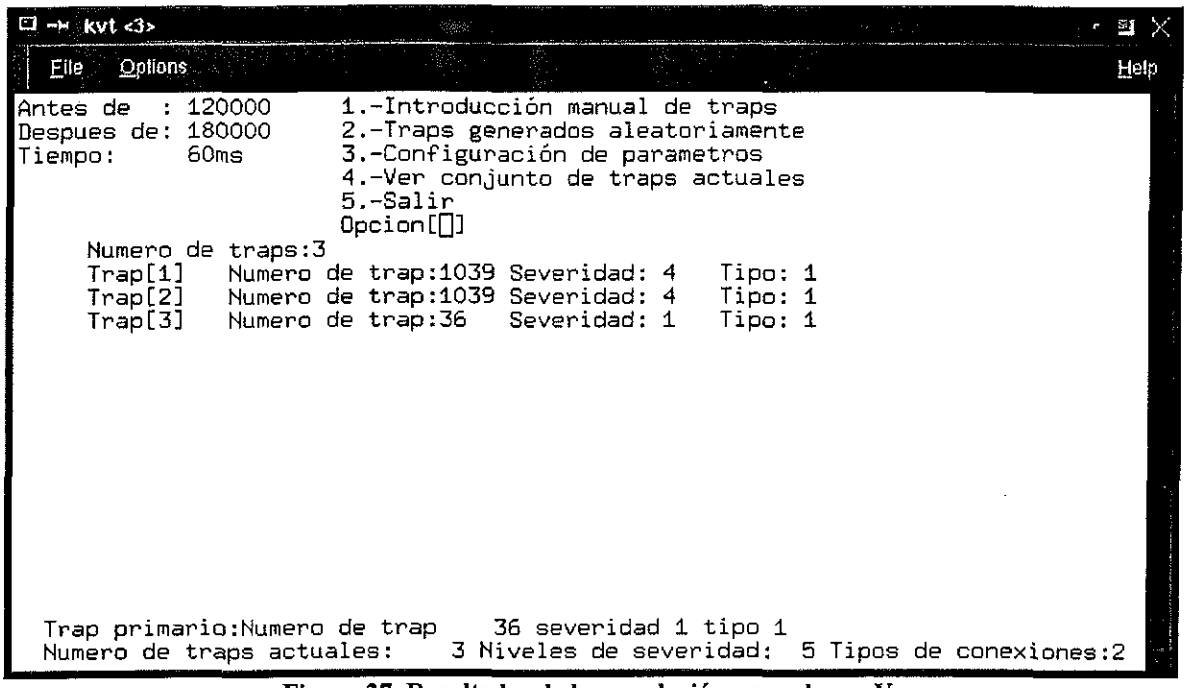


Figura 27. Resultados de la correlación para el caso V.

Para este caso la correlación nos indica que: "Trap primario: Número de trap 36 severidad 1 tipo 1". Se esperaba que si se presentara un trap primario, el programa *correlacion* nos entrega precisamente el trap que se esperaba. El tiempo de proceso es de 60ms.

Caso VI

Este caso consistió en escoger la opción 2 del programa, es decir generar los traps aleatoriamente, se le indicó al programa que generara 10 traps. Este caso, el caso 7, el caso 8 y el caso 9, tienen la finalidad de ver cual es el tiempo de respuesta del programa, esto para tratar de establecer si influye o no el número de traps en la ventana de tiempo. Se comentó en el capítulo VI, que el umbral de tiempo de procesamiento no debería exceder 1 seg. El resultado de aplicar la correlación a 10 traps se presenta en la figura 28.

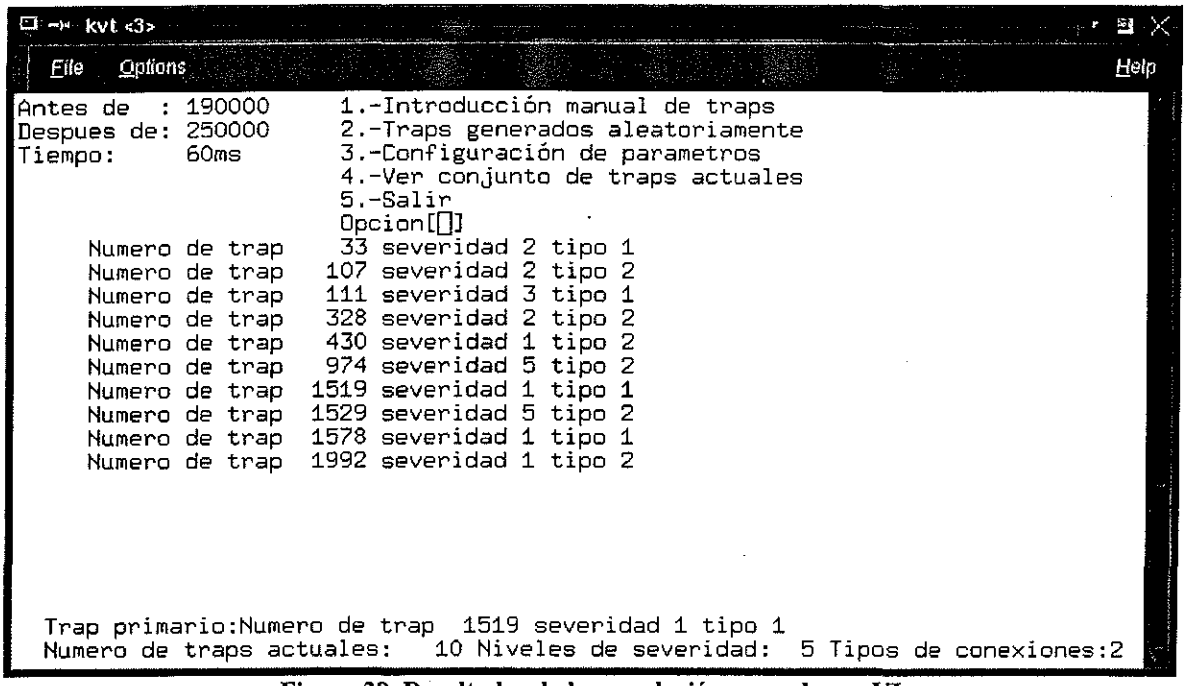


Figura 28. Resultados de la correlación para el caso VI

En este caso, la correlación indica que: "Trap primario: Número de trap 1519 severidad 1 tipo 1". Los traps aparecen con distintas severidades y con distinto tipo de conexión. En este conjunto de traps aparecen 4 traps con severidad 1, 430, 1519, 1578 y 1992, estos traps tienen el mismo nivel de severidad, el criterio que decide que trap es el más severo es el tipo de conexión (punto de acceso o conexión), solo hay dos traps con la misma severidad y el mismo tipo de conexión 1519 y 1578, recuérdese que el 1 es para el

programa correlación punto de acceso y el dos conexión. Con este último criterio el trap más severo puede ser cualquiera de los dos, esto difícilmente se presentaría en la vida real. El programa correlación respondió como se esperaba. El tiempo de proceso es de 60ms.

Caso VII

En este caso se le dió como entrada al programa 100 traps, estos se generaron aleatoriamente. El resultado se muestra en la figura 29.

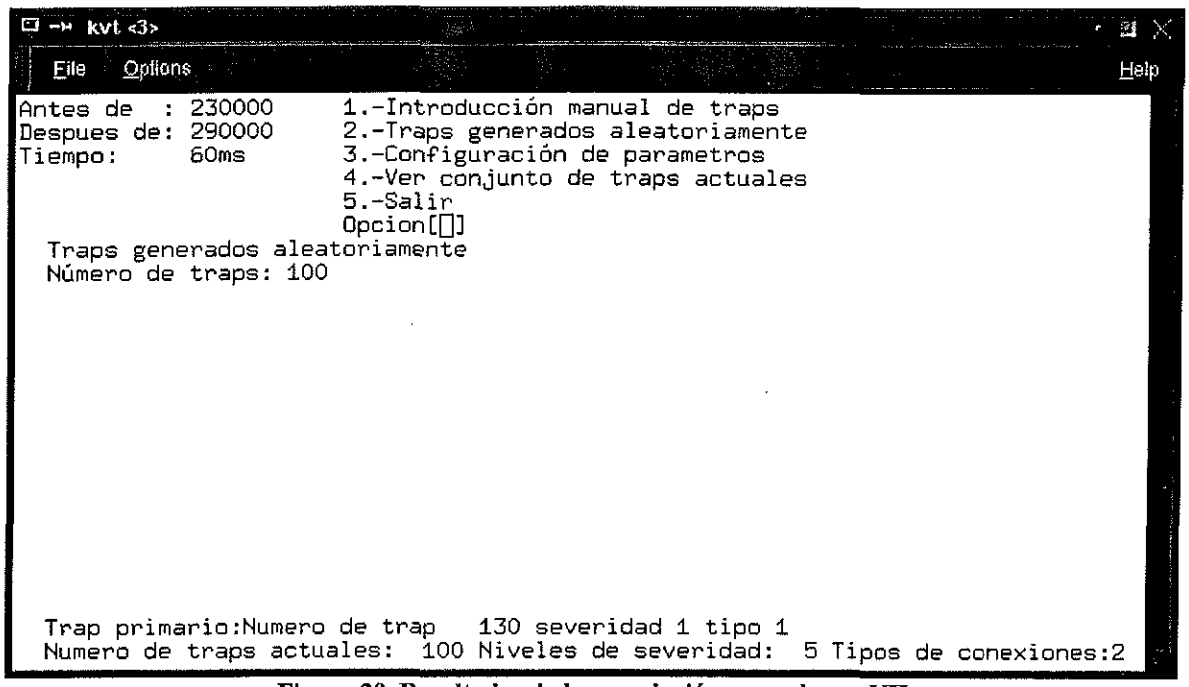


Figura 29. Resultados de la correlación para el caso VII

El programa entrega un trap, primario lo cual se podía esperar antes de esta prueba, dada su probabilidad de ocurrencia. El tiempo de procesamiento es de 60ms.

Caso VIII

En este caso se le dió como entrada al programa 1000 traps, estos se generaron aleatoriamente el resultado se muestra en la figura 30.

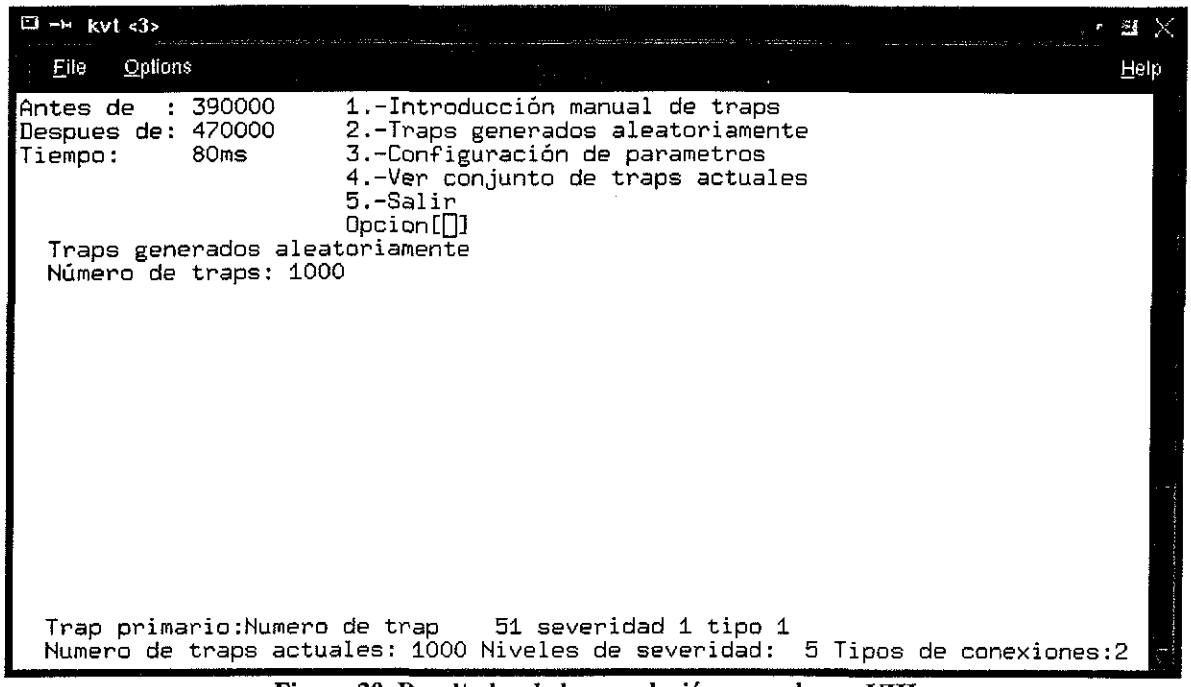


Figura 30. Resultados de la correlación para el caso VIII

Un trap primario vuelve a entregar el programa que implanta la correlación, la ejecución del programa para 1000 traps solo fue posible en los sistemas operativos LINUX y UNIX, esto debido a las limitaciones de memoria de DOS. El tiempo de procesamiento es de 80ms.

Caso IX

En este caso se utilizaron 10,000 traps para probar al programa *correlacion*, los resultados se muestran en la figura 31.

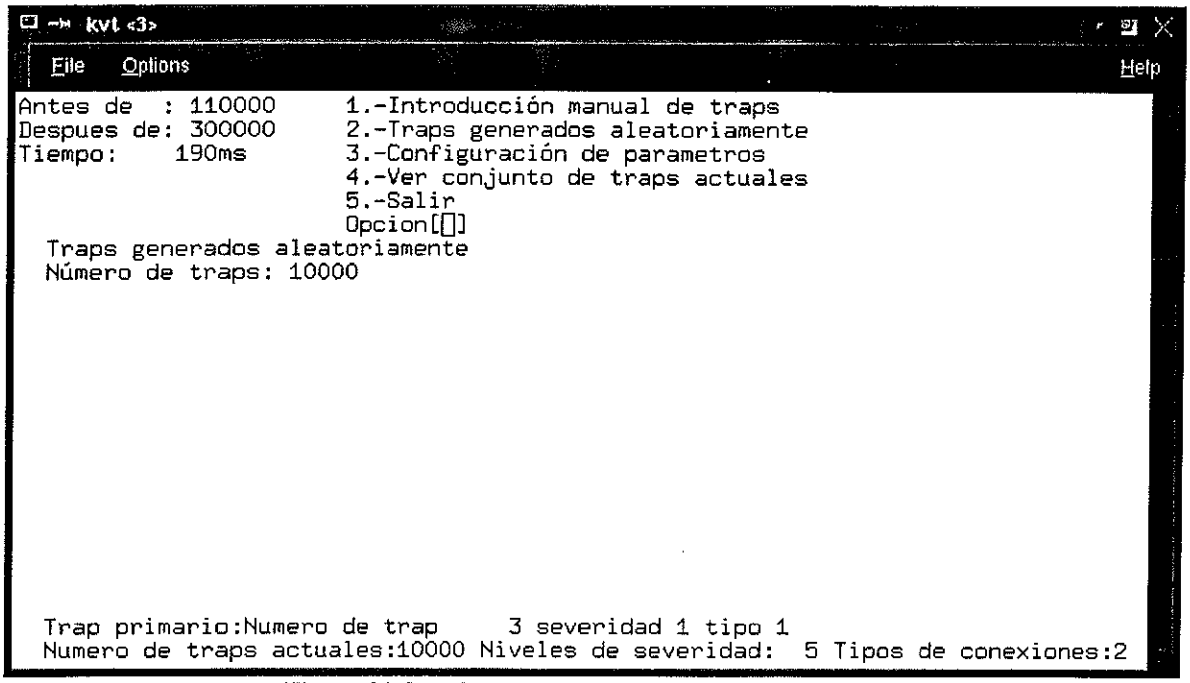


Figura 31. Resultados de la correlación para el caso IX

El programa *correlacion*, sigue trabajando sin problema y vuelve a entregar el trap primario. El tiempo de proceso es de 190ms.

El programa fue probado también para 20,000, 50,000, 100,000 y 1,000,000 de traps obteniéndose 320ms, 780ms, 1750ms y 46,320ms como tiempo de procesamiento respectivamente. Con estas pruebas y resultados puede decirse que la ventana de tiempo no depende del tiempo de procesamiento, en una topología de red sencilla, como la que se propone en este trabajo (ver figura 16)

Las pruebas y resultados al programa *correlacion*, se realizaron debido a que no se logró la integración completa de la clase *Correlación* al agente del conmutador (*qatm*), falta por integrar esta clase al programa *qatm*.

Cuando no se contempla la gestión de las fallas en *qatm*, todos los traps (SNMP) que se generan se transforman en notificaciones (CMIS/P), la Q3, al no contemplar la gestión de las fallas, lo único que hace es pasar las notificaciones al sistema de operación. Esto hace que el sistema de operación tenga que realizar mayor procesamiento. Al contemplar la gestión de fallas, la Q3 se encarga de realizar el proceso de correlación, así como solicitar la desconexión y la nueva conexión para evitar que un enlace se pierda debido a una falla. Al realizar la Q3 la gestión de fallas, bastará con notificar al sistema de operación superior que ha ocurrido una falla y se ha solucionado, siempre y cuando sea posible recuperar la falla. Si esto no es posible, aún así se liberaría de trabajo al sistema de operación superior, ya que solo se le notificaría de la falla primaria.

El hecho de que la Q3 realice la gestión de fallas (la correlación y la solicitud de conexión y desconexión), resultará en liberar al sistema de operación del proceso de la gestión de fallas.

VIII Conclusiones

Para poder llevar a cabo esta tesis, fue necesaria la instalación de las plataformas ISODE y OSIMIS, en una estación de trabajo SUN-SPARC(kumiai.citedi.mx). El proceso de instalación de estas plataformas se describe en el apéndice A. La instalación de éstas no es sencilla, por lo que se recomienda considerar el proceso de instalación detallado en dicho apéndice o bien consultar los manuales respectivos si se planea utilizar este software.

Con el uso de las plataformas ISODE-OSIMIS se logró que: el agente *qatm* desarrollado con las anteriores plataformas, fuese capaz de gestionar remotamente el conmutador FORE Modelo SX200BX.

Lo anterior fue demostrado en las pruebas que se realizaron entre una estación de trabajo las SUN-SPARC ULTRA del CITEDI (kumiai.citedi.mx) ubicada en Tijuana BC México y el conmutador de FORE en la UPC en Barcelona España. El agente para gestionar al conmutador fue instalado en la estación de trabajo mencionada y desde ésta se realizaron operaciones de gestión sobre el conmutador. Si bien es cierto que existen en el mercado productos de este tipo, estos suelen tener un precio elevado, no son compatibles entre los distintos fabricantes y no tienen flexibilidad para ser modificados o integrados dentro de otros programas. Sin embargo, la aplicación de gestión generada con el presente trabajo, por medio de OSIMIS es totalmente gratuita, y al disponerse de su código fuente puede modificarse para ser compatible con equipos de red de distintos fabricantes y, al mismo tiempo, puede integrarse en otros programas de gestión de nivel superior.

El tener el código fuente de la plataforma es una ventaja con respecto a las plataformas comerciales, ya que estas no permiten la modificación del código fuente. Por otra parte, las desventajas que se han podido constatar del uso de dicha plataforma de gestión son las habituales en todo producto de libre distribución, sobresaliendo en gran manera, la falta de soporte técnico.

VIII.1 Aportaciones

- Una de las aportaciones de este trabajo consistió en el desarrollo de la clase *Correlacion* (que realiza el proceso de correlación de traps), que será incorporada al resto de los programas y procedimientos de la Q3 de ATM del proyecto MISA. La Q3 anterior se transforma en una Q3 que contempla la gestión de las fallas. La Q3 previa a este trabajo no contemplaba la gestión de las fallas, por lo que este trabajo enriquece y agrega funcionalidad a la Q3 del proyecto MISA, siendo una aportación directa a un proyecto de investigación.
- Otra aportación, consistió en llevar al cabo la gestión remota entre una de las estaciones de trabajo (kumiai.citedi.mx) ubicado en Tijuana BC. México y el conmutador FORE ATM ubicado en la UPC en Barcelona España.
- Otra aportación, es el programa *correlacion,* este programa escrito en Lenguaje C y compilado en 3 sistemas operativos (DOS, LINUX y UNIX). Este presenta la flexibilidad de cambiar el número de niveles de severidad, cambiar el tipo de conexiones e introducir o generar sin problemas hasta 10,000 traps aleatoriamente así como la introducción manual de estos.

VIII.2 Conclusiones

Mediante el uso de OSIMIS se puede concluir que es viable:

- La Gestión "proxy" a través del protocolo de gestión SNMP: Las herramientas proporcionadas a tal efecto por OSIMIS están claramente orientadas a proveer una interfaz CMIS/P para la gestión de un elemento de red provisto de una MIB SNMP. En nuestro caso particular de interacción selectiva con el conmutador ATM, no nos interesaba en su totalidad, pero sí en cambio toda la realización del protocolo SNMP. También es viable utilizar el entorno ISODE como mecanismo implícito de comunicaciones, por debajo de OSIMIS (ver figura 9).

Con lo que se desarrollo en este trabajo es posible concluir:

- La gestión de fallas no está del todo desarrollada, se puede decir ésto tomando como base que en el recientemente terminado proyecto europeo MISA, la Q3 no contempla la gestión de las fallas
- Es posible concluir, tomando como base lo que se presentó en este trabajo, que la clase "*Correlacion*" que realiza la correlación, es necesaria como un primer paso para realizar la gestión de fallas en una Q3 para ATM.
- Tomando como base las pruebas y resultados del programa *correlacion* es posible concluir, que la ventana de tiempo es independiente del tiempo de proceso de la correlación, aún cuando el número de traps sea grande. Esto es válido al comparar el número de traps generados por el conmutador FORE (12 como máximo) en situación de falla, contra los 10,000 traps con que sé probó al programa *correlacion.*
- La ventana de tiempo se podrá fijar de acuerdo a las necesidades de la red, en el caso específico de este trabajo, bastaría con fijar la ventana en unas cuantas veces el tiempo de la ráfaga con el que se propagan los traps, dada la topología que se propuso en este trabajo (ver figura 16). Si se considera el tiempo de procesamiento de la correlación para 10,000 traps 190ms, con fijar la ventana de tiempo en 200ms basta, por un lado la correlación no rebasa el tiempo de la ventana y este tiempo es suficiente para la captura de la ráfaga de traps. Si se tiene una topología más compleja, la ventana de tiempo tendría que hacerse más grande, sin embargo esta no debería ser mayor que el umbral de 1 seg. [Deliverable D4. 1997].

El objetivo principal de este trabajo, de implantar la funcionalidad de gestión de fallas en una Q3 para ATM se ha cumplido de una manera parcial. La clase *Correlacion* no quedó completamente incorporada al resto del agente *qatm* por lo que esta funciona de manera autónoma del resto del agente *qatm,* y queda como trabajo futuro. Con la incorporación de la clase *Correlacion* y de una clase para realizar la recuperación de las

fallas (solicitud de desconexión y reconexión por otra trayectoria ver figura 16), la Q3 original se convierte en una Q3 que contempla la gestión de las fallas.

VIII.3 Trabajos y líneas futuras

- Realizar la integración completa al programa *qatm* de la clase *Correlacion* e implementar la clase que solicite las reconexiones una vez que se ha obtenido el trap primario.
- Optimización del código fuente de la Q3.
- Realizar la gestión de cualquier dispositivo de una red, no solo el conmutador de ATM.
- Realizar procesos de gestión en otras áreas además de la de fallas, por ejemplo contabilidad o seguridad.
- En cuanto a la plataforma OSIMIS, trabajar para hacer compatible esta plataforma con plataformas comerciales como HP-OpenView.
- Realizar la gestión en otras aplicaciones para superar la diversidad de las redes de las empresas así como se hizo con la Q3.
- En este trabajo se propuso una topología de red fija, es posible proponer cualquier otro tipo de topología, el problema se resumiría a obtener un algoritmo de enrutamiento para tal topología, en el caso de este trabajo se partió de la existencia previa de un algoritmo de enrutamiento, sería de mucho interés combinar la gestión de fallas presentada con un algoritmo de enrutamiento.
- Otra línea futura que puede desarrollarse es realizar la gestión independiente de la tecnología, es decir no limitarse a redes de ATM, sino realizar la gestión en cualquier red.

- A lo largo de este trabajo, se partió suponiendo una falla, ¿qué sucedería si se presentan realmente varias fallas? La correlación sólo entregara un trap, se tendría que implantar algún mecanismo de memoria e ir atendiendo cada trap por su nivel de severidad. Esto tiene que ver ya con gestión de configuración.
- La solución propuesta en este trabajo al problema planteado, propone no perder la conexión a pesar de una falla, esto se logra mediante enrutar el tráfico por otra trayectoria, sin embargo la falla continua, cuando se restablezca la falla será necesario volver a configurar las tablas de enrutamiento de la red, esto nuevamente tiene que ver con la gestión de la configuración.
- Finalmente se recomienda que aquellas personas que estén interesadas en realizar algún proyecto en el área de gestión y este proyecto implica el uso de alguna plataforma consideren aprender un poco de UNIX. Si se piensa trabajar con plataformas como HP-Openview o la empleada en este trabajo ISODE-OSIMIS.

Literatura Citada

Angeles Valencia Alfonso, Dos Santos Dantas Manoel, Serrat Fernandez Joan, Valle Alarcón Rafael. 1995 *Gestión de Sevicios Integrados de banda ancha:Perspectiva del usuarios.* Enrique Sánchez. Sistemas Avanzados de Telecomunicación. Vigo España pp 124-130.

CCITT M.3010 *Maintenance: Telecomunications Management Network. Principles for a Telecommunications Managemet Network.* 1993

CCITT M.3020 *Maintenance: Telecomunications Management Network. Metodología de interfaz de la Red de Gestión de las telecomunicaciones.* 1993

CCITT M.3100 *Generic Network Information Model.* 1993

CCITT X.700 *OSI Basic Reference Model Part 4:Management Framework.*

CCITT X.701 *Ssytem Management Overview.*

CCITT X.710 *Data communication network: Open System Interconnection (OSI); Management. Common Management Information Service Definition for CCITT Applications.* 1991.

CCITT X.7.11 *Data communication network: Open System Interconnection (OSI); Management. Common Management Information Service Protocol Specification for CCITT Aplications.* 1991.

CCITT X.720 *Management information Model.* 1991

CCITT X.721 *Definitions of Management.* 1991

CCITT X.722 *Guidelines for the Definitions of Managed Objects.* 1991

CCITT X.730. *Object Management Function.* 1991

CCITT X.739 *Workload Monitoring Function.* 1991

Deliverable D4. 1997. *Detailed MISA System Design* pp. 88-166.

GS Communications. 1996 *Tópicos en Telecomunicaciones "Redes de Datos"* Guadalajara Jalisco.

ISO 9595 *Common Management Information Service Definition.*

ISO 10733 *Element of Management Information Related to OSI Network Layer Standars*

ISO 10737 *Transport Layer Management*

ITU-T Q.2931 *Section 5 of Q.2931* Edinburg, 21 June, 1994

Pavlou George, McCarty Kevin, Bhatti Saleem, Knight Graham. *The OSIMIS: Making OSI Management Simple.* Departament of Computer Science, University College London.

Prycker Martin 1995. *Asynchronous Transfer Mode, Solution for Broadband ISDN.* Prentice Hall Third Edition London.

RFC 1150 *Structure and identifications of Management Information for TCP/IP bases Internets.* 1990.

RFC 1157 A Simple Network Management Protocol (SNMP). 1990.

RFC 1212 Concise MIB Definitions. 1991.

Silva Nino, Paiva Carlos 1998. *MISA Recovery Architecture.* Code: AC080-WP3-CET-00698-TC-CC3.0 pp 29.

Serrat Joan, Jaen Enric, Renter Oriol. 1998. *Qatm Agent: Detailde Design Document.* Code: AC080/MISA/WP5/UPC/QatmAgent/v3.0.

Sheldom Tom 1997. *LAN times, Guia de interoperabilidad, soluciones para interconectividad en red.* McGraw-Hill.

Stallings William 1996. *SNMP, SNMPv2, and RMON Practical Network Management.* Addison-Wesley Publishing Company. pp 446

Stallings William. 1993.*SNMP, SNMPv2 and CMIP*: The Practical Guide to Network Management Standards, ISBN 0-201-63331-0, Addison-Wesley, Publishing Company.

Apéndice A

Instalación de ISODE y OSIMIS

Para realizar la Q3 para ATM fue necesario instalar las plataformas ISODE y posteriormente OSIMIS, la instalación se realizó sobre solaris de SUN y se describe a continuación.

Primero se debe de instalar el siguiente software previamente:

Sistema Operativo: Sun SOLARIS 2.5.1. No se ha instalado ningún parche adicional.
GNU de FSF: binutils 2.8.1, bison 1.25, gcc 2.7.2.2, libg++ 2.7.2, gawk 3.0.3. Es posible conseguir las utilerias anteriores en Internet.

Instalación de ISODE

Es muy recomendable realizar la instalación como superusuario y con el shell de comandos bash.

La versión instalada corresponde a ISODE-8.0 + 3 parches:

isode-8.tar.gz
isode-8-patch1.tar.gz
isode-8-patch2-solaris.tar.gz
isode-8-patch3-linux.tar.gz.

Previamente, se tienen que descomprimir los archivos.gz utilizando, por ejemplo, gzip -d, en un directorio (p.ej. /export/home/apps/isode-8.0).

```
tar xvf isode-8.tar
tar xvf isode-8-patch1.tar
cd isode-8.0
patch -b -p0 < ../isode-8-patch1/isode-8-patch1
cd ..
tar xvf isode-8-patch2-solaris.tar
tar xvf isode-8-patch3-linux.tar
mv ./isode-8.0/tsap/ts2tli.c ./isode-8.0/tsap/ts2tli.c.BAK
cp ./ts2tli.c ./isode-8.0/tsap/
cp ./misa2.h ./isode-8.0/config/                    // Substituir aquí misa2 por el
hostname de la máquina donde se realiza la instalación.
cp ./misa2.make ./isode-8.0/config/

chown -R root:kumiai isode-8.0                      // Uniformizar propietarios y
permisos (según criterio del Administrador del Sistema).
chmod -R ug+w isode-8.0
```

```
cd isode-8.0
cd config
ln misa2.make CONFIG.make
ln misa2.h ../h/config.h
cp *.local ../support/
cd ..
./make once-only
./make
./make inst-all
./make all-quipu
./make inst-quipu
./make test
```

./make significa invocar un archivo script local, no confundir este archivo con el comando make o gmake de unix, ya que esto no permitirá la correcta instalación de la plataforma.

Instalación de OSIMIS

La versión instalada corresponde a OSIMIS-4.0 + 7 parches oficiales + 1 parch no oficial:

osimis-4.0.tar.gz
osimis-4.0-patch-01.tar.gz
osimis-4.0-patch-02.tar.gz
osimis-4.0-patch-03.tar.gz
osimis-4.0-patch-04.tar.gz
osimis-4.0-patch-05.tar.gz
osimis-4.0-patch-06.tar.gz
osimis-4.0-patch-07.tar.gz
Patch-Pavlou.

Previamente, se tienen que descomprimir los ficheros .gz utilizando, por ejemplo, gzip -d, en un directorio (p.ej. /export/home/apps/osimis-4.0).

```
tar xvf osimis-4.0.tar
tar xvf osimis-4.0-patch-01.tar
tar xvf osimis-4.0-patch-02.tar
tar xvf osimis-4.0-patch-03.tar
tar xvf osimis-4.0-patch-04.tar
tar xvf osimis-4.0-patch-05.tar
tar xvf osimis-4.0-patch-06.tar
tar xvf osimis-4.0-patch-07.tar
cd osimis
/opt/FSFpatch/bin/patch -b -p0 < ../Patch-Pavlou                    // Depende de donde se
encuentre el programa patch.
```

```
cp ../RMIBAgent.cc ./manager/rmib/
cp ../RMIButil.cc ./manager/rmib/
cp ../isoentities ./etc/
cp ../osimistailor ./etc/
cp ../osfcn.h ./include/
cp ../dbm-lock.h ./util/util/
cp ../imiscd.c ./misode/mimisc/
cp ../CONFIG.make ./
cp ../config.h ./include/isode/
cd ./compilers/mocompiler/byacc
./make all
cd ../flex
./make all
cd ../../..
cd ..

chown -R root:misa osimis                                    // Uniformizar propietarios y
permisos (según criterio del Administrador del Sistema).
chmod -R ug+w osimis

setenv OSIMISETCPATH /export/home/apps/osimis-4.0/osimis/etc        // Definir
variables de entorno según donde se pretenda instalar.
setenv GDMODIR /export/home/apps/osimis-4.0/osimis/etc/gdmodir

cd osimis
cd misode
sh ./make all
cd ..
sh ./make install-h
sh ./make
cd proxy
cd iqa
sh ./make
cd ..
cd ..
sh ./make install
cd misode
sh ./make install
cd ..
cd proxy
cd iqa
sh ./make install
```

Los pasos anteriores se realizaron para instalar las plataformas ISODE y OSIMIS.

Apéndice B

Listado de la clase "*Correlación*"

En este apéndice se presentan algunos de los listados del agente *qatm*, específicamente la clase *Correlacion* (la clase que implanta la correlación de traps), y alguna de las bibliotecas (includes) que utiliza.

El siguiente listado es el de la definición de clase Correlacion (corre.h)

```
#include<stdio.h>
#include "GenList.h"
#include "GenericKS.h"
#include " SNMP-types.h "

class TrapList :public List {};


class Correlacion : public KS
{
 private:

   static TrapList _lista;
   int *tabla,*trap;
   int n,i,j,k,traps, detecta;
   int inditrap,ve;

 public:

   int wakeUp(char *);


   void step1();
   void construye();
   Correlacion ();
   ~Correlacion();
};
```

El siguiente listado es la implantación de la clase anterior (corre.cc):

```
#include <stdio.h>
#include "corre.h"
```

```
#include "QatmFM.h"
#include "ListIterator.h"

#define  WindowTime        10



TrapList Correlacion::_lista;

//el metodo step determina el trap con mayor severidad, en este mismo
//proceso se contempla la aparicion de traps en parejas es decir, el trap del
// fallo y el trap del informe del borrado de tal fallo.

void
Correlacion::step1()
{

 for(i=0;i<n;i++) tabla[2+3*i]=0;

//el siguiente for se encarga de marcar los traps que tengan su pareja, es
//decir el trap del fallo y el trap de informe de restablecimeiento.

 for(i=0;i<n;i++)
 {
  if(tabla[1+3*i]==5)
  {
   tabla[2+3*i]=1;
   for(k=0;k<n;k++)
   {
    if((tabla[0+3*k]+1)== tabla[0+3*i] )
    {
     tabla[2+3*k]=1;
     break;
    }
   }
  }
 }

//el siguiente for determina el trap con mayor severidad
 for(i=1;i<n;i++)
 {
  if( (i==1) &&  (tabla[2+3*ve]==1)) ve++;
  if( (tabla[1+3*ve]>=tabla[1+3*i]) && (tabla[2+3*i]!=1)) ve=i;
 }
 k=0;
```

```
  trap=(int *)malloc(sizeof(int)*2*n);


// el siguiente for manda a una nueva tabla los traps con el mismo nivel de
// fallo y severidad.
  for(i=0;i<n;i++)
  {
   if(tabla[2+3*i]!=1)
   if(tabla[1+3*ve]==tabla[1+3*i])
   {
    trap[1+2*k]=tabla[1+3*i];
    inditrap++;
    k++;
   }
  }

  printf("aqui estamos\n");
}




// el siguiente metodo se encarga de construir la tabla de traps.
void Correlacion::construye()
{
 int k,j;
 struct type_SNMP_Snmp1157Msg *Trap;
 ListIterator listaTraps (&_lista);

 detecta=0;
 n=_lista.getCount();
 if(n>0)
 {
  tabla=(int *)malloc(sizeof(int)*3*n);
  for(k=0;k<n;k++)
  {
   Trap=(type_SNMP_Snmp1157Msg*)listaTraps.getNext();

  // se almacena el numero del trap en la tabla.

   tabla[0+3*k]=Trap->data->un.trap->specific__trap;
   j=0;
   while( (tabla[2*k+0]!=atmOsProbableCauses[j].getTrapId())
       && (j<k) ) j++;
```

```
    // si este if se cumple es posible determinar el tipo de trap
    // siempre deberá cumplirse este trap.

      if(tabla[0+3*k]==atmOsProbableCauses[j].getTrapId())
      {
       tabla[1+3*k]=atmOsProbableCauses[j].getSeverity();
       detecta=1;
      }
      else detecta=0;
    }
  }
}

Correlacion::wakeUp(char*)
{


 construye();
 if(detecta)
 {
  step1();
  step2();
 }
 else printf("No puedo determinar al trap primario\n");

}

Correlacion::Correlacion()
{
    scheduleWakeUps (WindowTime, NULLCP, True);
    // hacemos que se active el metodo wakeUp() cuando se cumpla el tiempo.
  }


Correlacion::~Correlacion(){}

/*Correlacion::~Correlacion()
{
type_SNMP_Snmp1157Msg *cur;
ListIterator listTraps (&_lista);

        for ( cur=(type_SNMP_Snmp1157Msg*)_lista.first(); cur;
                cur=(type_SNMP_Snmp1157Msg*)listTraps.getNext() )
          free_SNMP_Snmp1157Msg( cur );
    free(tabla);
```

```
    free(trap);
}*/
```

El siguiente listado muestra una de las bibliotecas que utiliza la clase corre (SNMP-types.h), el interés de este listado es el conocer como OSIMIS define las PDUs de SNMP asociado a los traps a partir de la MIB del conmutador FORE en SNMP, es también en este listado donde aparece el número específico(la variable asociada) de trap que es indispensable para poder llevar a cabo la correlación de traps, el listado es el siguiente:

```
#ifndef_module_SNMP_defined_
#define_module_SNMP_defined_

#ifndefPEPSY_VERSION
#definePEPSY_VERSION            2
#endif

#ifndefPEPYPATH
#include <isode/psap.h>
#include <isode/pepsy.h>
#include <isode/pepsy/UNIV-types.h>
#else
#include "psap.h"
#include "pepsy.h"
#include "pepsy/UNIV-types.h"
#endif



extern modtyp _ZSNMP_mod;
#define _ZGetRequest_PDUSNMP  5
#define _ZSnmpPrivMsgSNMP     0
#define _ZGetNextRequest_PDUSNMP     6
#define _ZPDUsSNMP        4
#define _ZAuditSNMP       18
#define _ZNetworkAddressSNMP  20
#define _ZResponse_PDUSNMP    8
#define _ZNsapAddressSNMP     24
```

```
#define _ZPDUSNMP        12
#define _ZSnmp1157MsgSNMP      19
#define _ZIpAddressSNMP  21
#define _ZTrap_PDUSNMP  23
#define _ZSetRequest_PDUSNMP   9
#define _ZObjectSyntaxSNMP       16
#define _ZSnmpMgmtComSNMP    3
#define _ZDisplayStringSNMP      17
#define _ZVarBindSNMP    13
#define _ZSNMPv2_Trap_PDUSNMP       11
#define _ZGetBulkRequest_PDUSNMP     7
#define _ZAuthInformationSNMP    2
#define _ZInformRequest_PDUSNMP       10
#define _ZTimeTicksSNMP  22
#define _ZObjectNameSNMP         15
#define _ZVarBindListSNMP        14
#define _ZSnmpAuthMsgSNMP        1

#ifndef lint
#define encode_SNMP_SnmpPrivMsg(pe, top, len, buffer, parm) \
   enc_f(_ZSnmpPrivMsgSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char *) parm)

#define decode_SNMP_SnmpPrivMsg(pe, top, len, buffer, parm) \
   dec_f(_ZSnmpPrivMsgSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char **) parm)

#define print_SNMP_SnmpPrivMsg(pe, top, len, buffer, parm) \
   prnt_f(_ZSnmpPrivMsgSNMP, &_ZSNMP_mod, pe, top, len, buffer)
#define print_SNMP_SnmpPrivMsg_P    _ZSnmpPrivMsgSNMP, &_ZSNMP_mod

#define encode_SNMP_SnmpAuthMsg(pe, top, len, buffer, parm) \
   enc_f(_ZSnmpAuthMsgSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char *) parm)

#define decode_SNMP_SnmpAuthMsg(pe, top, len, buffer, parm) \
   dec_f(_ZSnmpAuthMsgSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char **) parm)

#define print_SNMP_SnmpAuthMsg(pe, top, len, buffer, parm) \
   prnt_f(_ZSnmpAuthMsgSNMP, &_ZSNMP_mod, pe, top, len, buffer)
#define print_SNMP_SnmpAuthMsg_P    _ZSnmpAuthMsgSNMP, &_ZSNMP_mod

#define encode_SNMP_AuthInformation(pe, top, len, buffer, parm) \
   enc_f(_ZAuthInformationSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char *) parm)

#define decode_SNMP_AuthInformation(pe, top, len, buffer, parm) \
   dec_f(_ZAuthInformationSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char **)
parm)
```

```
#define print_SNMP_AuthInformation(pe, top, len, buffer, parm) \
   prnt_f(_ZAuthInformationSNMP, &_ZSNMP_mod, pe, top, len, buffer)
#define print_SNMP_AuthInformation_P    _ZAuthInformationSNMP, &_ZSNMP_mod

#define encode_SNMP_SnmpMgmtCom(pe, top, len, buffer, parm) \
   enc_f(_ZSnmpMgmtComSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char *) parm)

#define decode_SNMP_SnmpMgmtCom(pe, top, len, buffer, parm) \
   dec_f(_ZSnmpMgmtComSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char **)
parm)

#define print_SNMP_SnmpMgmtCom(pe, top, len, buffer, parm) \
   prnt_f(_ZSnmpMgmtComSNMP, &_ZSNMP_mod, pe, top, len, buffer)
#define print_SNMP_SnmpMgmtCom_P    _ZSnmpMgmtComSNMP, &_ZSNMP_mod

#define encode_SNMP_PDUs(pe, top, len, buffer, parm) \
   enc_f(_ZPDUsSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char *) parm)

#define decode_SNMP_PDUs(pe, top, len, buffer, parm) \
   dec_f(_ZPDUsSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char **) parm)

#define print_SNMP_PDUs(pe, top, len, buffer, parm) \
   prnt_f(_ZPDUsSNMP, &_ZSNMP_mod, pe, top, len, buffer)
#define print_SNMP_PDUs_P    _ZPDUsSNMP, &_ZSNMP_mod

#define encode_SNMP_GetRequest__PDU(pe, top, len, buffer, parm) \
   enc_f(_ZGetRequest_PDUSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char *)
parm)

#define decode_SNMP_GetRequest__PDU(pe, top, len, buffer, parm) \
   dec_f(_ZGetRequest_PDUSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char **)
parm)

#define print_SNMP_GetRequest__PDU(pe, top, len, buffer, parm) \
   prnt_f(_ZGetRequest_PDUSNMP, &_ZSNMP_mod, pe, top, len, buffer)
#define print_SNMP_GetRequest__PDU_P    _ZGetRequest_PDUSNMP,
&_ZSNMP_mod

#define encode_SNMP_GetNextRequest__PDU(pe, top, len, buffer, parm) \
   enc_f(_ZGetNextRequest_PDUSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char *)
parm)

#define decode_SNMP_GetNextRequest__PDU(pe, top, len, buffer, parm) \
   dec_f(_ZGetNextRequest_PDUSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char **)
parm)
```

```
#define print_SNMP_GetNextRequest__PDU(pe, top, len, buffer, parm) \
   prnt_f(_ZGetNextRequest_PDUSNMP, &_ZSNMP_mod, pe, top, len, buffer)
#define print_SNMP_GetNextRequest__PDU_P    _ZGetNextRequest_PDUSNMP,
&_ZSNMP_mod

#define encode_SNMP_GetBulkRequest__PDU(pe, top, len, buffer, parm) \
   enc_f(_ZGetBulkRequest_PDUSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char *)
parm)

#define decode_SNMP_GetBulkRequest__PDU(pe, top, len, buffer, parm) \
   dec_f(_ZGetBulkRequest_PDUSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char **)
parm)

#define print_SNMP_GetBulkRequest__PDU(pe, top, len, buffer, parm) \
   prnt_f(_ZGetBulkRequest_PDUSNMP, &_ZSNMP_mod, pe, top, len, buffer)
#define print_SNMP_GetBulkRequest__PDU_P    _ZGetBulkRequest_PDUSNMP,
&_ZSNMP_mod

#define encode_SNMP_Response__PDU(pe, top, len, buffer, parm) \
   enc_f(_ZResponse_PDUSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char *) parm)

#define decode_SNMP_Response__PDU(pe, top, len, buffer, parm) \
   dec_f(_ZResponse_PDUSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char **) parm)

#define print_SNMP_Response__PDU(pe, top, len, buffer, parm) \
   prnt_f(_ZResponse_PDUSNMP, &_ZSNMP_mod, pe, top, len, buffer)
#define print_SNMP_Response__PDU_P    _ZResponse_PDUSNMP, &_ZSNMP_mod

#define encode_SNMP_SetRequest__PDU(pe, top, len, buffer, parm) \
   enc_f(_ZSetRequest_PDUSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char *) parm)

#define decode_SNMP_SetRequest__PDU(pe, top, len, buffer, parm) \
   dec_f(_ZSetRequest_PDUSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char **)
parm)

#define print_SNMP_SetRequest__PDU(pe, top, len, buffer, parm) \
   prnt_f(_ZSetRequest_PDUSNMP, &_ZSNMP_mod, pe, top, len, buffer)
#define print_SNMP_SetRequest__PDU_P    _ZSetRequest_PDUSNMP,
&_ZSNMP_mod

#define encode_SNMP_InformRequest__PDU(pe, top, len, buffer, parm) \
   enc_f(_ZInformRequest_PDUSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char *)
parm)

#define decode_SNMP_InformRequest__PDU(pe, top, len, buffer, parm) \
```

```
   dec_f(_ZInformRequest_PDUSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char **)
parm)

#define print_SNMP_InformRequest__PDU(pe, top, len, buffer, parm) \
   prnt_f(_ZInformRequest_PDUSNMP, &_ZSNMP_mod, pe, top, len, buffer)
#define print_SNMP_InformRequest__PDU_P    _ZInformRequest_PDUSNMP,
&_ZSNMP_mod

#define encode_SNMP_SNMPv2__Trap__PDU(pe, top, len, buffer, parm) \
   enc_f(_ZSNMPv2_Trap_PDUSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char *)
parm)

#define decode_SNMP_SNMPv2__Trap__PDU(pe, top, len, buffer, parm) \
   dec_f(_ZSNMPv2_Trap_PDUSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char **)
parm)

#define print_SNMP_SNMPv2__Trap__PDU(pe, top, len, buffer, parm) \
   prnt_f(_ZSNMPv2_Trap_PDUSNMP, &_ZSNMP_mod, pe, top, len, buffer)
#define print_SNMP_SNMPv2__Trap__PDU_P    _ZSNMPv2_Trap_PDUSNMP,
&_ZSNMP_mod

#define encode_SNMP_PDU(pe, top, len, buffer, parm) \
   enc_f(_ZPDUSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char *) parm)

#define decode_SNMP_PDU(pe, top, len, buffer, parm) \
   dec_f(_ZPDUSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char **) parm)

#define print_SNMP_PDU(pe, top, len, buffer, parm) \
   prnt_f(_ZPDUSNMP, &_ZSNMP_mod, pe, top, len, buffer)
#define print_SNMP_PDU_P    _ZPDUSNMP, &_ZSNMP_mod

#define encode_SNMP_VarBind(pe, top, len, buffer, parm) \
   enc_f(_ZVarBindSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char *) parm)

#define decode_SNMP_VarBind(pe, top, len, buffer, parm) \
   dec_f(_ZVarBindSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char **) parm)

#define print_SNMP_VarBind(pe, top, len, buffer, parm) \
   prnt_f(_ZVarBindSNMP, &_ZSNMP_mod, pe, top, len, buffer)
#define print_SNMP_VarBind_P    _ZVarBindSNMP, &_ZSNMP_mod

#define encode_SNMP_VarBindList(pe, top, len, buffer, parm) \
   enc_f(_ZVarBindListSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char *) parm)

#define decode_SNMP_VarBindList(pe, top, len, buffer, parm) \
   dec_f(_ZVarBindListSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char **) parm)
```

```
#define print_SNMP_VarBindList(pe, top, len, buffer, parm) \
   prnt_f(_ZVarBindListSNMP, &_ZSNMP_mod, pe, top, len, buffer)
#define print_SNMP_VarBindList_P    _ZVarBindListSNMP, &_ZSNMP_mod

#define encode_SNMP_ObjectName(pe, top, len, buffer, parm) \
   enc_f(_ZObjectNameSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char *) parm)

#define decode_SNMP_ObjectName(pe, top, len, buffer, parm) \
   dec_f(_ZObjectNameSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char **) parm)

#define print_SNMP_ObjectName(pe, top, len, buffer, parm) \
   prnt_f(_ZObjectNameSNMP, &_ZSNMP_mod, pe, top, len, buffer)
#define print_SNMP_ObjectName_P    _ZObjectNameSNMP, &_ZSNMP_mod

#define encode_SNMP_ObjectSyntax(pe, top, len, buffer, parm) \
   enc_f(_ZObjectSyntaxSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char *) parm)

#define decode_SNMP_ObjectSyntax(pe, top, len, buffer, parm) \
   dec_f(_ZObjectSyntaxSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char **) parm)

#define print_SNMP_ObjectSyntax(pe, top, len, buffer, parm) \
   prnt_f(_ZObjectSyntaxSNMP, &_ZSNMP_mod, pe, top, len, buffer)
#define print_SNMP_ObjectSyntax_P    _ZObjectSyntaxSNMP, &_ZSNMP_mod

#define encode_SNMP_DisplayString(pe, top, len, buffer, parm) \
   enc_f(_ZDisplayStringSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char *) parm)

#define decode_SNMP_DisplayString(pe, top, len, buffer, parm) \
   dec_f(_ZDisplayStringSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char **) parm)

#define print_SNMP_DisplayString(pe, top, len, buffer, parm) \
   prnt_f(_ZDisplayStringSNMP, &_ZSNMP_mod, pe, top, len, buffer)
#define print_SNMP_DisplayString_P    _ZDisplayStringSNMP, &_ZSNMP_mod

#define encode_SNMP_Audit(pe, top, len, buffer, parm) \
   enc_f(_ZAuditSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char *) parm)

#define decode_SNMP_Audit(pe, top, len, buffer, parm) \
   dec_f(_ZAuditSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char **) parm)

#define print_SNMP_Audit(pe, top, len, buffer, parm) \
   prnt_f(_ZAuditSNMP, &_ZSNMP_mod, pe, top, len, buffer)
#define print_SNMP_Audit_P    _ZAuditSNMP, &_ZSNMP_mod

#define encode_SNMP_Snmp1157Msg(pe, top, len, buffer, parm) \
```

```
    enc_f(_ZSnmp1157MsgSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char *) parm)

#define decode_SNMP_Snmp1157Msg(pe, top, len, buffer, parm) \
    dec_f(_ZSnmp1157MsgSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char **) parm)

#define print_SNMP_Snmp1157Msg(pe, top, len, buffer, parm) \
    prnt_f(_ZSnmp1157MsgSNMP, &_ZSNMP_mod, pe, top, len, buffer)
#define print_SNMP_Snmp1157Msg_P    _ZSnmp1157MsgSNMP, &_ZSNMP_mod

#define encode_SNMP_NetworkAddress(pe, top, len, buffer, parm) \
    enc_f(_ZNetworkAddressSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char *) parm)

#define decode_SNMP_NetworkAddress(pe, top, len, buffer, parm) \
    dec_f(_ZNetworkAddressSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char **)
parm)

#define print_SNMP_NetworkAddress(pe, top, len, buffer, parm) \
    prnt_f(_ZNetworkAddressSNMP, &_ZSNMP_mod, pe, top, len, buffer)
#define print_SNMP_NetworkAddress_P    _ZNetworkAddressSNMP, &_ZSNMP_mod

#define encode_SNMP_IpAddress(pe, top, len, buffer, parm) \
    enc_f(_ZIpAddressSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char *) parm)

#define decode_SNMP_IpAddress(pe, top, len, buffer, parm) \
    dec_f(_ZIpAddressSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char **) parm)

#define print_SNMP_IpAddress(pe, top, len, buffer, parm) \
    prnt_f(_ZIpAddressSNMP, &_ZSNMP_mod, pe, top, len, buffer)
#define print_SNMP_IpAddress_P    _ZIpAddressSNMP, &_ZSNMP_mod

#define encode_SNMP_TimeTicks(pe, top, len, buffer, parm) \
    enc_f(_ZTimeTicksSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char *) parm)

#define decode_SNMP_TimeTicks(pe, top, len, buffer, parm) \
    dec_f(_ZTimeTicksSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char **) parm)

#define print_SNMP_TimeTicks(pe, top, len, buffer, parm) \
    prnt_f(_ZTimeTicksSNMP, &_ZSNMP_mod, pe, top, len, buffer)
#define print_SNMP_TimeTicks_P    _ZTimeTicksSNMP, &_ZSNMP_mod

#define encode_SNMP_Trap__PDU(pe, top, len, buffer, parm) \
    enc_f(_ZTrap_PDUSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char *) parm)

#define decode_SNMP_Trap__PDU(pe, top, len, buffer, parm) \
    dec_f(_ZTrap_PDUSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char **) parm)
```

```
#define print_SNMP_Trap__PDU(pe, top, len, buffer, parm) \
   prnt_f(_ZTrap_PDUSNMP, &_ZSNMP_mod, pe, top, len, buffer)
#define print_SNMP_Trap__PDU_P    _ZTrap_PDUSNMP, &_ZSNMP_mod

#define encode_SNMP_NsapAddress(pe, top, len, buffer, parm) \
   enc_f(_ZNsapAddressSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char *) parm)

#define decode_SNMP_NsapAddress(pe, top, len, buffer, parm) \
   dec_f(_ZNsapAddressSNMP, &_ZSNMP_mod, pe, top, len, buffer, (char **) parm)

#define print_SNMP_NsapAddress(pe, top, len, buffer, parm) \
   prnt_f(_ZNsapAddressSNMP, &_ZSNMP_mod, pe, top, len, buffer)
#define print_SNMP_NsapAddress_P    _ZNsapAddressSNMP, &_ZSNMP_mod


#endif  /* lint */

#definetype_SNMP_GetRequest__PDU      type_SNMP_PDU
#definefree_SNMP_GetRequest__PDU      free_SNMP_PDU

#definetype_SNMP_GetNextRequest__PDU      type_SNMP_PDU
#definefree_SNMP_GetNextRequest__PDU free_SNMP_PDU

#definetype_SNMP_GetBulkRequest__PDU      type_SNMP_PDU
#definefree_SNMP_GetBulkRequest__PDU free_SNMP_PDU

#definetype_SNMP_Response__PDU      type_SNMP_PDU
#definefree_SNMP_Response__PDU      free_SNMP_PDU

#definetype_SNMP_SetRequest__PDU      type_SNMP_PDU
#definefree_SNMP_SetRequest__PDU      free_SNMP_PDU

#definetype_SNMP_InformRequest__PDU  type_SNMP_PDU
#definefree_SNMP_InformRequest__PDU  free_SNMP_PDU

#definetype_SNMP_SNMPv2__Trap__PDU      type_SNMP_PDU
#definefree_SNMP_SNMPv2__Trap__PDU free_SNMP_PDU

#definetype_SNMP_ObjectName    OIDentifier
#definefree_SNMP_ObjectName    oid_free

#definetype_SNMP_ObjectSyntax    PElement
#definefree_SNMP_ObjectSyntax    pe_free

#definetype_SNMP_DisplayString    qbuf
#definefree_SNMP_DisplayString    qb_free
```

```
#definetype_SNMP_NetworkAddress        type_SNMP_IpAddress
#definefree_SNMP_NetworkAddress        free_SNMP_IpAddress

#definetype_SNMP_IpAddress      qbuf
#definefree_SNMP_IpAddress      qb_free

#definetype_SNMP_NsapAddress    qbuf
#definefree_SNMP_NsapAddress    qb_free

struct type_SNMP_SnmpPrivMsg {
  OID    privDst;

  struct qbuf *privData;
};
#definefree_SNMP_SnmpPrivMsg(parm)\
        (void) fre_obj((char *) parm, _ZSNMP_mod.md_dtab[_ZSnmpPrivMsgSNMP],
&_ZSNMP_mod, 1)

struct type_SNMP_SnmpAuthMsg {
  struct type_SNMP_AuthInformation *authInfo;

  struct type_SNMP_SnmpMgmtCom *authData;
};
#definefree_SNMP_SnmpAuthMsg(parm)\
        (void) fre_obj((char *) parm, _ZSNMP_mod.md_dtab[_ZSnmpAuthMsgSNMP],
&_ZSNMP_mod, 1)

struct type_SNMP_AuthInformation {
  int       offset;
#definetype_SNMP_AuthInformation_1      1
#definetype_SNMP_AuthInformation_2      2

  union {
    struct qbuf *choice_SNMP_0;

    struct element_SNMP_0 {
      struct qbuf *authDigest;

      integer    authDstTimestamp;

      integer    authSrcTimestamp;
    } *choice_SNMP_1;
  }     un;
};
#definefree_SNMP_AuthInformation(parm)\
```

```
        (void) fre_obj((char *) parm,
_ZSNMP_mod.md_dtab[_ZAuthInformationSNMP], &_ZSNMP_mod, 1)

struct type_SNMP_SnmpMgmtCom {
   OID    dstParty;

   OID    srcParty;

   OID    context;

   struct type_SNMP_PDUs *pdu;
};
#definefree_SNMP_SnmpMgmtCom(parm)\
        (void) fre_obj((char *) parm,
_ZSNMP_mod.md_dtab[_ZSnmpMgmtComSNMP], &_ZSNMP_mod, 1)

struct type_SNMP_PDUs {
   int       offset;
#definetype_SNMP_PDUs_get__request     1
#definetype_SNMP_PDUs_get__next__request      2
#definetype_SNMP_PDUs_get__bulk__request      3
#definetype_SNMP_PDUs_response 4
#definetype_SNMP_PDUs_set__request     5
#definetype_SNMP_PDUs_inform__request6
#definetype_SNMP_PDUs_snmpV2__trap  7
#definetype_SNMP_PDUs_trap       8

   union {
      struct type_SNMP_GetRequest__PDU *get__request;

      struct type_SNMP_GetNextRequest__PDU *get__next__request;

      struct type_SNMP_GetBulkRequest__PDU *get__bulk__request;

      struct type_SNMP_Response__PDU *response;

      struct type_SNMP_SetRequest__PDU *set__request;

      struct type_SNMP_InformRequest__PDU *inform__request;

      struct type_SNMP_SNMPv2__Trap__PDU *snmpV2__trap;

      struct type_SNMP_Trap__PDU *trap;
   }     un;
};
#definefree_SNMP_PDUs(parm)\
```

```
        (void) fre_obj((char *) parm, _ZSNMP_mod.md_dtab[_ZPDUsSNMP],
&_ZSNMP_mod, 1)

struct type_SNMP_PDU {
   integer    request__id;

   integer    error__status;
#defineint_SNMP_error__status_noError    0
#defineint_SNMP_error__status_tooBig     1
#defineint_SNMP_error__status_noSuchName      2
#defineint_SNMP_error__status_badValue 3
#defineint_SNMP_error__status_readOnly 4
#defineint_SNMP_error__status_genErr     5
#defineint_SNMP_error__status_noAccess 6
#defineint_SNMP_error__status_wrongType       7
#defineint_SNMP_error__status_wrongLength     8
#defineint_SNMP_error__status_wrongEncoding   9
#defineint_SNMP_error__status_wrongValue      10
#defineint_SNMP_error__status_noCreation11
#defineint_SNMP_error__status_inconsistentValue 12
#defineint_SNMP_error__status_resourceUnavailable      13
#defineint_SNMP_error__status_commitFailed     14
#defineint_SNMP_error__status_undoFailed15
#defineint_SNMP_error__status_authorizationError16
#defineint_SNMP_error__status_notWritable      17
#defineint_SNMP_error__status_inconsistentName 18

   integer    error__index;

   struct type_SNMP_VarBindList *variable__bindings;
};
#definefree_SNMP_PDU(parm)\
        (void) fre_obj((char *) parm, _ZSNMP_mod.md_dtab[_ZPDUSNMP],
&_ZSNMP_mod, 1)

struct type_SNMP_VarBind {
   struct type_SNMP_ObjectName *name;

   struct type_SNMP_ObjectSyntax *value;
};
#definefree_SNMP_VarBind(parm)\
        (void) fre_obj((char *) parm, _ZSNMP_mod.md_dtab[_ZVarBindSNMP],
&_ZSNMP_mod, 1)

struct type_SNMP_VarBindList {
      struct type_SNMP_VarBind *VarBind;
```

```
    struct type_SNMP_VarBindList *next;
};
#definefree_SNMP_VarBindList(parm)\
        (void) fre_obj((char *) parm, _ZSNMP_mod.md_dtab[_ZVarBindListSNMP],
&_ZSNMP_mod, 1)

struct type_SNMP_Audit {
  struct type_SNMP_DisplayString *source;

  struct        qbuf    *dateAndTime;

  integer    sizeOfEncodingWhichFollows;
};
#definefree_SNMP_Audit(parm)\
        (void) fre_obj((char *) parm, _ZSNMP_mod.md_dtab[_ZAuditSNMP],
&_ZSNMP_mod, 1)

struct type_SNMP_Snmp1157Msg {
  integer    version;
#defineint_SNMP_version_version__1        0

  struct qbuf *community;

  struct type_SNMP_PDUs *data;
};
#definefree_SNMP_Snmp1157Msg(parm)\
        (void) fre_obj((char *) parm, _ZSNMP_mod.md_dtab[_ZSnmp1157MsgSNMP],
&_ZSNMP_mod, 1)

struct type_SNMP_TimeTicks {
  integer    parm;
};
#definefree_SNMP_TimeTicks(parm)\
        (void) fre_obj((char *) parm, _ZSNMP_mod.md_dtab[_ZTimeTicksSNMP],
&_ZSNMP_mod, 1)

struct type_SNMP_Trap__PDU {
  OID    enterprise;

  struct type_SNMP_NetworkAddress *agent__addr;

  integer    generic__trap;
#defineint_SNMP_generic__trap_coldStart  0
#defineint_SNMP_generic__trap_warmStart1
#defineint_SNMP_generic__trap_linkDown 2
```

```
#defineint_SNMP_generic__trap_linkUp     3
#defineint_SNMP_generic__trap_authenticationFailure      4
#defineint_SNMP_generic__trap_egpNeighborLoss 5
#defineint_SNMP_generic__trap_enterpriseSpecific6

   integer    specific__trap;

   struct type_SNMP_TimeTicks *time__stamp;

   struct type_SNMP_VarBindList *variable__bindings;
};
#definefree_SNMP_Trap__PDU(parm)\
        (void) fre_obj((char *) parm, _ZSNMP_mod.md_dtab[_ZTrap_PDUSNMP],
&_ZSNMP_mod, 1)
#endif
```