



CINVESTAV-IPN
Biblioteca de Ingeniería Eléctrica

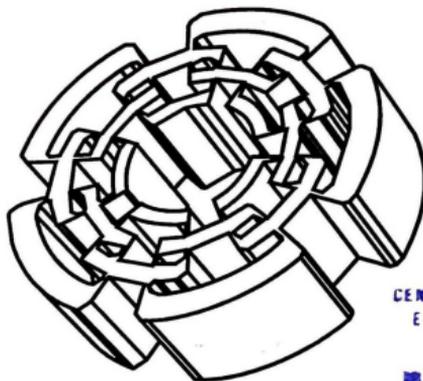


FB0000011714

CENTRO DE INVESTIGACION Y DE
ESTUDIOS AVANZADOS DEL
I. P. N.
BIBLIOTECA
INGENIERIA ELECTRICA

Centro de Investigación y de Estudios Avanzados del I.P.N.

Departamento de Ingeniería Eléctrica
Sección de Computación



CENTRO DE INVESTIGACION Y DE
ESTUDIOS AVANZADOS DEL
I. P. N.
BIBLIOTECA
INGENIERIA ELECTRICA

UNA PROPUESTA DE AGENTE Y DE ADMINISTRADOR
PARA LA ADMINISTRACION DE REDES

Tesis que presenta el Ing. Carlos Alberto Guizar Gómez para obtener
el grado de MAESTRO EN CIENCIAS dentro de la especialidad de
INGENIERIA ELECTRICA con opción en COMPUTACION

Trabajo dirigido por el Dr. Héctor Ruíz Barradas



XM

CLASIF:	97.34
ADQUIS:	B1-15248
FECHA:	1998
PROCD:	TESIS 1998
	\$

CLASIF.	
ORIGEN:	DEPTO. SERV. BIBL.
FECHA:	18-11-98
PROCESO:	72315-1998
	\$

CENTRO DE INVESTIGACION Y DE
ESTUDIOS AVANZADOS DEL
I. P. N.
BIBLIOTECA
INGENIERIA ELECTRICA

AGRADECIMIENTOS

- *A mi Madre por que gracias a su esfuerzo, estoy logrando ser una realidad*
- *A Ivonne por todo lo que ha sido, es y será*
- *A mis hermanas y hermanos por su gran influencia en mi vida*
- *A mi Padre, a mi abuelo y a mi tío donde quiera que estén*
- *A mis compañeros de la maestría y a mis amigos que siempre me han apoyado*
- *A la Sra. Sofía, al personal y a los profesores de la sección de computación por su apoyo y enseñanza en mi formación académica*
- *Al CINVESTAV y al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo económico brindado.*

CENTRO DE INVESTIGACION Y DE
ESTUDIOS AVANZADOS DEL
I. P. N.
BIBLIOTECA
INGENIERIA ELECTRICA

RESUMEN

Administrador y agente juegan un papel importante en la administración de redes. Los administradores requieren y capturan de los objetos administrados en los agentes, información administrada utilizando un protocolo para la administración, los agentes son utilizados para obtener información y/o cambiar el estado de los objetos administrados, y adelantar notificaciones de los objetos administrados a los administradores. En esta tesis se seleccionó el protocolo de administración SNMP y se presentan las propuestas de arquitectura y servicios, de un administrador y un agente para la administración de red, así como el desarrollo de los prototipos utilizando el lenguaje JAVA, lo cual les permite que se ejecuten en diferentes plataformas sin problemas y se integren y trabajen fácilmente con productos comerciales para la administración de red.

ABSTRACT

Manager and agent play an important role in network management. Managers request and capture management information of the agents and yours managed objects using a management protocol, agents are used to gather information and change the state of managed objects, and forward notifications of events from managed objects to managers. This thesis select SNMP management protocol and presents the proposal of architecture and service interface of a manager and agent for network management, as well as an implementation of prototypes developed using the JAVA language. Noticeable among them are portability between heterogeneous platforms, and very simple of use with network management commercial products.

INDICE

INTRODUCCION	1
---------------------------	----------

Capítulo 1: ADMINISTRACION DE REDES

1.1 DEFINICION.....	4
1.2 ENFOQUES VIABLES PARA LA ADMINISTRACION DE REDES.....	4
1.3 ADMINISTRACION INTEGRAL DE RED.....	5
1.4 COMPONENTES DE LA ADMINISTRACION DE RED.....	6
1.5 OBJETOS ADMINISTRADOS.....	7
1.6 DEFINICION DE OBJETOS ADMINISTRADOS EN INTERNET.....	8
1.7 DEFINICION DE OBJETOS ADMINISTRADOS EN IEEE.....	9
1.8 DIFERENCIAS ENTRE LAS NORMAS OSI E INTERNET.....	10
1.9 BASE DE DATOS DE LA INFORMACION ADMINISTRADA.....	14
1.10 ARQUITECTURA PARA LA ADMINISTRACION DE RED DE INTERNET.....	14
1.11 ARQUITECTURA PARA LA ADMINISTRACION DE RED DE IEEE.....	15
1.12 SISTEMA DE DOMINIO DE NOMBRES DE INTERNET.....	16
1.13 PROPUESTA A DESARROLLAR.....	17
1.14 PASOS A SEGUIR EN LA PROPUESTA.....	18

Capítulo 2: ESTRUCTURA DE LA INFORMACION ADMINISTRADA

2.1 MANEJO DE LA INFORMACION ADMINISTRADA.....	21
2.2 PRESENTANDO INFORMACION ADMINISTRADA.....	21
2.3 ELEMENTOS DE ASN.1.....	22
2.4 TIPOS Y OBJETOS DE ASN.1.....	25
2.5 REGLAS DE CODIFICACION.....	32
2.6 NOMBRES DE LOS OBJETOS.....	44
2.7 CONCISA DEFINICION DE SMI.....	47

Capítulo 3: BASE DE DATOS DE LA INFORMACION ADMINISTRADA

3.1 MIB DE INTERNET.....	52
3.2 JERARQUIA DE NOMBRAMIENTO DE INTERNET.....	52
3.3 TIPOS Y SINTAXIS DE LA MIB.....	53
3.4 ESTRUCTURA DE LA MIB.....	54
3.5 REVISION DE LOS GRUPOS DE OBJETOS.....	55
3.6 NOTACIONES PARA LOS OBJETOS.....	56
3.7 MODELOS PARA DEFINIR OBJETOS.....	57
3.8 MIB DE ALTO NIVEL.....	59
3.9 MIB EN MAS DETALLE.....	60
3.10 OTRAS MIBS.....	61

Capítulo 4: PROTOCOLO SNMP

4.1 OBJETIVOS Y ARQUITECTURAS.....	63
4.2 OPERACION DE SNMP.....	65
4.3 UNIDADES DE DATOS DEL PROTOCOLO.....	68
4.4 DEFINICION ASN.1 DE SNMP.....	78

Capítulo 5: PROPUESTAS DE ADMINISTRADOR Y DE AGENTE

5.1 PROPUESTA DE AGENTE.....	82
5.2 PROPUESTA DE ADMINISTRADOR.....	88

Capítulo 6: PROTOTIPOS

6.1 LENGUAJE DE PROGRAMACION.....	93
6.2 ESTRUCTURA DE CLASES.....	94
6.3 PROTOTIPO DEL ADMINISTRADOR.....	96
6.4 PROTOTIPO DEL AGENTE.....	98
6.5 CODIGOS FUENTE.....	99

CONCLUSIONES	102
---------------------------	-----

APENDICES

Apendice A: JAVA.....	106
Apendice B: PRODUCTOS COMERCIALES.....	110
Apendice C: CODIGOS FUENTE.....	115
Apendice D: OBJETOS DE LA MIB.....	171

GLOSARIO	190
-----------------------	-----

BIBLIOGRAFIA	199
---------------------------	-----

INTRODUCCION

En los últimos tiempos, el mundo de la computación con sus diferentes ramas se encuentra en medio de una gran evolución tecnológica que sin lugar a dudas todavía nos reserva muchos cambios. Las redes son una de estas ramas y sufre igualmente ésta evolución, lo cuál obliga a mejorar y simplificar constantemente las técnicas, herramientas y componentes utilizados en éstas.

La administración es uno de los aspectos más complejos en la operación de una red. La administración de redes aparece como una necesidad ante el crecimiento incesante en tamaño y complejidad de estos medios modernos de comunicación e inicia su auge a principios de 1994. Su complejidad a provocado que sólo un selecto grupo de investigación y los fabricantes de redes más importantes del mundo, hayan desarrollado normas y productos para la administración de redes. Lo que generó que existan varias normas en este campo. Sin embargo, no obstante la heterogeneidad en las normas de administración, éstas pueden abstraerse en un esquema que consta de tres componentes principales: El administrador que dirige las operaciones en los agentes, el agente que informa las condiciones de los elementos administrados en la red y la información administrada. Diferentes fabricantes proponen productos sobre un esquema particular con grandes capacidades de administración, sin embargo éstos presentan cuatro grandes inconvenientes: primero su complejidad de instalación y utilización, segundo su falta de transparencia para integrar varios sistemas, tercero cada producto está desarrollado para trabajar en una plataforma(sistema operativo) de red en particular y cuarto, la subordinación a las propuestas y los productos propios de cada fabricante, si se desean aumentar o cambiar las capacidades de un componente, o desarrollar uno nuevo.

Con el fin de corregir tales inconvenientes, en la presente tesis se proponen componentes y se desarrollan prototipos de estos componentes de administrador y de agente, que permiten integrarse a productos para la administración de redes ya existentes. Además, se proporciona la información necesaria para que cualquier profesionista de las redes, logre comprender como se constituye y como funciona la administración de redes.

El orden para comprender la administración de redes y realizar este trabajo se muestra a continuación capítulo a capítulo. En el capítulo 1, se muestra la definición de administración de redes, un vistazo a los diferentes estándares para ésta y sus diferencias entre si. Así cómo, la selección del estándar a utilizar y los pasos a seguir en nuestra propuesta. En el capítulo 2, se habla de la estructura de la información administrada la cuál proporciona un mecanismo para describir y nombrar los objetos que son administrados, además, permite que los valores de los objetos sean conocidos y manipulados. Esto se logra utilizando un lenguaje para la descripción de mensajes, conocido como la notación de sintaxis abstracta uno (ASN.1). ASN.1 es utilizado para definir la sintaxis, o forma de un mensaje de administración, una vez que ésta ha sido especificada, las reglas básicas de codificación codifican el mensaje en un formato que puede ser transmitido por una red. En

el capítulo 3, se habla de la base de datos de la información administrada (MIB), la cuál define más precisamente los objetos administrados y los organiza para que sean fáciles de utilizar. Existen diferentes tipos de MIB dependiendo del fabricante y de su uso. En el capítulo 4, se habla del protocolo simple para la administración de red (SNMP), el cuál proporciona el mecanismo para que los administradores se comuniquen con los agentes. Esta comunicación involucra leer los valores de los objetos en una MIB y alterar los valores cómo sea apropiado. En otras palabras, administrar los objetos. En el capítulo 5, se habla de las propuestas realizadas de administrador y de agente, así como de la arquitectura de cada una de ellas, la descripción de sus componentes y la relación entre éstos. En el capítulo 6, se habla de los prototipos desarrollados, así cómo, del lenguaje de programación utilizado, de las clases y de los códigos desarrollados y por último, de las pruebas realizadas con los prototipos y productos comerciales.

CAPITULO 1

ADMINISTRACION DE REDES

1.1 DEFINICION DE LA ADMINISTRACION DE RED

Diferentes organizaciones tienen su punto de vista en administración de redes y, por lo tanto, emplean diferentes definiciones del término. Quizás el enfoque más útil es pedir prestado una definición de una escuela comercial de administración, debido a que la definición es aplicable para la administración de redes.

Hoy está comúnmente aceptado que la administración involucra la planeación, organización, monitoreo, contabilidad, y el control de actividades y recursos. Ésta definición puede estar ciertamente aplicada a la administración de redes. Sin embargo, las estructuras para la administración de redes de **OSI** e **INTERNET** están enfocadas principalmente en monitorear, contabilizar, y controlar las actividades y recursos de las redes.

Los otros dos aspectos de la administración de redes, aquellos de planear y organizar, no están involucrados en el plan de OSI/INTERNET, pero son los aspectos más importantes de la administración de redes y consume la mayoría de los recursos de la organización. Sin duda, si la red no está planificada y adecuadamente organizada, cualquier cantidad de monitoreo, contabilidad, y control será inútil.

1.2 ENFOQUES VIABLES PARA LA ADMINISTRACION DE RED

Muchas compañías operan en un entorno heterogéneo y usan una amplia variedad de componentes de hardware, así como diferentes protocolos de comunicaciones. Las organizaciones se están enfrentando con el gran problema de construir paquetes únicos de interconexión para cada uno de sus sistemas específicos. Esto no es sólo una tarea compleja, sino que también requiere recursos importantes. Este panorama se pondrá más complejo si se consideran factores como redes de voz, vídeo y datos que crecen dentro de las organizaciones. Con lo anterior, las relaciones entre cliente, y proveedor crecen cada día, donde es necesario soportar diferentes interfaces y protocolos.

OSI es cada vez más utilizado en la industria de las telecomunicaciones para definir el marco de comunicación entre sistemas heterogéneos. Sin embargo, los protocolos de INTERNET todavía son más prevalentes en la industria. Ante este panorama la IEEE ha encaminado sus esfuerzos a pavimentar el camino para normalizar los sistemas LAN y WAN, lo que permitirá una mayor transparencia entre los sistemas.

Estas normas no sólo facilitan la tarea de interconectar diferentes computadoras y demás componentes de una red, también dan al usuario de red más flexibilidad en la selección de equipo y de software, porque definen interfaces y protocolos estándares entre los diferentes proveedores. Con esto, el interés del usuario por adquirir un componente estandarizado aumenta.

Estas normas permiten una plataforma común para el desarrollo de aplicaciones de usuario, software de apoyo, y aplicaciones de administración de red. Este enfoque simplifica las interfaces entre el centro de control de red y los recursos de red administrados del usuario.

1.3 ADMINISTRACION INTEGRAL DE RED

Un objetivo importante de la administración, es tener un enfoque integral de la administración de una o varias redes, las cuales constan de computadoras, paquetes de software, y equipo de comunicaciones de diferentes fabricantes. Varios puntos clave acerca de la administración integral de red tienen que estar enfatizados.

- La administración integral de red es necesaria para reducir los costos de interconexión de diferentes sistemas.
- Debido a que la integración es necesaria, la administración de red requiere tener un intercambio uniformizado de información administrada entre los diferentes productos de los fabricantes.
- La administración integral de red no significa que los servicios sean los mismos para todos los usuarios, significa que un nivel básico de servicio de administración es el mismo para las definiciones de desempeño de red, contabilidad, configuración, seguridad, criterios de fallas y el intercambio de unidades de información de protocolos comunes.
- Integración y uniformidad no previenen servicios de valor agregado que surjan en el futuro dentro del contexto de estas normas, lo anterior debido a la evolución que se está dando en la administración.

Para colocar estos puntos en perspectiva, imagine los beneficios potenciales de utilizar enfoques de administración de red comunes con Ethernet, TCP/IP, teléfono de voz, Interfaces de Datos Distribuidos en Fibra (FDDI), Red Digital de Servicios Integrados (ISDN), ATM, Frame Relay, entre otros. Actualmente, la gran mayoría de paquetes de administración difieren a través de estas redes.

El objetivo clave de las normas de administración de red es desarrollar un conjunto integrado de procedimientos y normas que apliquen de igual manera a diferentes fabricantes y redes.

Unos de los aspectos más caros de la administración de red es el software de administración de red, no solamente en su adquisición y/o desarrollo, sino también en su mantenimiento. El software común puede guiarnos a reducir substancialmente los costos y a un entorno menos complejo.

1.4 COMPONENTES DE LA ADMINISTRACION DE RED

Las normas de administración de red de OSI, INTERNET, e IEEE definen la responsabilidad para un **proceso administrador** (llamado un **sistema de administración de red** en algunos productos de fabricantes) y un **agente administrador** (también conocido como un proceso agente).

En el sentido estricto, un sistema de administración de red sólo contiene protocolos que transportan información de los elementos de la red hacia los diferentes agentes en el sistema y de ahí al proceso administrador.

Otro componente vital para un sistema de administración de red, es la **base de información administrada** o **biblioteca** (llamada en adelante **MIB**) de sus siglas en ingles. Este objeto conceptual es realmente una base de datos que está compartida entre administradores y agentes para proveer información acerca de los elementos de red administrados.

Estos conceptos importantes se resumen con las siguientes notaciones:

- **AGENTE:** Informa al proceso administrador la condición de los elementos de la red administrados y reciben del proceso administrador las acciones para desempeñar en estos elementos
- **PROCESO ADMINISTRADOR:** Dirige las operaciones de los agentes
- **MIB:** Determina la estructura y contenido de la información administrada utilizada por el proceso administrador y por el agente

La figura 1-1 muestra una configuración típica en una red local mostrando la ubicación del proceso administrador, el software agente, y la MIB. Las normas de administración de red de OSI, INTERNET, e IEEE no requieren que estén situadas en una ubicación particular de la red.

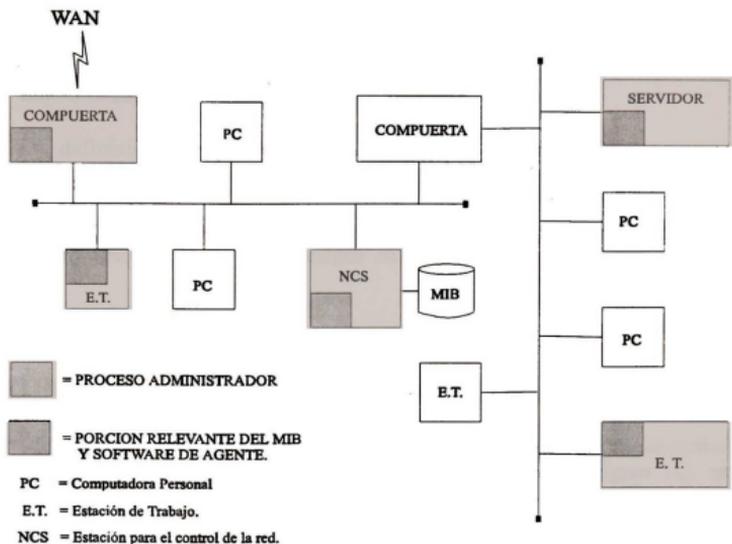


Figura 1-1 Localización de los componentes a administrar en la red.

En configuraciones actuales, el software **agente** está situado generalmente en componentes como servidores, computadoras, puentes, y enrutadores. Típicamente, un puesto de mando de red (NCS) actúa como el **proceso administrador**. La MIB está localizada generalmente en el NCS y la parte de la MIB que es para el agente está localizada en el agente.

1.5 OBJETOS ADMINISTRADOS

Los recursos que son supervisados y controlados por la administración de red son llamados **objetos administrados**. Un objeto administrado puede ser cualquier elemento considerado importante por organizaciones que están utilizando las normas de administración de red. Como ejemplos de objetos "hardware" administrados podemos citar: Interruptores, estaciones de trabajo, tarjetas de red y puertos de comunicación multiplexores. Como ejemplos de objetos "software" administrados podemos citar: programas de colas, algoritmos de enrutamiento, y rutinas de administración de memoria.

Otros aspectos importantes de los objetos administrados son:

- Las operaciones permisibles de administración que pueden realizarse en un objeto tienen que formar parte de su definición.
- La definición de un objeto puede incluir el efecto que las operaciones de administración tienen en los recursos relacionados del sistema.
- El estado del objeto o sus propiedades, determinan el tipo de operación de administración que se puede realizar en él.

1.6 DEFINICION DE OBJETOS ADMINISTRADOS EN INTERNET

Desde la perspectiva de INTERNET un objeto administrado está descrito de una manera más concreta que en el modelo OSI. Como se muestra en la figura 1-2, un objeto administrado está descrito por:

- La SINTAXIS utilizada para modelar el objeto,
- El nivel de ACCESO permitido al objeto,
- Su ESTADO requisito para la instrumentación del objeto y
- Un NOMBRE no ambiguo del objeto.



FIGURA 1-2. OBJETOS ADMINISTRADOS DE INTERNET

1.6.1 SINTAXIS

En el modelo de INTERNET, la sintaxis define el tipo de dato. En contraste a la administración de red de OSI, el tipo de INTERNET está bastante limitado. Estos tipos se definen con Notación de Sintaxis Abstracta 1 (ASN.1) por convención y pueden ser entero, cadena de octeto, o secuencias. Además, la MIB de INTERNET aclara que el uso de CADENA DE OCTETO solo se hace en el contexto de caracteres imprimibles. Por consiguiente, una cadena desplegada puede ser identificada como una CADENA DE OCTETO en la cual el conjunto de caracteres ASCII es definido.

1.6.2 ACCESO

La definición de acceso provee información de cómo un objeto administrado puede ser acezado. Las siguientes operaciones están permitidas en un objeto administrado:

- De sólo lectura: Las instancias del objeto pueden ser leídas pero no escritas.
- Lectura escritura: Las instancias del objeto pueden ser leídas o escritas.
- Sólo escritura: Las instancias del objeto pueden ser escritas pero no leídas.
- No accesible: Las instancias del objeto pueden ser no leídas ni escritas.

1.6.3 ESTADO

El estado del objeto está definido actualmente como (1) *Obligatorio*, cada nodo administrado es requerido para generar o activar su objeto administrado; (2) *Opcional*, los nodos son o no requeridos para implantar el objeto o (3) *Obsoleto*, objetos administrados, no están definidos ampliamente y los nodos necesarios no están implantados.

1.6.4 NOMBRE

La definición de nombre es un IDENTIFICADOR DE OBJETO en la notación ASN.1, que es utilizado para identificar de manera única un objeto administrado. Veremos en breve cómo un nombre está derivado de una jerarquía denominadora utilizada en INTERNET y en ISO.

1.7 DEFINICION DE OBJETOS ADMINISTRADOS IEEE

Considerando la definición de los objetos administrados, los estándares de la IEEE se asemejan mucho a los del modelo OSI. En los distintos documentos de OSI están incluidos los estándares del administrador IEEE 802.1 de LAN/MAN. Es posible que las

operaciones mencionadas anteriormente citadas en la sección de OSI, puedan ser utilizadas. Pero las operaciones de la IEEE también incluirán el siguiente grupo:

- *Captura*: Obtiene un valor de un objeto identificado.
- *fijar*: Fija un valor sobre un objeto.
- *Comparar y fijar*: Ejecuta un conjunto de pruebas, y si las pruebas tienen éxito, entonces el objeto será fijado con un valor en particular.
- *Acción*: Ejecuta una secuencia de operaciones sobre un objeto y/o requerir que este objeto transite hacia un estado ya identificado. Esta operación requiere que la información sea regresada considerando el éxito o falla de la operación.
- *Evento*: Esta operación no es iniciada desde un servicio de usuario; más bien, es un evento localmente inicializado por la capa de entidad administrada (LME), (algo así como un mensaje sin haberlo solicitado - una interrupción).

1.8 DIFERENCIAS ENTRE LAS NORMAS OSI E INTERNET

Existen muchas diferencias de representación de datos y de operación entre las normas para la administración de redes, a continuación sólo se mencionarán las más importantes.

1.8.1 CLASES DE OBJETOS Y GRUPOS

Desde la perspectiva de OSI, los objetos administrados que tienen características similares son agrupados en una **clase de objeto**, que es llamada una **clase de objeto administrada** (MOC). En OSI, las características utilizadas para determinar una clase de objeto son los atributos, las operaciones, y las notificaciones.

MOC proporciona medios convenientes para agrupar recursos relacionados, y a través del uso de técnicas, denominadas jerárquicas, es posible derivar nuevas clases de clases ya existentes. Esto significa que es posible encapsular (contener) objetos dentro de otros objetos y, haciendo así, invocar operaciones o recibir notificaciones solamente en la "capa" relevante de los objetos encapsulados de esa manera.

El modelo INTERNET no trata con clases de objetos como el modelo OSI. El enfoque más cercano a una clase de objetos en INTERNET es un **grupo**. Para propósitos de

clasificación, INTERNET administra grupos de objetos. Por ejemplo, el grupo de interfaces incluye todos los objetos que pertenecen a un enlace de información físico o interfaz de capa, como ETHERNET, Control de Enlace de Información Síncrono (SDLC), etc. Como otro ejemplo, todos los objetos que pertenecen a protocolos de administración de red son clasificados en el grupo del protocolo simple para la administración de red (SNMP).

1.8.2 ARBOL DE INFORMACION ADMINISTRADA Y SISTEMA DE DOMINIO DE NOMBRES

La figura 1-3 muestra uno de los principales conceptos de administración de red de OSI y de INTERNET. En el modelo OSI, éste es llamado el árbol de información administrada (MIT); En INTERNET éste es llamado el sistema de dominio de nombres (DNS).

Base Object/Root

Nivel 1

Nivel 2

Nivel 3

Nivel 4

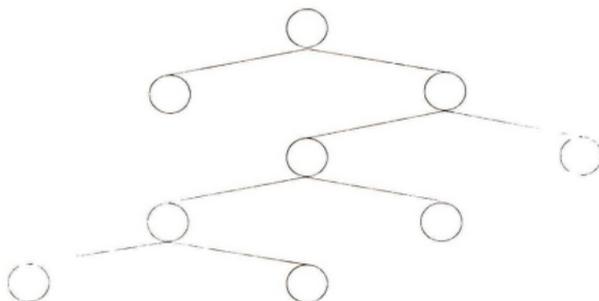


FIGURA 1-3. Vista general del MIT//DNS.

MIT/NDS se utiliza como el fundamento para la identificación y administración de recursos (objetos) en una red. El árbol es una taxonomía jerárquica. La parte superior del árbol es la raíz, también conocido como un objeto base. Descendiendo abajo en el árbol son nodos identificados como nivel 1, 2, 3, y así sucesivamente.

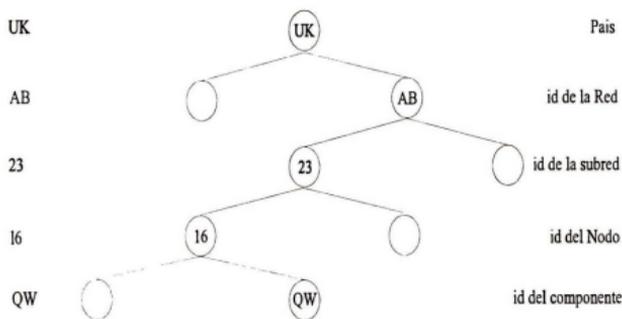
Estos nodos están considerados subordinados a los nodos sobre ellos. El otro término para describir ésta relación superior--subordinado es el árbol padre--hijo. Los nodos en los niveles medios pueden ser hijos de un nodo anterior y padres de nodos debajo.

La idea del árbol jerárquico es proveer una estructura para la identificación y nombramiento de objetos. Para uso posterior, el lector deberá observar que aquellos nodos sin entradas subordinadas (es decir, el fin de la relación jerárquica) son llamados entradas de hoja.

1.8.3 NOMBRE RELATIVAMENTE DISTINGUIDO Y NOMBRE DISTINGUIDO

Cada etiqueta de nivel es conocida como el **nombre relativamente distinguido** (RDN). Este término es bastante útil, porque permite el uso apropiado de las etiquetas dentro de la jerarquía, para lograr una identificación no ambigua. Por ejemplo, en la figura 1-4 el nombre relativamente distinguido, QW, es ciertamente suficiente para identificar sin ambigüedad el componente id dentro del nodo. Por otra parte, el ID del componente, QW, es probablemente insuficiente para únicamente identificar el componente a través de diferentes redes.

Nombre Relativamente Distinguido



Nombre Distinguido: País= UK, id de la red= AB, id de la subred= 23, id del nodo=16, id del componente=QW
o bien: UK.AB.23.16.QW

FIGURA 1-4. Nombres de objetos

El administrador de red puede utilizar el RDN de bajo nivel dentro de un subdominio específico de una red y no tener que estar involucrado con los nombres largos (y búsquedas pesadas en directorios) asociados con identificadores de nivel superior.

El identificador completo para el árbol en estas figuras es conocido como un **nombre distinguido** (DN). Este es designado porque permite la identificación completamente no ambigua de un objeto a través de la jerarquía. La creación del DN es un

proceso muy simple. La notación bajo el árbol nos permite encadenar las etiquetas asociadas con cada nodo para producir el DN de: UK. AB. 23.16. QW.

La administración de red de OSI y las normas de INTERNET utilizan el plan denominado jerárquico para buscar en el MIB y para acceder la información de las bibliotecas acerca de los objetos administrados. Si se busca en el árbol completo (lo cual podría ser un proceso largo), la búsqueda comienza en la raíz del árbol. Esto no es necesario en muchas instancias, para búsquedas desde la raíz, puede ser posible comenzar la búsqueda en algún nodo medio en el árbol con un RDN. No obstante, el término de OSI **objeto base administrado** se refiere a una parte del subárbol donde una búsqueda comienza.

1.8.4 ATRIBUTOS Y VALORES, Y TIPOS E INSTANCIAS

El MIT de OSI puede ser aplicado al concepto de atributos y valores de atributos. Como se muestra en la figura 1-5, cada nodo en el árbol es llamado una entrada, que en el contexto de administración de red de OSI, es un objeto administrado. Una entrada es "explotada" con el fin de examinar sus contenidos. Contiene uno o varios atributos. Cada atributo es identificado por su tipo, que puede ser utilizado como un identificador (como un atributo id). Enseguida del tipo está el valor o los valores del atributo.

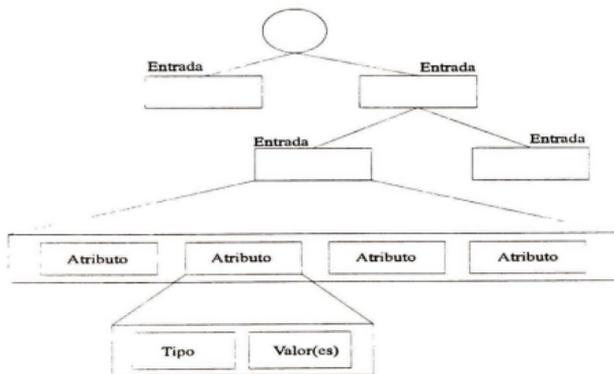


Figura 1-5. Atributos y Valores de los atributos

Los conceptos de INTERNET son bastante similares a los de OSI, todos estos son utilizados excepto el término **tipos de objeto e instancias**. Un tipo de objeto es la definición de un objeto administrado (su sintaxis, modos de acceso, etc.). Una instancia de

objeto es una ocurrencia o instanciación del tipo de objeto. Por ejemplo, un tipo de objeto puede ser definido para una tabla de interfaces, y las entradas en la tabla son una instancia del tipo.

1.9 BASE DE DATOS DE LA INFORMACION ADMINISTRADA

La MIB es un conjunto de información administrada sobre el sistema abierto (esa porción del sistema de organización que se adhiere a las normas de OSI). Ésta ha sido también adoptada por las normas de INTERNET e IEEE, por lo que muchas de las estructuras MIB de OSI también son encontradas en las MIBs de INTERNET e IEEE.

La MIB de OSI es conceptual en naturaleza, y en menor grado, también lo es la MIB de INTERNET. (IEEE no ha definido una MIB.) A pesar de este aspecto abstracto de las MIBs, ellas forman el componente clave de la administración de red. Después de todo, es en la MIB la definición de los objetos administrados. La MIB contiene sus nombres, su comportamiento permisible, y las operaciones que pueden estar desempeñado cada uno de ellos. Posteriormente, las MIBs y los objetos administrados definidos en éstas, se verán con detalle.

1.10 ARQUITECTURA PARA LA ADMINISTRACION DE RED DE INTERNET

En la figura 1-6, se muestran las capas de la arquitectura de INTERNET para las normas de administración de red, ésta arquitectura es más simple que la de OSI. El protocolo SNMP forma parte de los fundamentos para la arquitectura. Las aplicaciones de administración de red no están definidas en las especificaciones de INTERNET. Estas aplicaciones consisten de módulos de administración de red específicos de los fabricantes, tales como administración de fallas, control de conexiones, seguridad y guías de auditoría, entre otros.



Figura 1-6. Capas de INTERNET para la administración de red.

Como se ilustra en la figura 1-6, el SNMP descansa sobre el Protocolo de datagrama de usuario (UDP) y éste se implanta encima del protocolo IP, el cual se programa sobre las capas más bajas (la capa de enlace de información y la capa física).

1.11 ARQUITECTURA PARA LA ADMINISTRACION DE RED IEEE

La figura 1-7 muestra la arquitectura para los estándares de administración de red LAN/WAN de IEEE. Principalmente, este trabajo es patrocinado por el comité 802.1 y los fabricantes de productos están encontrando su utilidad. Ya que su principal atracción es la simplicidad y eficiencia que otorga a la serie del protocolo.

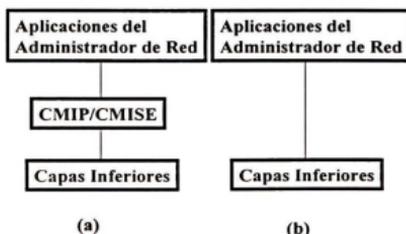


Figura 1-7 Las capas de administración para LAN/MAN de la IEEE

Como se muestra en la figura 1-7a, la arquitectura consiste del protocolo CMIP (protocolo para la administración de redes del estándar OSI) sobre las capas más bajas de IEEE. (En una LAN/MAN de IEEE, sólo las capas física y de enlace de datos son incorporadas dentro de los estándares de la LAN/MAN de IEEE).

Alternativamente, la administración puede ocurrir sin el uso del protocolo CMIP, como se ilustra en la figura 1-7b. Por lo tanto, el estándar de administración de LAN/MAN ubica al CMIP sobre la capa más baja o invocando los esquemas de administración de red dentro de las dos capas más bajas.

1.12 SISTEMA DE DOMINIO DE NOMBRES DE INTERNET (DNS)

La clasificación jerárquica de DNS se muestra en figura 1-8. Como el MIT de OSI, está organizado alrededor de una raíz y una estructura de árbol. Cada etiqueta de un nodo en el árbol al mismo nivel de nodo tiene que ser completamente no ambigua y distinta, es decir, la etiqueta tiene que ser un nombre relativamente distinguible a este nivel de nodo.

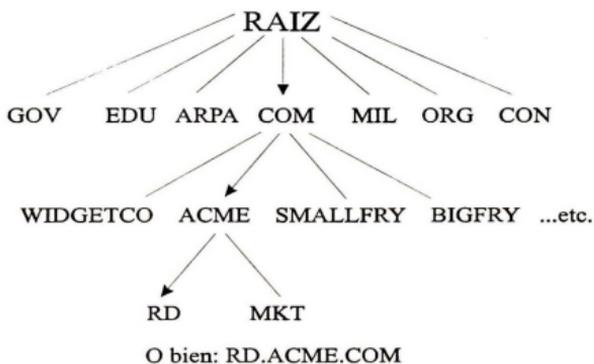


Figura 1-8. CLASIFICACION JERARQUICA DEL DNS

Actualmente, el DNS contiene siete nombres de dominios de nivel superior, los cuales se muestran en la figura 1-8 y son como siguen:

GOV	Identifica cualquier cuerpo del gobierno en los Estados Unidos de Norteamérica.
EDU	Identifica una institución educativa
ARPA	Identifica un huésped ARPANET-INTERNET
COM	Cualquier empresa comercial
MIL	Organizaciones militares
ORG	Cualquier otra organización no identificada por descriptores previos
CON	Identificación de países utilizando el estándar de ISO para nombres de países (ISO 3166)

Existen otro dominios designados a cada país, por ejemplo:

MX	Para el registro de todas las entidades en México
UK	Para el registro de todas las entidades en el Reino Unido, etc.

1.13 PROPUESTA A DESARROLLAR

Una vez analizadas las diferentes normas para la administración de red. Se hace necesario seleccionar una de éstas, para desarrollar el presente trabajo. La norma seleccionada es la de INTERNET la cual tiene asociado el Protocolo Simple para la Administración de Red (SNMP), esto por las razones que se explican a continuación:

- Participación en el mundo real (comercial)
- Simplicidad
- Documentación existente

1.13.1 MUNDO REAL (COMERCIAL)

Este elemento es el más importante que se considero para la selección. Cuando se desarrolla una aplicación es muy importante la universalidad que ésta tendrá al momento de su liberación.

En la actualidad, el mercado mundial de la administración de redes está distribuido como sigue: el 15% se rige por la norma de OSI y el 85% se rige por la norma de INTERNET, lo cual cada día es más notorio, un ejemplo es el acceso a lo que se llama la super carretera de la información y los servidores WEB.

1.13.2 SIMPLICIDAD

Este elemento es el segundo en importancia que se considero para seleccionar la norma. Y se observó que la complejidad de la administración en OSI es mucho mayor que la complejidad de la administración en INTERNET. Un ejemplo de esto es que, la norma de OSI tiene asociado el protocolo CMIP que cuenta con un conjunto de siete instrucciones para realizar todo su trabajo. Mientras que la norma INTERNET tiene asociado el protocolo SNMP que cuenta con un conjunto de cuatro instrucciones.

1.13.3 DOCUMENTACION

En este elemento se observó y analizó la cantidad de bibliografía, artículos escritos y una gran cantidad de la información en general que existe para esta norma. La cual está mejor redactada que la documentación de la norma de OSI. Un elemento que también influyo es la complejidad de la documentación.

1.14 PASOS A SEGUIR EN LA PROPUESTA

El trabajo a desarrollar es proponer componentes de administrador y de agente, así cómo probarlos desarrollando sus prototipos, los cuales deben permitir de manera clara y sencilla, ser integrados a un sistema para la administración de redes independientemente de la plataforma. Teniendo como consecuencia, que se realice una administración de cada uno de los nodos conectados a una red, sencilla y fácilmente sin estar limitados por la plataforma utilizada.

Para todo lo anterior se seguirán los siguientes pasos:

- Proposición de un administrador y un agente bajo la norma INTERNET y el protocolo SNMP.
- Definición del alcance de cada una de las propuestas, esto es, el agente para la administración de nodos y el administrador para la red.
- Selección de un lenguaje de programación y desarrollo de los prototipos que muestren la utilidad de cada propuesta.
- Aplicación de las propuestas en casos prácticos, con aplicaciones comerciales.

Es importante mencionar que dentro de cada uno de estos pasos, se tienen que contemplar una gran cantidad de bases de las cuales hay que partir, para generar la propuesta. Estas serán dadas conforme se avance en este trabajo, ya que requieren cierto orden en su estudio. A continuación en la figura 1-9, se muestran en general los componentes de la administración de redes, en una conexión en particular.

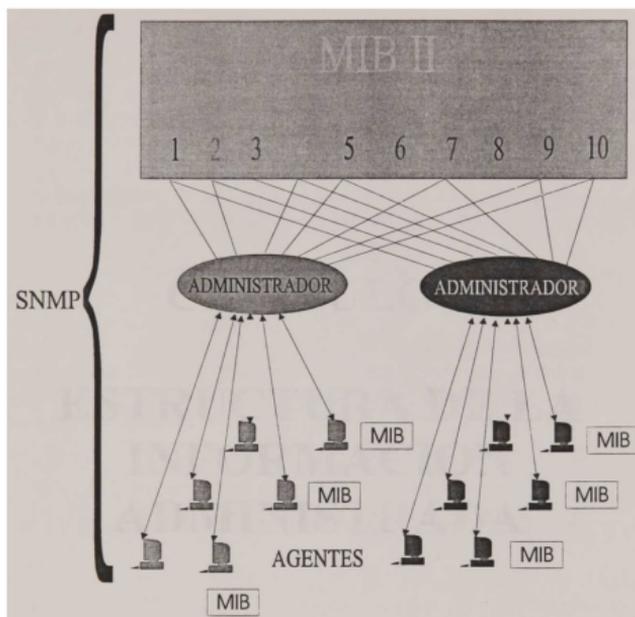


Figura 1-9. Componentes para la administración de red y su relación.

En la figura 1-9, se puede observar un sistema para la administración de redes, con dos administradores que utilizan la misma MIB, cada uno de los administradores en este caso, tiene un grupo de agentes a administrar independiente del otro. Y toda la transferencia de información la realizan vía el protocolo SNMP.

CAPITULO 2

ESTRUCTURA DE LA INFORMACION ADMINISTRADA

En este capítulo, se habla acerca de la estructura de la información administrada (SMI), la cual define las reglas para identificar objetos administrables. La SMI está descrita en la RFC 1155 [2-1], y refinada en la RFC 1212 y RFC 1215.

2.1 MANEJO DE LA INFORMACION ADMINISTRADA

En la administración de redes, el manejo de los objetos de red debe ser accesible física y lógicamente. Por *físicamente accesible* se entiende que algunas entidades administradoras deben verificar físicamente la dirección, contar los paquetes, o de otra manera cuantificar la información administrada en la red. El *acceso lógico* significa que la información administrada debe ser almacenada en algún lugar, y por lo tanto, que la información debe ser recuperable y modificable (SNMP actualmente desempeña la recuperación y modificación). La estructura de la información administrada (SMI) organiza, nombra y describe la información de manera que pueda accederse lógicamente.

La SMI indica que cada objeto administrable debe contar con un nombre, una sintaxis y una codificación. El nombre, un identificador de objeto (OID), únicamente identifica el objeto. La sintaxis define el tipo de dato, como por ejemplo un entero o una cadena de octetos. La codificación describe cómo la información asociada con el objeto administrable es tratada para la transmisión entre máquinas. A continuación se describen la sintaxis, la codificación, y finalmente los nombres de los objetos. De esta manera, se va de lo abstracto hacia lo concreto.

2.2 PRESENTANDO INFORMACION ADMINISTRADA

En términos del modelo ISO/OSI [2-4], la sintaxis ASN.1 (Notación de Sintaxis Abstracta), funciona en la capa de Presentación (capa 6). Recuerde que la capa de Presentación define el formato de los datos que son enviados a través de la red y de la seguridad de los mismos.

Para que los administradores y agentes intercambien datos en orden, ambos deben entender ésto, sin importar la manera cómo la máquina representa los datos internamente. Para que esto ocurra, dos aspectos deben ser normalizados, **la sintaxis abstracta** y **la sintaxis de transferencia**. La *sintaxis abstracta* define las especificaciones para la notación de datos. La *sintaxis de transferencia* define codificaciones transmisibles para los elementos en la sintaxis abstracta.

La SMI de INTERNET especifica que ASN.1 define la sintaxis abstracta para mensajes, esto es, ASN.1 define los elementos básicos del lenguaje y provee reglas para combinar elementos dentro de mensajes. Las Reglas Básicas para Codificación (BER) proporcionan la sintaxis de transferencia. Las BER están asociadas con la sintaxis abstracta y proporcionan comunicación entre máquinas a nivel bit. Así, la SMI y el SNMP utilizan

las normas ASN.1 y BER para definir varios aspectos de la estructura de administración de la red INTERNET.

2.3 ELEMENTOS DE ASN.1

Los administradores de redes algunas veces critican la ASN.1 por su complejidad. Algunas de sus críticas son justas: es bastante difícil interpretar el estándar. Aunque, la ASN.1 tiene un objetivo directo, está diseñada para definir mensajes con información estructurada de una manera independiente de la máquina. Para hacer esto, ASN.1 define tipos de datos básicos, tales como enteros y cadenas, y nuevos tipos de datos que están basados en combinaciones de los básicos. Las BER entonces definen la manera en que los datos son codificados para la transmisión.

La ASN.1 define los datos como un patrón de bits en la memoria de la computadora, tal como un lenguaje de programación de alto nivel define datos que el lenguaje manipula como variables. Las BER definen una manera estándar para convertir las definiciones de la ASN.1 en patrones de bits para transmisión, y entonces ellas son las que realmente transfieren los datos entre las computadoras. Las BER son necesarias porque la norma ASN.1 es “leíble por el hombre” y deben ser traducidas diferente para cada tipo de computadora. La representación BER, es siempre la misma para cualquier descripción ASN.1, sin importar si las computadoras envían o reciben información. Esto asegura la comunicación entre computadoras, sin importar su arquitectura interna.

El objetivo aquí es describir la ASN.1 al nivel de detalle necesario para aplicarlo a la administración de redes y SNMP, (SNMP utiliza un subconjunto de ASN.1 con el objeto de tener simplicidad). La ASN.1 utiliza algunos términos únicos para definir sus procedimientos, incluyendo los tipos de definiciones, asignación de valores, definiciones de macros, evocaciones y definición de módulos. Se necesitan comprender estos términos antes de poder proceder a discutirlos. Además, la ASN.1 especifica algunas palabras clave, o secuencias de caracteres reservadas. Las palabras clave tales como INTEGER, OBJECT y NULL tienen un significado especial y aparecen en letras mayúsculas.

2.3.1 TIPOS Y VALORES

Un *tipo* es una clase de dato. Este define la estructura de los datos que la máquina necesita para entender y procesar la información. La SMI define tres tipos: Primitivo, Constructor y Definido. En ASN.1 se definen varios *tipos primitivos* (también conocidos como tipos simples), incluyendo INTEGER, OCTET STRING, OBJECT IDENTIFIER, y NULL. Por convención, los tipos inician con una letra mayúscula y después minúsculas. (La ASN.1 también define los cuatro tipos listados aquí como secuencias reservadas de caracteres, y por lo tanto se representan en mayúsculas). Los *tipos constructor* (también conocidos como los tipos aggregate) generan listas y tablas. Los *tipos definidos* son

nombres alternativos para tipos simples o complejos de la ASN.1 y son usualmente más descriptivos. Ejemplos de tipos definidos SNMP incluyen *IpAddress*, el cual representa una dirección INTERNET de 32 bits, y *TimeTicks*, el cual es una marca de tiempo.

El *valor* cuantifica el tipo. En otras palabras, una vez que se conoce el tipo, tal como un INTEGER o un OCTET-STRING, el *valor* proporciona un caso específico para ese tipo. Por ejemplo, un valor puede ser una entrada en una tabla de enrutamiento. Por conveniencia, los valores inician con una letra minúscula.

Algunas aplicaciones sólo permiten un subconjunto de los posibles valores de un tipo. Una *especificación de subtipo* se indica como una limitación. La especificación de subtipo aparece después del tipo y muestra el valor o valores permitidos, llamados los *valores de subtipo*, entre paréntesis. Por ejemplo, si una aplicación utiliza un tipo INTEGER y los valores permisibles deben adaptarse dentro de un campo de 8 bits, el rango posible de valores debe encontrarse entre 0 y 255. Se podría expresar esto como

INTEGER (0..255)

Los dos puntos (..) son el separador de rango e indican la validez de cualquier valor entero entre 0 y 255.

2.3.2 MACROS

La utilización de macros permite extender el lenguaje ASN.1. Por convención, una referencia a nombre de macro aparece enteramente en letras mayúsculas. Por ejemplo, las definiciones MIB hacen extensivo el uso de la macro ASN.1, OBJECT TYPE (originalmente definido en la RFC 1155 y ahora reemplazado por la definición en la RFC 1212). El primer objeto en la MIB es una descripción del sistema (*sysDescr*). La RFC 1213 utiliza la macro OBJECT-TYPE para definir *sysDescr* de la siguiente manera :

sysDescr OBJECT-TYPE

SYNTAX DisplayString (SIZE (0..255))

ACCESS read-only

STATUS mandatory

DESCRIPTION

“Una descripción textual de la entidad. Este valor debe incluir el nombre completo e identificación de la versión del tipo de hardware del sistema, software de sistema operativo, y software de red. Esto debe contener solo caracteres ASCII imprimibles”.

:: = {system 1}

Así, una macro define el objeto *sysDescr*. Mas adelante se ven los detalles de la macro OBJECT-TYPE (SINTAX, ACCESS, y otras). Nótese que el rango de la notación (0..255) especifica el tamaño permitido del tipo *DisplayString*, el cual en este caso es entre 0 y 255.

2.3.3 MODULOS

ASN.1 también colecta descripciones dentro de grupos convenientes, llamados módulos. Por ejemplo el módulo RMON (Administración Remota de Objetos de Red) inicia con un nombre, como por ejemplo RFC1271-MIB. Los nombres de los módulos deben iniciar con una letra mayúscula. Las declaraciones BEGIN y END envuelven el cuerpo del módulo. El cuerpo puede contener IMPORTS, los cuales son los nombres de los tipos, valores, macros, y los módulos en los cuales éstos son declarados. A continuación está la sección de encabezado de la RMON MIB (RFC 1271), representando un módulo MIB. Las líneas de comentario dentro de la sintaxis de la ASN.1 inician con un doble guión (--).

```
RFC1271-MIB DEFINITIONS ::=BEGIN
```

```
IMPORTS
```

```
Counter FROM RFC1155-SMI
```

```
DisplayString FROM RFC1158-MIB
```

```
mib-2 FROM RFC1213-MIB
```

```
OBJECT-TYPE FROM RFC-1212;
```

```
-- Este módulo MIB utiliza la macro extendida OBJECT-TYPE  
-- como está definida en la RFC 1212.
```

```
-- Monitoreo Remoto de la Red MIB
```

```
rmon OBJECT IDENTIFIER ::= { mib-2 16 }
```

```
-- convenciones textuales
```

```
.
```

```
.
```

```
END
```

En el ejemplo anterior, se puede apreciar el valor de la notación OBJECT IDENTIFIER para rmon. Más adelante se verá esta notación en detalle, pero por ahora simplemente note que el valor de RMON es el 16º objeto definido bajo el árbol del objeto mib-2. Los corchetes {} indican el inicio y fin de una lista, en este caso una lista de los valores OBJECT IDENTIFIER definidos en rmon.

2.3.4 RESUMEN DE CONVENCIONES ASN.1

En este resumen, ASN.1 hace una distinción entre letras mayúsculas y minúsculas, como sigue:

Artículo	Convención
Tipos	Letra inicial mayúscula
Valores	Letra inicial minúscula
Macros	Todas las letras mayúsculas
Módulos	Letra inicial mayúscula
Palabras clave ASN.1	Todas las letras mayúsculas

Las palabras clave de ASN.1 que son frecuentemente utilizadas dentro de SNMP son BEGIN, CHOICE, DEFINED, DEFINITIONS, END, EXPORTS, IDENTIFIER, IMPORTS, INTEGER, NULL, OBJECT, OCTET, OF, SEQUENCE y STRING.

ASN.1 también proporciona un significado especial a los siguientes caracteres:

<u>Caracteres</u>	<u>Nombre</u>
-	Número con signo
--	Comentario
::=	Asignación (definida como)
	Alternativa (opciones de una lista)
{ }	Inicio y fin de una lista
[]	Inicio y fin de una etiqueta
()	Inicio y fin de una expresión subtipo
..	Indica un rango

La siguiente sección enfatiza algunos de esos caracteres especiales.

2.4 TIPOS Y OBJETOS DE ASN.1

Esta sección se enfoca en los objetos y tipos de datos ASN.1 utilizados dentro de la Estructura de Administración de Red de INTERNET. Donde sea posible, se proporcionarán ejemplos derivados de la SMI, las definiciones concisas MIB, y los documentos MIB-II.

2.4.1 DEFINICION DE OBJETOS EN LAS MIBS

Una MIB contiene los objetos a ser manejados. La macro OBJECT-TYPE define esos objetos en un formato estándar que es consistente a través de las MIBs públicas y privadas. Un ejemplo de definición ASN.1 para la MIB aparece como sigue:

```
tcpInSegs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= {tcp 10}
```

Esta notación ASN.1 define un objeto llamado *tcpInSegs* que contiene la información *Counter* (contador). El tipo *Counter* es un número no negativo que se incrementa monótonamente. Este objeto es de sólo-lectura y es *obligatorio* para todos los dispositivos de red manejados que soporten estos padres, *mib.tcp*. Cuando un protocolo de administración accesa este objeto, este utiliza el nombre {tcp 10}, el cual identifica el 10º objeto definido dentro del grupo tcp.

2.4.2 TIPOS PRIMITIVOS (SIMPLES)

Para mantener la simplicidad de SNMP, la SMI de INTERNET utiliza un subconjunto de los tipos de datos ASN.1. Estos son divididos en dos categorías, los tipos Primitivos y los tipos Constructor. Los tipos de datos Primitivos (Simples) incluyen INTEGER, OCTET STRING, OBJECT IDENTIFIER, y NULL. Los siguientes ejemplos vienen de la MIB-II.

INTEGER es un tipo primitivo con valores únicos que son todos los números positivos y negativos, incluyendo el cero. El tipo INTEGER tiene dos casos especiales. El primero es el *tipo entero enumerado*, en el cual los objetos tienen un número no-cero específico, tal como el 1, 2, o 3. El segundo, el *tipo entero cadena de bits*, es usado para cadenas cortas de bits como (0..127) y desplegar el valor en hexadecimal. Un ejemplo de INTEGER es:

```
ipDefaultTTL OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
```

“El valor por default insertado dentro del campo Time-To-Live del encabezado IP de los datagramas originados en esta entidad, donde un valor TTL no es reemplazado por la capa de transporte del protocolo”.

```
::= {ip 2}
```

OCTET STRING es un tipo primitivo cuyos valores distinguibles son una secuencia ordenada de cero, uno, o más octetos. SNMP utiliza tres casos especiales del tipo OCTET STRING: *DisplayString*, *OctetBitstring*, y el *PhysAddress*. En el tipo *DisplayString*, todos los octetos son caracteres ASCII imprimibles. *OctetBitstring* es usado para cadenas de bits que exceden 32 bits de longitud (TCP/IP frecuentemente incluye campos de 32 bits; esta cantidad es un valor típico para el ancho de palabra interno de varios procesos en servidores y enrutadores dentro de INTERNET). MIB define la *PhysAddress* y la utiliza para representar direcciones físicas). Un ejemplo del uso de *DisplayString* es:

sysContact OBJECT-TYPE**SYNTAX DisplayString (SIZE (0..255))****ACCESS read-write****STATUS mandatory****DESCRIPTION**

“La identificación textual de la persona contactada para este nodo administrado, y la información de como contactar a esta persona.”

::={system 4}

Observe que el subtipo indica que el tamaño permitido para *DisplayString* es entre 0 y 255 octetos.

OBJECT IDENTIFIER es un tipo cuyos valores distinguibles son el conjunto de todos los identificadores de objeto permitidos de acuerdo con las normas ISO 8824 (Norma ISO especificando ASN.1). El tipo *ObjectName*, es un caso especial que utiliza SNMP, está restringido a los identificadores de objeto de los objetos y subárboles dentro de las MIB, como por ejemplo:

ipRouteInfo OBJECT-TYPE**SYNTAX OBJECT IDENTIFIER****ACCESS read-only****STATUS mandatory****DESCRIPTION**

“Una referencia a las definiciones MIB específicas al protocolo de enrutamiento responsable para esa ruta, como determinado por el valor especificado en el valor *ipRouteProto* de la ruta. Si ésta información no está presente, su valor puede ser inicializado al OBJECT IDENTIFIER {0,0}, el cual es un objeto identificador sintácticamente válido, y cualquier implementación conforme con ASN.1 y BER debe ser capaz de generar y reconocer este valor”.

::={ipRouteEntry 13}

NULL es un tipo con un solo valor también llamado null. Que funciona como un inicializador, pero actualmente no es utilizado por los objetos SNMP. (se puede ver NULL utilizado como un inicializador en el campo de las variables enlazadas del PDU *GetRequest* de SNMP. NULL es asignado para ser el valor de una variable desconocida, esto es, el valor buscado del PDU *GetRequest*.

2.4.3 TIPOS CONSTRUCTOR (ESTRUCTURADOS)

Los tipos constructor, SEQUENCE y SEQUENCE OF, definen tablas y renglones dentro de esas tablas. Por convención, los nombres para tablas terminan con el sufijo "Table", y los nombres para los renglones terminan con el sufijo "Entry". A continuación se definen los tipos constructor.

SEQUENCE es un tipo constructor definido mediante una lista de tipos con referencia compuesta y ordenada. Algunos de los tipos pueden ser opcionales, y todos pueden ser diferentes a los tipos ASN.1. Cada valor del nuevo tipo consiste de una lista ordenada de valores, uno por cada tipo de componente. SEQUENCE como una totalidad, define un renglón dentro de una tabla. Cada entrada en SEQUENCE especifica una columna dentro del renglón.

SEQUENCE OF es un tipo constructor que está definido mediante un solo tipo existente; cada valor en el nuevo tipo es una lista ordenada de cero, uno o más valores de un tipo ya existente. Como SEQUENCE, SEQUENCE OF define los renglones en una tabla; a diferencia de SEQUENCE, SEQUENCE OF sólo utiliza elementos del mismo tipo ASN.1.

La tabla *TCP connection* que sigue ilustra a ambos, SEQUENCE y SEQUENCE OF.

```

TcpConnTable  OBJECT-TYPE
    SYNTAX      SEQUENCE OF TcpConnEntry
    ACCESS      not-accessible
    STATUS      mandatory
    DESCRIPTION
        "Una tabla conteniendo información específica de la
        conexión TCP"
    ::= { tcp 13 }
tcpConnEntry OBJECT-TYPE
    SYNTAX      TcpConnEntry
    ACCESS      not-accessible
    STATUS      mandatory
    DESCRIPTION
        "Información acerca de una conexión actual TCP. Un objeto
        de este tipo es transeúnte; éste termina cuando la conexión
        realiza la transición al estado CLOSED".
INDEX { tcpConnLocalAddress,

```

```

    tcpConnLocalPort,
    tcpConnRemAddress,
    tcpConnRemPort }
 ::= {tcpConnTable 1}
 TcpConnEntry ::=
 SEQUENCE {
   tcpConnState
     INTEGER,
   tcpConnLocalAddress
     IpAddress,
   tcpConnLocalPort
     INTEGER (0..65535)
   tcpConnRemAddress
     IpAddress,
   tcpConnRemPort
     INTEGER (0..65535)
 }

```

Este ejemplo amplía la gramática ASN.1. El nombre de la tabla, *tcpConnTable*, termina con el sufijo *Table*. El nombre del renglón, *tcpConnEntry*, termina con el sufijo *Entry*. El nombre de secuencia, *TcpConnEntry*, es el mismo que el nombre del renglón. Excepto que éste inicia con una letra mayúscula. La cláusula INDEX define la construcción y orden de las columnas que constituyen los renglones. Un excelente artículo de Dave Perkins' "Como Leer y Usar un MIB SNMP" [2-11] explora los objetos tablas y renglones con gran detalle.

2.4.4 TIPOS DEFINIDOS

La Estructura de Administración de Red INTERNET utiliza los tipos definidos. Los tipos definidos incluyen *NetworkAddress*, *IpAddress*, *Counter*, *Gauge*, *TimeTicks* y *Opaque*. El ejemplo siguiente viene de la RFC 1213.

NetworkAddress es un tipo que fue diseñado para representar la dirección de una de varias familias de protocolos. Un CHOICE es un tipo primitivo que proporciona alternativas entre otros tipos, y es encontrado en varias secciones de la definición SMI proporcionada más adelante. Sin embargo, actualmente solo una familia de protocolos, la familia INTERNET (*Internet IpAddress* en la definición SMI) ha sido definida para este CHOICE. Aquí hay un ejemplo.

```

atNetAddress OBJECT-TYPE
    SYNTAX NetworkAddress
    ACCESS read-write
    STATUS deprecated

```

DESCRIPTION

“La *NetworkAddress* (por ejemplo, la dirección IP) correspondiente a la dirección del medio ‘físico’ dependiente”.

::={atEntry 3}

Debido a que sólo soporta direcciones IP, el uso de este tipo es desalentador.

IpAddress es un tipo de amplia aplicación que representa una dirección Internet de 32 bits. Este tipo es representado como un OCTET STRING de longitud 4 (octetos) en orden de bytes de la red (el orden de los bytes que son transmitidos por la red).

TcpConnRemAddress OBJECT-TYPE

SYNTAX *IpAddress*
ACCESS read-only
STATUS mandatory
DESCRIPTION

“La dirección IP remota para esta conexión TCP”.

::={tcpConnEntry 4}

Counter es un tipo de amplia aplicación que representa a un entero no negativo que se incrementa monótonamente hasta que éste alcanza un máximo valor, entonces se reinicia, y se incrementa nuevamente desde cero. El máximo valor del contador es $2^{32}-1$, o 4,294,967,295 en decimal. En otras palabras, el *Counter* es un número sin signo de 32 bits. Un INTEGER es un valor con signo de 32 bits. Por convención, se escribe el nombre de un objeto COUNTER en plural; y éste termina en una letra minúscula *s*. Aquí hay un ejemplo :

icmpInDestUnreachs OBJECT-TYPE

SYNTAX *Counter*
ACCESS read-only
STATUS mandatory
DESCRIPTION

“El número de mensajes ICMP destino no recibidos”.

::={icmp 3}

Gauge es un tipo de amplia aplicación que representa un entero no negativo. Este puede incrementarse o decrementarse, pero limitado a un valor máximo. El máximo valor del contador es $2^{32}-1$. Aquí hay un ejemplo :

ifSpeed OBJECT-TYPE

SYNTAX *Gauge*
ACCESS read-only
STATUS mandatory
DESCRIPTION

“Una estimación del ancho de banda actual de la interface en bits por segundo. Para interfaces que no varían en ancho de banda, o para aquellos en donde no puede realizarse una estimación precisa, este objeto debe contener el ancho de banda nominal”.

::={ifEntry 5}

TimeTicks es un tipo de amplia aplicación que representa un entero no negativo que cuenta el tiempo en centésimas de segundo desde algún momento, o punto en el tiempo. Cuando la MIB define los tipos de objeto que utiliza este tipo ASN.1, la descripción del tipo de objeto identifica la época de referencia, como se observa en el siguiente ejemplo :

sysUpTime OBJECT-TYPE

SYNTAX TimeTicks

ACCESS read-only

STATUS mandatory

DESCRIPTION

“El tiempo (en centésimas de segundo) desde que la porción de manejo de red del sistema fue reinicializado por última vez”.

::={system 3}

Opaque es un tipo de amplia aplicación que permite el paso de sintaxis ASN.1 arbitrariamente. Las reglas básicas ASN.1 codifican un valor dentro de una cadena de octetos. Esta cadena, está codificada como un OCTET STRING, dando un efecto “doble-envoltura” sobre el valor original de ASN.1. SNMP actualmente no utiliza *Opaque*, aunque éste puede ser utilizado en algunas MIBs privadas.

2.4.5 TIPOS ETIQUETA (PROCESOS)

Las etiquetas(*tags*) se distinguen entre objetos definidos inequívocamente. Mientras un lector humano puede ser capaz de distinguir objetos definidos a través de sus nombres en notación ASN.1, una máquina no puede hacerlo sin información adicional. Por lo tanto, los tipos etiquetados(*tagged*) utilizan un tipo previamente definido como base, entonces agregan una información única. ASN.1 define cuatro clases de etiquetas: universal, aplicación, contexto específico, y privado. ASN.1 define etiquetas universales. Otras normas, como las normas INTERNET, asignan clases de aplicación de etiquetas. La definición SNMP interpreta clases etiquetas de contexto específico, de acuerdo a su contexto. Aplicaciones específicas de una empresa utilizan clases etiquetas privadas. Un número dentro de corchetes ([]) identifica tipos *tagged*. Por ejemplo, la definición SMI que se verá más adelante nos muestra que :

TimeTicks ::=
[APPLICATION 3]
IMPLICIT INTEGER (0..4294967295)

Por lo tanto, el tipo *TimeTicks* es un tipo *tagged*, designado a APPLICATION 3. Este es de la clase aplicación, y el número etiqueta es 3, Además puede tomar cualquier valor entre 0 y 4294967295. La palabra clave IMPLICIT indica que la etiqueta asociada con el tipo INTEGER no es transmitida, pero la etiqueta asociada con *TimeTicks* sí. Esto reduce la cantidad de datos que deben ser codificados y transmitidos.

2.5 REGLAS DE CODIFICACION

Anteriormente se vio la sintaxis abstracta que representa la información administrada. Ahora se verán las reglas de codificación que permiten que la información sea transmitida en una red. Las Reglas Básicas de Codificación (*BER*) definen esta sintaxis de transferencia, y la ISO 8825 la específica [2-3].

2.5.1 CODIFICANDO INFORMACION ADMINISTRADA.

Recordando que cada máquina en el sistema de administración puede tener su propia representación interna de la información administrada, la sintaxis ASN.1 describe dicha información en una manera standard. La sintaxis de transferencia ejecuta la comunicación a nivel de bits (la representación externa) entre máquinas. Por ejemplo, asumiendo que un servidor necesita información administrada de otro dispositivo, la aplicación de administración generaría un requerimiento SNMP, en el cual las BER codificarían tal requerimiento y se transmitiría en la red. La máquina destino recibiría la información de la red, la decodificaría utilizando las reglas BER, y lo interpreta como un comando SNMP. La respuesta SNMP la regresaría de una manera similar. La estructura de codificación utilizada por la representación externa es llamada Codificación Tipo-Longitud-Valor (*Type-Length-Value encoding*). Vea la figura 2-1.

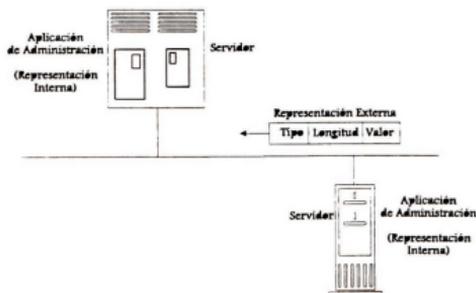


Figura 2-1. Representación de datos interna y externa

2.5.2 CODIFICACION TIPO-LONGITUD-VALOR

Para definir la representación externa de datos, las BER primero especifican la posición de cada bit dentro de los octetos a ser transmitidos. De cada octeto primero se transmite el bit más significativo (MSB), y este se define como el bit 8 en el lado izquierdo del octeto. En el octeto se define al bit 1 como el bit menos significativo (LSB) en el lado derecho del octeto. Vea la figura 2-2.



Figura 2-2. Ordenación de bits en base a las BER, como se define en la ISO 8825

La estructura de codificación de datos tiene tres componentes: Tipo, Longitud y Valor (TLV). Note que en la literatura se pueden encontrar otros nombres para Tipo-Longitud-Valor, incluyendo Tag-Longitud-Valor e Identificador-Longitud-Contenido (ISO 8825). La estructura de una codificación TLV utilizada con SNMP es mostrada en la figura 2-3.

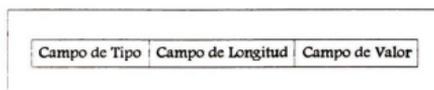


Figura 2-3. Codificación de Tipo-Longitud-Valor (TLV).

Definiendo el orden y la estructura de los bits, las BER garantizan que ambas terminales del canal de comunicación interpretan el flujo de bits consistentemente. A continuación se examina la estructura de cada campo TLV individualmente.

CAMPO TIPO

El campo Tipo es el primero de la codificación e indica el destino para la estructura que lo sigue. Así, el campo Tipo contiene una identificación para la estructura codificada; éste codifica la etiqueta ASN.1 (la clase y el número) para el tipo de dato contenido en el campo Valor. Un subcampo dentro del campo Tipo contiene un bit designado como P/C

que indica donde el código es Primitivo ($P/C=0$) o Constructor ($P/C=1$), como se muestra en la figura 2-4.

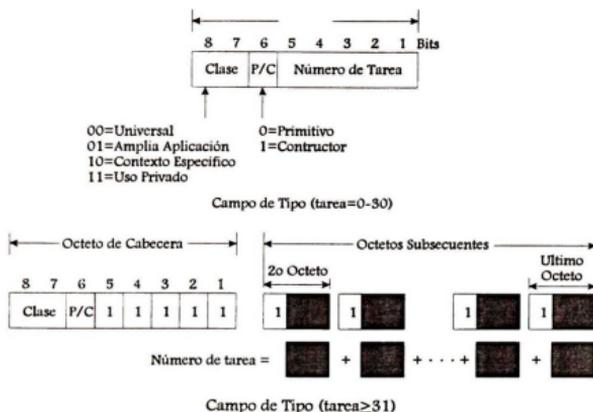


Figura 2-4. Codificación del campo Tipo

Existen dos tipos de campos Tipo, y su uso depende de la magnitud del número de etiqueta. Cuando el número de etiqueta es entre 0 y 30, el campo Etiqueta contiene un solo octeto (vea figura 2-4). Cuando el número de etiqueta es 31 o mayor, el campo Etiqueta contiene múltiples octetos. En este caso, el primer octeto contiene tres subcampos: la Clase, el bit P/C, y el número de etiqueta. El subcampo Clase codifica la clase de etiqueta en uso:

<u>Clase</u>	<u>Bit 8</u>	<u>Bit 7</u>
Universal	0	0
Aplicación	0	1
Contexto específico	1	0
Privada	1	1

Las aplicaciones SNMP utilizan las primeras tres clases: universal, aplicación y contexto específico. La clase universal codifica los tipos INTEGER, OCTET STRING, entre otros. La clase aplicación codifica los tipos definidos *IpAddress*, *Counter*, y otros. La clase contexto específico codifica los cinco PDUs de SNMP, *GetRequest*, *GetResponse*, y otros.

El subcampo P/C indica la forma del elemento de dato. Una codificación Primitiva ($P/C=0$) significa que el contenido del octeto representa el valor directo. Una codificación Constructor ($P/C=1$) significa que el contenido del octeto codifica uno o más valores de datos adicionales, tal como un SEQUENCE. SNMP utiliza los números de etiqueta entre 0 y 30. El número de etiqueta aparece en el tercer subcampo y es representado en binario. El bit 5 es el MSB de la etiqueta; el bit 1 es su LSB.

La ISO 8824 contiene números de etiqueta para la clase universal (por ejemplo, UNIVERSAL 2 representa el tipo INTEGER). La especificación de SMI, la RFC 1155, contiene números de etiqueta para la clase aplicación (por ejemplo, *IpAddress* es un tipo primitivo con etiqueta [0]). La especificación de SNMP, la RFC 1157, contiene números de etiqueta para la clase contexto específico (por ejemplo, *GetRequest* PDU es un tipo constructor con etiqueta [0]). La siguiente lista resume las tres clases de campos Tipo utilizadas con SNMP y las codificaciones para éstos: clase, P/C, y número de etiqueta. Dichas codificaciones aparecen en notación binaria y hexadecimal.

Clase Universal	Valor del Campo Tipo
INTEGER	00000010 = 02H
OCTET STRING	00000100 = 04H
NULL	00000101 = 05H
OBJECT IDENTIFIER	00000110 = 06H
SEQUENCE	00110000 = 30H
SEQUENCE-OF	00110000 = 30H

Clase Aplicación	Valor del Campo Tipo
IpAddress	01000000 = 40H
Counter	01000001 = 41H
Gauge	01000010 = 42H
TimeTicks	01000011 = 43H
Opaque	01000100 = 44H

Clase Contexto Específico	Valor del Campo Tipo
GetRequest	10100000 = A0H
GetNextRequest	10100001 = A1H
GetResponse	10100010 = A2H
SetRequest	10100011 = A3H
Trap	10100100 = A4H

Aunque las BER también proveen números de etiqueta de valor 31 o mayor, SNMP no los utiliza. SNMP utiliza un formato diferente para los números de etiqueta que son mayores de 31, en el campo Tipo. El número de etiqueta en el primer octeto se cambia al binario 11111, y los octetos subsecuentes son sumados al acarreo del número de etiqueta. Si el bit 8 de un octeto es 1, esto indica que lo seguirán más octetos; si el bit 8 de un octeto es 0, especifica el último octeto. Los bits del 7 al 1 de cada octeto subsecuente acarrean el entero binario sin signo del número de etiqueta. El bit 7 del primer octeto subsecuente indica el MSB del número de etiqueta.

CAMPO DE LONGITUD

El campo de Longitud sigue al campo de Tipo y determina el número de octetos que contendrá el campo de Valor. El campo de Longitud puede tomar la forma definitiva corta o la forma definitiva larga, como se muestra en la figura 2-5. Por “definitiva” indicamos que la longitud de la codificación es conocida anteriormente a la transmisión, “no definitiva” indica que la longitud es desconocida.

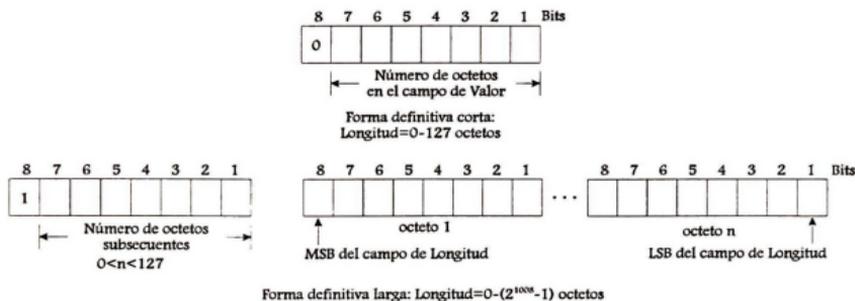


Figura 2-5. Codificación del campo de Longitud

La forma definitiva corta indica una longitud entre 0 y 127 octetos en el campo de contenido, la forma definitiva larga indica 128 o más octetos en el campo de contenido, aunque este puede indicar longitudes más cortas. La forma larga utiliza múltiples octetos para representar la longitud total. En la forma larga, el primer octeto del campo de longitud tiene en el bit 8 un 1, seguido por un número binario indicando el número de octetos a seguir. Este número debe ser entre 1 y 126, el 127 está reservado para extensiones futuras. El bit 8 del segundo octeto es considerado el MSB del campo de Longitud, y los siguientes octetos constituyen el resto de la longitud. Así, la forma definitiva larga puede representar una longitud de hasta $2^{1008}-1$ octetos.

CAMPO DE VALOR

El campo de Valor contiene cero o más octetos de contenido, los cuales transmiten los valores de los datos. Los ejemplos que se verán a continuación, incluyen un entero, un carácter ASCII, o un OBJECT IDENTIFIER.

2.5.3 EJEMPLOS DE CODIFICACION

Anteriormente se mencionó que la SMI de INTERNET define un subconjunto de los tipos ASN.1. Este subconjunto incluye los tipos primitivos y universales siguientes: INTEGER, OCTET STRING, OBJECT IDENTIFIER, y NULL. Los tipos constructor universales son SEQUENCE y SEQUENCE OF. Los tipos primitivos de aplicación son *IpAddress*, *Counter*, *Gauge*, y *TimeTicks*. Las aplicaciones relacionadas a SNMP solo utilizan esos diez tipos.

CODIFICACION DEL TIPO INTEGER

El tipo INTEGER es un tipo simple que tiene números enteros con valores de cero, positivos o negativos. Este es un tipo primitivo codificado con un campo de valor que contiene uno o más octetos de contenidos. Los octetos de contenidos son un número de complemento a dos binarios, igual al valor del entero, y puede utilizar tantos octetos como sea necesario. Por ejemplo, un león, pesa 75 kilos, el valor de su peso podría codificarse como: campo de Tipo = 02H, Campo de Longitud = 01H, y campo de Valor = 4BH (vea figura 2-6). Note que el valor aparece en comillas (Value = "75") para indicar que se representa una cantidad la cual puede ser numérica, caracteres ASCII, una dirección IP, etc.

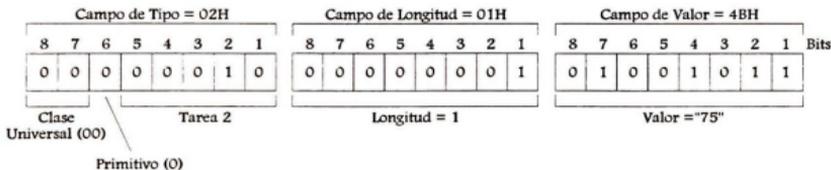


Figura 2-6. Codificación para el tipo INTEGER, Valor="75"

CODIFICACION DEL TIPO OCTET STRING

OCTET STRING es un tipo simple cuyos valores distinguidos son una secuencia ordenada de cero, uno o más octetos, cada uno de los cuales debe ser un múltiplo de 8 bits. La codificación para los valores OCTET STRING es primitiva, con el campo de Tipo = 04H. El campo de Longitud y el campo de Valor dependen de la información codificada. Un ejemplo para mostrar la codificación del tipo OCTET STRING, se observa en la figura 2-7, la cual muestra cómo se codifica el valor para los caracteres BBM. El campo de Tipo contiene 04H, indicando un tipo primitivo, OCTET STRING (número de etiqueta 4). El campo de longitud indica 3 octetos en el campo de Valor. La codificación del campo de valor viene de la tabla de caracteres ASCII.



Figura 2-7. Codificación del tipo OCTET STRING, Valor="BBM"

CODIFICACION DEL TIPO OBJECT IDENTIFIER (OID)

Los OBJECT IDENTIFIER en SNMP, identifican objetos administrados. Su campo valor contiene una lista ordenada de subidentificadores. Para ahorrar esfuerzo en la codificación y en la transmisión, se puede tomar ventaja del hecho de que el primer subidentificador es un número pequeño, 0, 1 o 2, y combine éste matemáticamente con el segundo subidentificador, el cual puede ser más grande. Por lo tanto, el número total de subidentificadores es menor que el número de componentes objetos identificadores en el valor OID que es codificado. Este número reducido (uno menos) resulta de una expresión matemática que utiliza los primeros dos componentes OID para producir otra expresión:

Dado que X es el valor del primer OID, y Y el segundo:
Primer subidentificador = $(X * 40) + Y$

Los valores para esos subidentificadores son codificados y colocados dentro del campo Valor. El bit 8 de cada octeto indica si ese octeto es o no el último en la serie de octetos requerida para describir completamente el valor. Si el bit 8 es igual a 1, por lo menos sigue un octeto; si el bit 8 es igual a 0, indica el último (o único) octeto. Los bits 7 al 1 de cada octeto codifican subidentificadores. Usando un ejemplo del árbol de objetos MIB, el grupo *System*, asume que un OBJECT IDENTIFIER tiene un valor de

{iso org(3) dod(6) internet(1) mgmt(2) mib(1) 1}

Del árbol objeto, éste es representado por

{1 3 6 1 2 1 1}

Utilizando los valores de X=1 y Y=3, y la expresión de arriba para el primer valor del subidentificador se tiene :

$$(1 * 40) + 3 = 43$$

Esto nos da como resultado un valor para el primer subidentificador de 43, el valor para el segundo subidentificador de 6, el valor para el tercer subidentificador de 1, etc. El primer valor (43) necesita 6 bits, o un octeto, para codificarse (00101011). El segundo valor (6) necesita 3 bits para codificarse (110), y requiere solo un octeto. Los valores subsecuentes

también requieren un octeto. Como se puede ver en la figura 2-8, la codificación es: campo de Tipo = 06H (OBJECT IDENTIFIER, etiqueta = 6); campo de Longitud = 06H y campo de Valor = 2B 06 01 02 01 01 H.

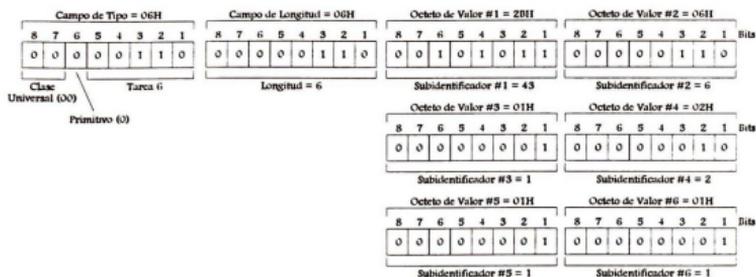


Figura 2-8. Codificación para el tipo OBJECT IDENTIFIER, Valor = {1 3 6 1 2 1 1}

CODIFICACION DEL TIPO NULL

El tipo NULL es un almacenador temporal que comunica la ausencia de información. Por ejemplo, cuando un administrador solicita el valor de una variable, éste utiliza el tipo NULL como un almacenador temporal en la posición donde el agente llenará la respuesta. La codificación para el tipo NULL es un primitivo. El campo de Tipo = 05H, y el campo de Longitud = 00H. El campo de Valor está vacío (no tiene valor en los octetos), como se muestra en la figura 2-9.

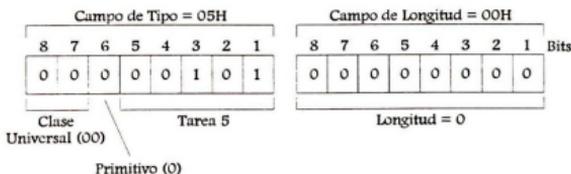


Figura 2-9. Codificación para el tipo NULL, Valor = NULL

CODIFICACION DEL TIPO SEQUENCE

Como se vio anteriormente, el tipo SEQUENCE es una lista de tipos ASN.1. Un valor SEQUENCE siempre está codificado en una forma construida. Las variables enlazadas utilizadas (*VarBind*) dentro de los mensajes SNMP proporcionan un buen ejemplo de SEQUENCE. Las variables enlazadas unen el nombre de un objeto con su

valor, el cual es transmitido dentro del campo de Valor. Como se muestra a continuación, SNMP define *VarBind* como:

```

VarBind ::=
  SEQUENCE {
    name
      ObjectName
    value
      ObjectSyntax
  }
  
```

```

VarBindList ::=
  SEQUENCE OF
    VarBind
  
```

Como muestra esta sintaxis, *VarBind* es un SEQUENCE de pares ordenados constituidos por un nombre y un valor, y *VarBindList* es una lista de nombres y valores.

Veamos un ejemplo más específico de un tipo SEQUENCE. Supongamos que se necesita la descripción del sistema para un objeto administrado en particular cuyo nombre es *sysDescr*. Para obtener la descripción del sistema, el Administrador transmite un *GetRequest* SNMP hacia el Agente que pregunta por el valor del objeto *sysDescr*. El Agente responde con un mensaje *GetResponse* SNMP, el cual contiene el valor, como por ejemplo "Local Ethernet Bridge Modelo 2265M". *VarBind* asocia el objeto *sysDescr* con su valor, como se muestra en la figura 2-10.

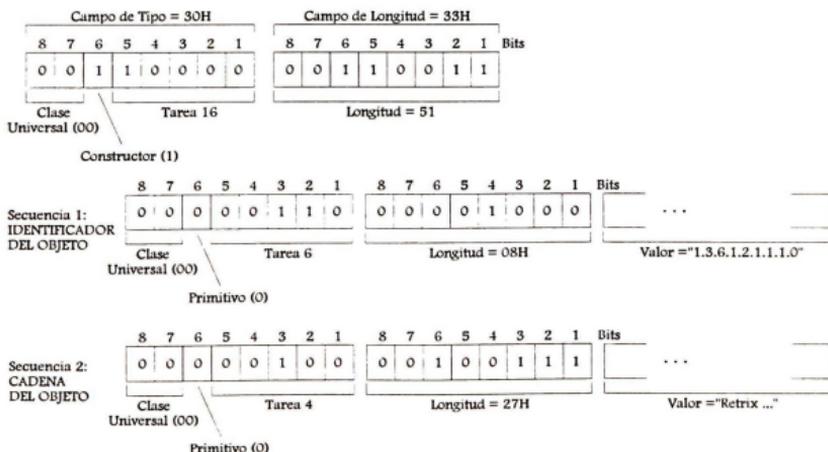


Figura 2-10. Codificación del tipo SEQUENCE, una Variable Enlazada (VarBind)

El primer campo de Tipo (30H) indica un tipo construido, con Etiqueta = 16 (SEQUENCE). El primer campo de Longitud contiene 33H, indicando que sigue el octeto de Valor 51. Entonces son aplicadas las BER para cada tipo en la SEQUENCE. La primera secuencia identifica un tipo primitivo con Etiqueta = 6 (OBJECT IDENTIFIER) y Longitud 08H. El campo de valor contiene la representación numérica del objeto *sysDescr* {1.3.6.1.2.1.1.1.0}. La segunda secuencia identifica un tipo primitivo con Etiqueta = 4 (OCTET STRING), y Longitud 27H. El segundo campo de Valor representa el valor del objeto *sysDescr* ("Local..."). Observe la longitud total de la codificación. La Secuencia #1 contiene 10 octetos (1 del tipo de campo + 1 de la longitud + 8 del valor). La Secuencia #2 contiene 41 octetos (1 + 1 + 39). La Secuencia #1 mas la Secuencia #2 (10 + 41) es igual al valor del primer campo de Longitud (51 octetos).

CODIFICACION DEL TIPO SEQUENCE OF

El valor del tipo SEQUENCE OF está codificado en forma construida de la misma manera que el tipo SEQUENCE.

CODIFICACION DEL TIPO IPAddress

Los ejemplos de codificación ahora se mueven a la clase de aplicación. Desde que hay todas las codificaciones de las aplicaciones *Class* (01), *primitive* (P/C=0), con números de etiqueta entre 0 y 4, los campos Tipo oscilarán de 40 a 44H (vea figuras 2-11 a la 2-14).

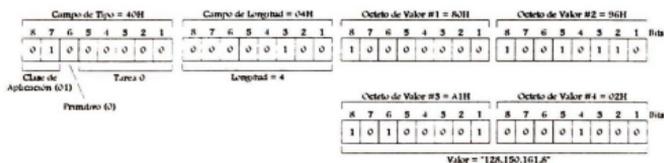


Figura 2-11. Codificación para el tipo *IpAddress*, Valor="128.150.161.6"

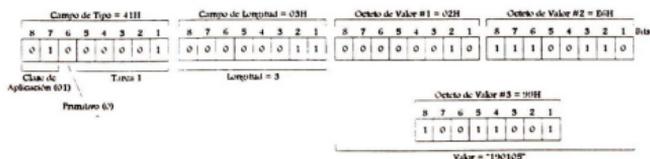
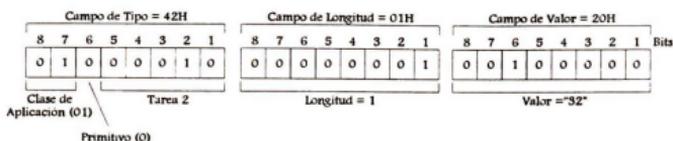
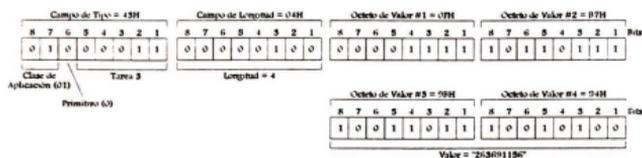


Figura 2-12. Codificación para el tipo *Counter*, Valor="190105"

Figura 2-13. Codificación para el tipo *Gauge*, Valor="32"Figura 2-14. Codificación del tipo *TimeTicks*, Valor="263691156"

La SMI define el tipo *IpAddress*. Éste contiene una dirección de 32 bits, la cual se representa en cuatro octetos. Para codificar el tipo *IpAdEntAddr* (ver figura 2-11), el campo de Tipo se pone a 40H (clase aplicación, Primitiva, Etiqueta=0) y el campo de Longitud a 4, representando los cuatro octetos de una dirección IP. El campo de Valor contiene cuatro octetos de contenido, los cuales transmiten la dirección IP en notación decimal punteada. Para la dirección mostrada en el ejemplo (128.150.161.8), el primer octeto en el campo de Valor contiene el equivalente binario de 128 (10000000), el segundo el binario equivalente de 150, y así sucesivamente.

CODIFICACION DEL TIPO COUNTER

Un tipo *Counter* representa un entero no negativo que se incrementa monótonamente a un máximo de 4,294,967,295, y entonces se vuelve a cero. El grupo ICMP de la MIB como se verá en otro capítulo, utiliza varios contadores para grabar estadísticas de mensajes. Como ejemplo, un objeto *icmpInMsgs* graba el número de mensajes que el grupo ICMP procesa en un enrutador o un servidor. Una codificación simple (ver figura 2-12), podría tener un campo de Tipo=41H, representando la clase aplicación, codificación primitiva, y Etiqueta=1. El Valor (190,105) requiere tres octetos. El campo de Longitud es por lo tanto 03H, y el campo de Valor contiene 02 E6 99H, representando los mensajes 190,105.

CODIFICACION DEL TIPO GAUGE

Un tipo *Gauge* es un entero no negativo que puede incrementarse o decrementarse, pero limitado a un valor máximo de 4,294,967,295. El tipo *Gauge* no es usado frecuentemente. La MIB lo define solamente para los objetos *ifSpeed*, *ifOutQLen*, y *topCurrEstab*. Por ejemplo, la figura 2-13 asume que la máxima longitud de la cola de salida de una interface particular es de 32 paquetes. Para codificar en el tipo *Gauge* este valor, el campo de Tipo se cambia a 42H (clase de aplicación, Primitivo, Etiqueta=2). Un octeto codifica el decimal 32, por lo tanto, el campo de Longitud=01H y el campo de valor contiene 20H, el valor deseado del decimal 32.

CODIFICACION DEL TIPO TIME TICKS

El tipo *TimeTicks* contiene una marca de tiempo que mide el tiempo transcurrido (en centésimas de segundo) desde algún evento. El objeto *sysUpTime* mide el tiempo desde que fue reinicializada la entidad de administración de la red en un dispositivo. Si el valor *sysUpTime* para un dispositivo en particular fue 263,691,156 centésimas de segundo (cerca de 30 días), este valor sería codificado como se muestra en la figura 2-14. El campo de Tipo se colocaría en 43H (clase de aplicación, Primitivo, Etiqueta=3). Cuatro octetos representan un Valor igual a 263691156. Por lo tanto, el campo de Longitud contiene 04H. Los cuatro octetos en el campo de Valor contienen la representación binaria del valor *TimeTicks*.

CODIFICACION DEL CONTEXTO ESPECIFICO PARA SNMP

La clase final de codificación discutida en este capítulo es la codificación de contexto específico, el cual es utilizado dentro del contexto de SNMP. Cinco unidades de datos de protocolo (PDUs), los cuales se discuten con detalle mas adelante, transmiten información de SNMP. Los PDUs son *GetRequest*, *GetNextRequest*, *GetResponse*, *SetRequest*, y *Trap*. Estos PDU tienen números de etiqueta del 0 al 4, respectivamente. Estas codificaciones son de clase contexto específico (10) y Constructor (P/C=1). Los campos de Tipo tienen así valores en el rango de A0 hasta A4H (ver figura 2-15). Los campos de Longitud y Valor dependen de la información transmitida.

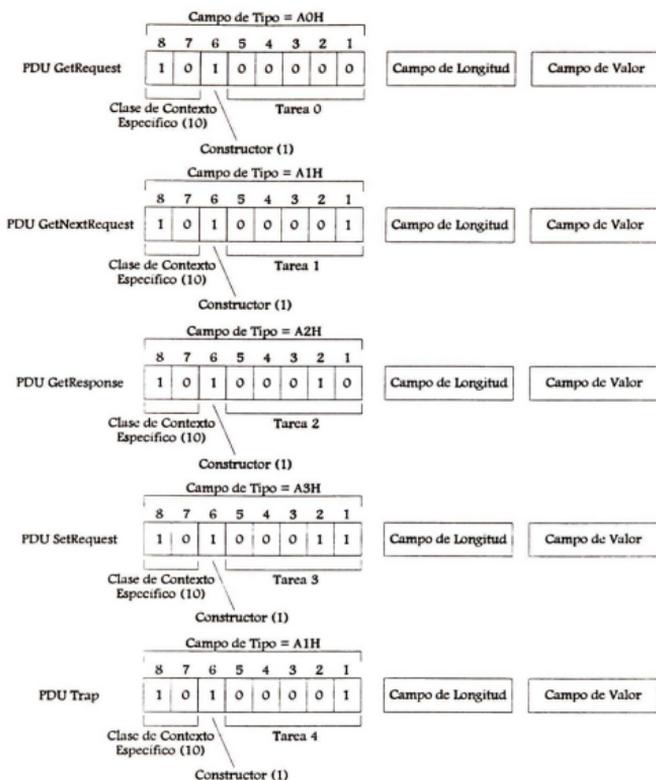


Figura 2-15. Codificación de los tipos de contexto específico utilizados con SNMP

2.6 NOMBRES DE LOS OBJETOS

Cada objeto, sea éste un dispositivo o la característica de un dispositivo, debe tener un nombre por el cual sólo éste pueda ser identificado. Ese nombre es el *identificador de objeto* (OID). Se escribe como una secuencia de enteros, separada por puntos. Por ejemplo, la secuencia {1.3.6.1.2.1.1.1.0} especifica la descripción del sistema, dentro del grupo System del subárbol mgmt que se verá más adelante.

Los anexos B, C y D del estándar ISO 8824, definen las secuencias numéricas, y éstas aparecen como un árbol con una raíz y varios corchetes directamente adjuntos, refiriéndose a ellos como *niños*. Vea la figura 2-16, esos corchetes conectan a otros

corchetes. Se puede utilizar la estructura de la raíz, corchetes, subcorchetes, y dejar para diagrama a todos los objetos dentro de una MIB particular y sus relaciones.

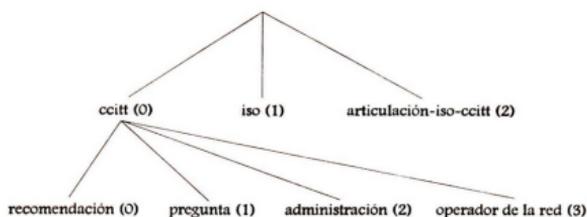


Figura 2-16. El nodo raíz y los valores de los componentes OBJECT IDENTIFIER asignados por la CCITT.

La raíz no necesita una designación, pero un valor numérico específico designa los tres corchetes conectados. La ISO administra el corchete etiquetado como 1, la CCITT administra el corchete etiquetado como 0, y la ISO y la CCITT administran juntas el corchete 2. El corchete de la CCITT tiene cuatro hijos: recomendación (0), identifica las recomendaciones de la CCITT; pregunta (1) es usado por CCITT para grupos de estudio; administración (2) identifica los valores de los Códigos de Datos por País (DDCs); y operador de la red (3) identifica los valores de los Códigos de Identificación de Datos en la Red (DNICs).

El corchete de la ISO (figura 2-17) también tiene cuatro hijos: standard (0) designa estándares internacionales; autoridad de registro (1) está reservado para un apéndice para la ISO 8824; cuerpo de miembro (2) es un código numérico de país de tres dígitos que asigna la ISO 3166 a cada miembro de ISO/IEC; y organizaciones identificadas (3) tienen valores de un designador de código internacional (ICD), definido en la ISO 6523. El Departamento de Defensa de E.U. está asignado a uno de los hijos bajo 1.3, y está designado como 6. Bajo este árbol, la comunidad INTERNET tiene la designación de 1.

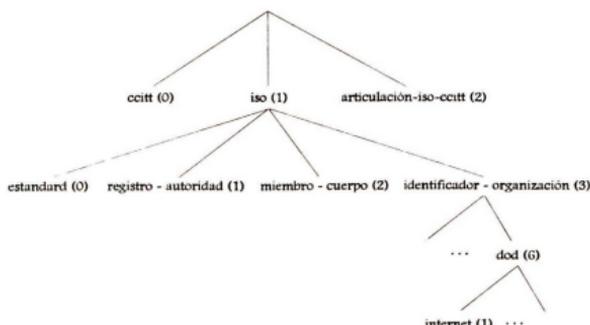


Figura 2-17. El nodo raíz y los valores del componente OBJECT IDENTIFIER asignados por la ISO

Para identificar una posición particular en el árbol, deben listarse los valores numéricos en una cadena, separados por puntos. Por ejemplo, para identificar la posición del subárbol INTERNET, se debe iniciar en la raíz y moverse hacia abajo hasta que alcance la posición. Al nivel INTERNET (figura 2-18), se comienzan a ver detalles para la administración de red y SNMP. El subárbol INTERNET tiene cuatro corchetes:

- El subárbol de directorio(1), {*internet 1*} o {1.3.6.1.1} reservado para usos futuros por el directorio OSI dentro del INTERNET.
- El subárbol mgmt(2), {*internet 2*}, manejado por la Autoridad que Asigna Números INTERNET, e incluye MIBs estándar.
- El subárbol experimental(3), {*internet 3*}, usado para experimentos en Internet.
- El subárbol privado(4), {*internet 4*}, registro de objetos por los fabricantes.

La autoridad que asigna números INTERNET (IANA) administra esos subárboles y los publica en el documento de números asignados (actualmente la RFC 1340).

Más adelante se ve la MIB en detalle. En este capítulo se observa la estructura de los árboles aplicable a la estructura de la administración estándar de red INTERNET. la MIB Estándar INTERNET es definida por {*mgmt 1*}. Bajo este árbol existen objetos definidos por la MIB (RFC 1213), que se verá en el siguiente capítulo.

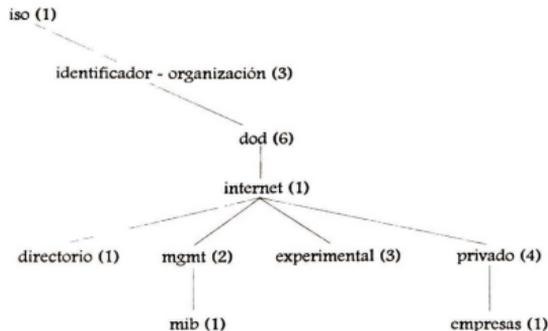


Figura 2-18. Valores del componente OBJECT IDENTIFIER asignados por Internet.

Regresaremos ahora al ejemplo dado al inicio. Ahora que conocemos las identidades de las estructuras del árbol individual, podemos construir la siguiente secuencia:

```
internet OBJECT IDENTIFIER ::= {iso org(3) dod(6) 1}
mgmt OBJECT IDENTIFIER ::= {internet 2}
mib OBJECT IDENTIFIER ::= {mgmt 1}
system OBJECT IDENTIFIER ::= {mib 1}
sysDescr OBJECT IDENTIFIER ::= {system 1}
```

Cuando las estructuras de árbol de arriba son combinadas, el resultado nos da:

```
sysDescr OBJECT IDENTIFIER ::= {1.3.6.1.2.1.1.1}
```

Para el OID se necesita agregar un último elemento - un sufijo que identifica si una variable en particular ocurre sólo una vez (un escalár), o si la variable ocurre múltiples veces (como en entradas columnares).

Puesto que *sysDescr* es un escalár, no columnar, solo hay una instancia de esta variable. Por lo tanto, un .0 es agregado al final del OID:

```
{1.3.6.1.2.1.1.1.0}
```

Si el objeto fuera una entrada columnar, un índice mas un sufijo no-cero (.1, .2, una dirección IP, y otros) identificarían al objeto dentro de la tabla.

Los códigos experimentales, con prefijo {1.3.6.1.3}, han sido asignados para muchos objetos LAN y WAN y MIBs, tales como servicio de red sin conexión (ISO CLNS), objetos de Protocolo Punto a Punto (PPP), y la Red Óptica Sincrónica (SONET). Como los grupos de trabajo de la Fuerza de Trabajo de Ingeniería Internet (IETF) desarrollan MIBs, ellos las definen bajo un corchete en el árbol experimental. Una vez que esas MIBs son publicadas y colocadas en las normas estándar, ellos las mueven a un corchete bajo el subárbol INTERNET.

Las MIBs de fabricantes específicos utilizan códigos de empresa privada con el prefijo {1.3.6.1.4.1}. Esta es un área de rápido crecimiento, donde numeroso fabricantes desarrollan sus estructuras para soportar sus dispositivos de Interconexión en red, servidores, entre otros. Algunos ejemplos incluyen *3Com Corporation* con código {1.3.6.1.4.1.43}; *FTP Software Inc.* Con código {1.3.6.1.4.1.121}; y *US West Advanced Technologies* con código {1.3.6.1.4.1.312}.

2.7 CONCISA DEFINICION DE SMI

Quizás la mejor manera para resumir este capítulo es incluir un módulo titulado RFC1155-SMI de la RFC 1155, mostrado a continuación. Este módulo define todas las construcciones discutidas en este capítulo y las coloca en una forma útil para futuras referencias. Note que una revisión en la RFC 1212 [2-9] hace a la macro OBJECT-TYPE originalmente dada en la RFC 1155 obsoleta. La definición SMI incluye una nueva macro

llamada OBJECT-TYPE. Las líneas de comentario encerradas dentro de los signos <...>, indican el inicio y fin de la sección revisada de la RFC 1212.

Concisa Definición de SMI

```

RFC1155-SMI DEFINITIONS ::= BEGIN
  EXPORTS – EVERYTHING
  internet, private, enterprises,
  OBJECT-TYPE, ObjectName, ObjectSyntax, SimpleSyntax,
  ApplicationSyntax, NetworkAddress, IpAddress,
  Counter, Gauge, TimeTicks, Opaque;
  -- the path to the root (from RFC 1155)
  internet OBJECT IDENTIFIER ::= {iso org(3) dod(6) 1}
  directory OBJECT IDENTIFIER ::= {internet 1}
  mgmt OBJECT IDENTIFIER ::= {internet 2}
  experimental OBJECT IDENTIFIER ::= {internet 3}
  private OBJECT IDENTIFIER ::= {internet 4}
  enterprises OBJECT IDENTIFIER ::= {private 1}

  <definition of object types (taken from RFC 1212)>

  OBJECT-TYPE MACRO ::=
  BEGIN
    TYPE NOTATION ::=
      -- must conform to
      -- RFC1155's ObjectSyntax
      "SYNTAX" type(ObjectSyntax)
      "ACCESS" Access
      "STATUS" Status
      DescrPart
      ReferPart
      IndexPart
      DefValPart
    VALUE NOTATION ::= value (VALUE ObjectName)
    Access ::= "read-only"
      | "read-write"
      | "write-only"
      | "not accessible"
    Status ::= "mandatory"
      | "optional"
      | "obsolete"
      | "deprecated"
    DescrPart ::=
      "DESCRIPTION" value (description DisplayString)
      | empty
    ReferPart ::=

```

```

    "REFERENCE" value (reference DisplayString)
    |empty
IndexPart ::=
    "INDEX" "{" IndexTypes "}"
    |empty
IndexTypes ::=
    IndexType | IndexTypes "," IndexType
IndexType ::=
    -- if indexobject, use the SYNTAX
    -- value of the correspondent
    -- OBJECT-TYPE invocation
    value (indexobject ObjectName)
    -- otherwise use named SMI type
    -- must conform to IndexSyntax below
    |type (indextype)
DefValPart ::=
    "DEFVAL" "{" value (defvalue ObjectSyntax) "}"
    |empty
END
IndexSyntax ::=
CHOICE {
    number
        INTEGER (0..MAX),
    string
        OCTET STRING,
    object
        OBJECT IDENTIFIER,
    address
        NetworkAddress,
    ipAddress
        ipAddress
}
<names of the objects in the MIB (taken from RFC 1155)>
ObjectName ::= OBJECT IDENTIFIER
-- syntax of objects in the MIB
ObjectSyntax ::=
    CHOICE {
        simple
            SimpleSyntax,
        -- note that simple SEQUENCES are not directly mentioned here
        -- to keep things simple (i.e., prevent mis-use). However,
        -- application-wide types which are IMPLICITLY encoded simple
        -- SEQUENCES may appear in the following CHOICE
        application-wide
            ApplicationSyntax
    }
SimpleSyntax ::=

```

```

CHOICE {
    number
        INTEGER,
    string
        OCTET STRING,
    object
        OBJECT IDENTIFIER,
    empty
        NULL
}
ApplicationSyntax ::=
CHOICE {
    address
        NetworkAddress,
    counter
        Counter,
    gauge
        Gauge,
    ticks
        TimeTicks,
    arbitrary
        Opaque,
-- other application-wide types, as they are
-- defined, will be added here
}
-- application-wide types
NetworkAddress ::=
CHOICE {
    internet
        IPAddress
}
IPAddress ::=
[APPLICATION 0] -- in network-byte order
IMPLICIT OCTET STRING (SIZE (4))
COUNTER ::=
[APPLICATION 1]
IMPLICIT INTEGER (0..4294967295)
Gauge ::=
[APPLICATION 2]
IMPLICIT INTEGER (0..4294967295)
TimeTicks ::=
[APPLICATION 3]
IMPLICIT INTEGER (0..4294967295)
Opaque ::=
[APPLICATION 4] -- arbitrary ASN.1 value
IMPLICIT OCTET STRING -- "double-wrapped"
END

```

CAPITULO 3

BASE DE DATOS DE LA INFORMACION ADMINISTRADA

En el capítulo 1 se habló sobre la importancia de las bases de datos de información administrada de red, a menudo referida como la Base de Información Administrada (MIB) o librería de información de Administración. En éste capítulo se examina la MIB con detalle.

3.1 MIB DE INTERNET.

La SMI de INTERNET describe el esquema de identificación y la estructura para los objetos administrados en una red. La SMI trata principalmente con temas administrativos y organizacionales. Esto deja las tareas de definición de objetos, a las peticiones para comentarios (RFCs) de la administración de red. La SMI describe los nombres usados para identificar los objetos administrados, y utiliza la convención de nombramiento introducida anteriormente.

3.2 JERARQUIA DE NOMBRAMIENTO DE INTERNET.

Los objetos en una red tienen muchas características comunes a través de las subredes, los productos de fabricantes y los componentes individuales. Esto puede ser un gran problema para cada organización, ya que gasta preciosos recursos y tiempo codificando la ASN.1 para describir estos recursos. Por lo tanto la MIB de INTERNET proporciona un esquema de registro donde los objetos pueden ser definidos y categorizados dentro de un registro jerárquico. Este concepto puede ser encontrado en el acuerdo de convención ISO/CCITT, el cuál es utilizado por el estándar de INTERNET para identificar los objetos dentro de un sistema.

La figura 3-1 muestra el árbol de registro de INTERNET para la MIB. En el nivel raíz, tres ramas identifican la jerarquía de registro como CCITT(0), ISO (1) y el conjunto CCITT/ISO (2). Se hace énfasis en la etiqueta IE-ORG(3), el cual viene de la perspectiva de ISO(1). La siguiente rama identifica al Departamento de Defensa (DOD) con el valor 6. INTERNET es localizado debajo de estos nodos con un valor de 1. Dentro de su jerarquía son cuatro nodos; uno es etiquetado Administrador (Mgmt) con un valor de 2. La siguiente entrada de esta rama es etiquetada mib (1). Este ejemplo muestra la ruta para la entrada de la hoja que identifica una interfaz Ethernet. El número de registro completo es 1.3.6.1.2.1.2.1.3.6. Si un agente reporta el estado de una interfaz Ethernet, éste agente debe identificar el tipo de interfaz con este valor de IDENTIFICADOR DE OBJETO.

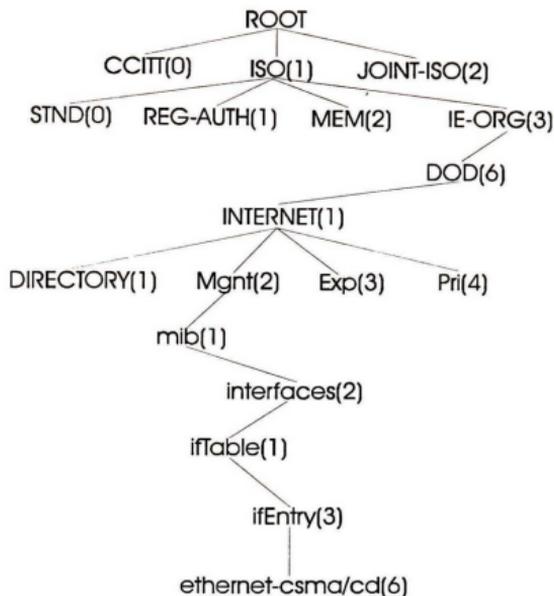


Figura 3-1. Un ejemplo de la jerarquía de registro de INTERNET

3.3 TIPOS Y SINTAXIS DE LA MIB

Los estándares de INTERNET usan construcciones ASN.1 en la MIB para describir la sintaxis de los tipos de objetos. Los siguientes tipos primitivos son permitidos: INTEGER, OCTET STRING, OBJECT IDENTIFIER y NULL. En adición a los tipos primitivos, estos tipos constructor son permitidos: SEQUENCE y SECUENCE OF.

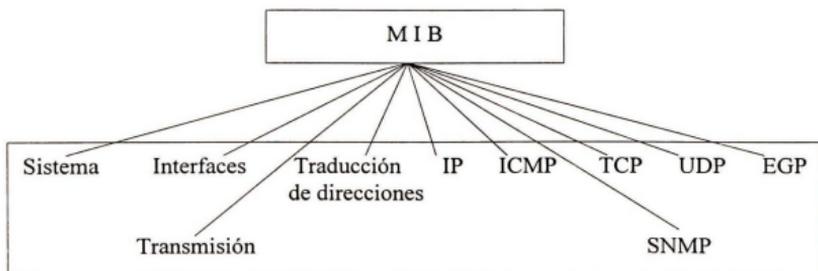
El estándar SMI define seis tipos mayores de objetos administrados de INTERNET. Ellos son los siguientes:

- *Network address*: Este tipo permite una selección de la familia de protocolos de INTERNET. EL tipo es definido en la notación modificada ASN.1 como CHOICE el cual permite seleccionar el protocolo dentro de la familia. Actualmente, sólo la familia INTERNET esta presente.
- *IP Address*: Esta dirección es usada para definir direcciones de 32-bits de INTERNET. La notación ASN.1 es un OCTET STRING.

- *Time Ticks*: Este tipo representa un entero no negativo, el cual es usado para registrar eventos tales como el último cambio para un objeto administrado, la última actualización para la base de datos, etc. El estándar SMI requiere que éste represente un incremento de tiempo de centésimas de un segundo.
- *Gauge*: La definición SMI para este tipo es un entero no negativo el cual puede tener un rango desde 0 a 2^{31-1} . La definición de *Gauge* no permite contar de manera cíclica y éste puede incrementarse o decrementarse en valores.
- *Counter*: Este tipo es definido como un entero no negativo, en un rango de 0 a 2^{31-1} ; sin embargo, este tipo difiere del de *Gauge*, ya que los valores pueden incrementarse de manera cíclica y sólo en valor.
- *Opaque*: Este tipo definido permite a cualquier objeto administrado pasar como un OCTET STRING. Esto es nombrado así porque las codificaciones son pasadas transparentemente.

3.4 ESTRUCTURA DE LA MIB

La estructura de administración de red de INTERNET es organizada alrededor de grupos de objetos. Actualmente, 10 grupos de objetos han sido definidos como miembros de la MIB. La figura 3-2 muestra la composición de los grupos de objetos. Cada uno de estos grupos de objetos es definido en detalle en la MIB de INTERNET. El RFC 1213 proporciona una descripción detallada (con una notación modificada ASN.1) para cada uno de los grupos de objetos.



Nota: Los Grupos de Transmisión y el SNMP han sido adicionados a la MIB.

Figura 3-2. Grupos de Objetos de la MIB.

3.5 REVISION DE LOS GRUPOS DE OBJETOS

Cada grupo de objetos es descrito brevemente en esta sección. El RFC 1213 debe ser estudiado cuidadosamente para apreciar las funciones completas soportadas por las definiciones MIB.

- Grupo de Objetos del Sistema (*The system object group*) describe:
 - (1) Nombre y versión del hardware , así como el sistema operativo y el software de red de la entidad,
 - (2) Nombre jerárquico del grupo e
 - (3) Identificación de cuando (en tiempo) la porción administrada del sistema fue reinicializada.

- Grupo de Objetos de Interfaces (*The interfaces object group*) describe:
 - (1) Número de interfaces de red soportadas,
 - (2) Tipo de interfaz operando bajo el protocolo de INTERNET (IP),
 - (3) Tamaño del datagrama aceptable para la interfaz,
 - (4) Velocidad en bits/2 para la interfaz,
 - (5) Dirección de la interfaz,
 - (6) Estado operacional de la interfaz,
 - (7) Cantidad de tráfico recibido, entregado o descartado y las razones.

- Grupo Traductor de Dirección (*The address translation group*) describe las tablas para la traducción de direcciones, para la dirección física-a-red o para la dirección de red-a-física.

- Grupo IP (*The IP group*) describe:
 - (1) Si la máquina adelanta datagramas,
 - (2) Valor del tiempo de vida para los datagramas que originó éste sitio,
 - (3) Cantidad de trafico recibido, enviado o descartado y sus razones,
 - (4) Información sobre fragmentación de operaciones,
 - (5) Tablas de direcciones, incluyendo mascara subnet e
 - (6) Incluyendo tablas de enrutamiento, dirección de destinatario, distancias métricas, tiempo de ruta, siguiente salto, protocolo desde que ruta fue tomada (RIP), protocolo externo de compuerta (EGP), etc.

- Grupo del Protocolo de Control de Mensajes de INTERNET (*The INTERNET Control Message Protocol (ICMP)*) describe:
 - (1) Número de varios mensajes ICMP recibidos y transmitidos y
 - (2) Estadística sobre los problemas encontrados.

- Grupo del Protocolo de Control de Transmisión (*The Transmission Control Protocol (TCP)*) describe:
 - (1) Algoritmos de transmisión y los valores máximos y mínimos retransmitidos,
 - (2) Número de conexiones TCP que la entidad puede soportar,
 - (3) Información sobre el estado de operaciones en transmisión,
 - (4) Información sobre el tráfico recibido y enviado y
 - (5) Puerto y número de IP para cada conexión.
- Grupo del Protocolo de Datagrama de Usuario (*The User Datagram Protocol (UDP)*) describe:
 - (1) Información sobre el tráfico recibido y enviado e
 - (2) Información sobre problemas encontrados.
- Grupo del Protocolo Externo de Compuerta (*EGP*) describe:
 - (1) Información sobre el tráfico enviado, recibido y problemas encontrados,
 - (2) Tabla de vecino EGP,
 - (3) Dirección a vecinos,
 - (4) Estado EGP con cada vecino.
- Grupo de Transmisión (*The Transmission group*) fue adicionado para la segunda versión de la MIB. Este es para proporcionar información sobre los tipos de interfaces y esquemas de transmisión.
- Grupo del Protocolo de Administración Simple de Red (*SNMP*) fue también adicionado a la MIB. Este contiene 30 objetos que son usados con SNMP. La mayoría de los objetos tratan con:
 - (1) Capacidad de reporte de error y
 - (2) Estadísticas sobre tráfico de SNMP.

3.6 NOTACIONES PARA LOS OBJETOS

Los objetos de INTERNET son descritos nuevamente con ciertas palabras reservadas. Cinco notaciones son usadas para el formato de objeto administrado:

- *Object (Descriptor)*: Describe el objeto en texto ASCII.
- *Syntax*: Describe el tipo de dato ASN.1 utilizado por el objeto.
- *Definition*: Describe en texto el objeto administrado, para ayudar al lector a comprender la notación.

- *Access*: Describe si el objeto administrado puede ser:
 - (1) Sólo de lectura,
 - (2) Sólo de escritura,
 - (3) Lectura y escritura, o
 - (4) No accesible.
- *Status*: Describe información tal como:
 - (1) Sí el objeto es obligatorio,
 - (2) Sí el objeto es opcional,
 - (3) Sí el objeto es obsoleto.

3.7 MODELOS PARA DEFINIR OBJETOS

Todos los objetos son definidos con plantillas y código ASN.1. Las plantillas son examinadas en esta sección. El formato de la plantilla es mostrado en la figura 3-3. Los campos fueron listados y descritos anteriormente.

Cada uno de los grupos de objetos en la fig. 3-2 está definido en el RFC 1213 con su formato de plantilla estándar, es de poco valor que se repitan estas plantillas que consumen arriba de 50 páginas en el estándar. Cada grupo fue descrito brevemente en la sección previa para dar al lector una idea de sus funciones principales. El resto de esta sección muestra un ejemplo, tomado del Grupo de Interfaces de Objeto.

<p><u>OBJECT:</u></p> <p>Un nombre para un tipo de objeto, con su correspondiente OBJECT IDENTIFIER.</p> <p>Sintaxis: El código ASN.1 describe la sintaxis del tipo de objeto.</p> <p>Definición: La descripción textual del tipo de objeto.</p> <p>Acceso (<i>Access</i>): Opciones de Acceso (Access Options)</p> <p>Estado (<i>Status</i>): Estado del tipo de objeto (Status of object type)</p>
--

Figura 3-3. Plantilla para la definición del tipo de objeto de la MIB.

La figura 3-4 muestra la plantilla para el acceso a la hoja del árbol de registro para las interfaces TCP/IP.

OBJECT:

ifType {ifEntry 3}

Sintaxis:

```

INTEGER {
    other (1)      -- ninguno de los siguientes
    regular1822 (2),
    hdh1822 (3),
    ddn-x25 (5),
    rfc877-x25 (5),
    ethernet-csmacd (6),
    iso88023-csmacd (7),
    iso88024-tokenBus (8),
    iso88025-tokenRing (9),
    iso88026-man (10),
    starLan(11),
    proteon-10Mbit (12),
    proteon-80Mbit (13),
    hyperchannel (14),
    fddi (15),
    lapb (16),
    sdlc (17),
    t1-carrier (18),
    cept (19),          -- equivalencia europea del T-1
    basicsdn (20),
    primarylsdn (21),
                    -- propiedad serial
    --... y otros no mostrados,
}

```

Definición:

El tipo de interfaz... inmediatamente abajo
el IP en la pila de protocolo

Acceso:

Sólo lectura.

Estado:

Obligatorio.

-Nota: ... significa que algo del material del RFC es omitido.

Figura 3-4. Plantilla del tipo de interfaz (*ifType*).

Observar que varios nodos intermedios en el árbol no están incluidos en este ejemplo. La notación de *ifType* {*ifEntry* 3} significa que el tipo *ifType* pertenece al padre *ifEntry* 3 en el árbol.

La cláusula de sintaxis describe la sintaxis abstracta en ASN.1 del tipo de objeto. Como muestra la entrada, las interfaces física, de enlace y de subred que existen debajo de IP son descritas y asignadas a un valor entero. Así dos máquinas que intercambian información acerca de la interfaz soportada bajo la capa IP son requeridas para usar estos valores. Por ejemplo, *ifType* = 6 debe ser usado para identificar una Interfaz *Ethernet*, así como el número más alto de nodos en el árbol (como se vio en la figura. 3-1).

3.8 MIB DE ALTO NIVEL

La figura 3-5 muestra la notación en ASN.1 del RFC 1213 para la MIB. El código puede ser entendido en el contexto de los objetos ilustrados en la figura 3-2 y en el árbol jerárquico de nombres en la figura 3-1.

La declaración IMPORTS señala que un número de definiciones son importadas de los RFC 1155 y 1212. Todos los objetos son etiquetados como OBJECT IDENTIFIERS y definidos con otro nombre dentro del árbol de nombres ({*mgmt* 1}, {*MIB-2* 1}, entre otros).

```

RFC1213-MID DEFINITIONS ::= BEGIN
IMPORTS
    mgmt, NetworkAddress, IpAddress, Counter, Gauge, TimeTicks
    FROM RFC1155-SMI;
OBJECT-TYPE
FROM RFC-1212;

mib-2      OBJECT IDENTIFIER ::= {mgmt 1}
system    OBJECT IDENTIFIER ::= {mib -2 1}
interfaces OBJECT IDENTIFIER ::= {mib -2 1}
at        OBJECT IDENTIFIER ::= {mib -2 1}
ip        OBJECT IDENTIFIER ::= {mib -2 1}
icmp     OBJECT IDENTIFIER ::= {mib -2 1}
tcp      OBJECT IDENTIFIER ::= {mib -2 1}
udp      OBJECT IDENTIFIER ::= {mib -2 1}
egp      OBJECT IDENTIFIER ::= {mib -2 1}
cmot     OBJECT IDENTIFIER ::= {mib -2 1}
transmission OBJECT IDENTIFIER ::= {mib -2 1}
snmp     OBJECT IDENTIFIER ::= {mib -2 1}
END

```

Figura 3-5. Definición de los niveles más altos del MIB.

3.8.1 Definiciones de Bajo Nivel

La figura 3-6 muestra el siguiente bajo nivel de definición con relación a la definición de la MIB en la figura 3-5. Por conveniencia, este código muestra en la primera línea el grupo de objetos del sistema, identificado como MIB-2 1.

La figura 3-6 también muestra uno de los objetos en el sistema etiquetado como *sysUptime*. Las entradas en el *sysUptime* codificado definen las características del momento en que el sistema se activo con las definiciones SYNTAX, ACCESS y STATUS explicadas al principio de este capítulo.

System OBJECT IDENTIFIER ::= {MIB -2 1}	Desde la definición de los más altos niveles ver figura anterior.
SysUpTime OBJECT-TYPE SYNTAX TimeTicks CACEES read-only STATUS mandatory ::= {system 3}	Time Ticks (.001s): Tiempo desde la última Reinicialización Parte del grupo de objeto del sistema con ID de 3.

Figura 3-6. Parte de la definición del grupo de objetos del sistema.

3.9 MIB EN MAS DETALLE.

En ésta sección se proporciona información más detallada sobre la MIB, ahora designada MIB-II. El principal cambio para este estándar fue hecho para reflejar nuevos requerimientos, principalmente en nuevas operaciones de definición y en nuevos grupos. Algunos cambios en la editorial fueron hechos para legibilidad y claridad técnica, y varios objetos fueron definidos a futuro. MIB-II introduce el Objeto desaprobadado (*deprecated*). Este termino significa que el objeto debe ser soportado, pero este probablemente será removido en otra versión de la MIB. Actualmente, la MIB-II designa sólo un objeto, una tabla, como desaprobadado. Sin embargo, porque una tabla es desaprobadada, el Grupo de Traducción de Direcciones completo también es desaprobadado. Esto no produce malos efectos sobre las operaciones porque esta capacidad esta presente ahora en el grupo IP.

El apéndice D muestra de manera gráfica y a detalle, cada grupo de objetos de la MIB con sus respectivos objetos y una breve descripción de cada uno de ellos.

CAPITULO 4

PROTOCOLO SNMP

4.1 OBJETIVOS Y ARQUITECTURAS DE SNMP

El RFC 1157 explica los estados de “cómo SNMP minimiza explícitamente el número y la complejidad de las funciones de administración realizadas por el mismo agente”. En otras palabras SNMP está diseñado para ser simple. El costo de desarrollo del software de agente fue reducido, así como el costo a los vendedores en el soporte al protocolo y así poder incrementar la aceptación del protocolo. SNMP es extenso y tiene funciones y obliga a los vendedores ha adicionar funciones de administración de red. Este separa la arquitectura de administración de la arquitectura de los dispositivos de hardware, tales como servidores y enrutadores, ampliando la base del soporte a los fabricantes.

SNMP tiene una arquitectura muy avanzada. La figura 4-1 compara la arquitectura de SNMP al modelo OSI y al modelo DARPA, bajo los cuales se desarrollaron los protocolos INTERNET y TCP/IP. Note que las cuatro capas del modelo DARPA no se mapean a las siete capas del modelo OSI.

Modelo OSI	Función relacionada SNMP	Modelo DARPA
Aplicación	Aplicación de administración (PDU de SNMP)	
Presentación	Estructura de la información administrada (Codificación ASN.1 y BER)	Proceso / Aplicación
Sesión	Autenticación (Cabecera SNMP)	
Transporte	Protocolo Datagrama de Usuario (UDP)	Host - a - Host
Red	Protocolo Internet (IP)	Internet
Enlace	Interface del Protocolo de LAN o WAN	Interface de red
Física		

Figura 4-1. Comparando la arquitectura SNMP con los modelos OSI y DARPA

Usemos un ejemplo para ver como los procesos interactúan con la arquitectura SNMP. Suponga que una consola de administración requiere información de uno de los nodos administrados. Los procesos SNMP, el administrador y el agente, responden a la consola. La codificación ASN.1 de la capa de aplicación proporciona la sintaxis apropiada para los mensajes SNMP. Los procesos permanecen autenticando el dato unido a la cabecera SNMP y comunicando la información requerida. Porque mucha de la información administrada no es liberada como la provee un sistema orientado a conexión, el canal de comunicación entre el administrador SNMP y el agente es sin conexión. Cuando se compara el modelo SNMP con el modelo OSI, el mecanismo de comunicación sin conexión de SNMP remueve algunas de las necesidades para una *capa de sesión* y reduce las responsabilidades de las cuatro capas inferiores. Para otras implementaciones diferentes, UDP realiza las funciones de *la capa de transporte*, IP proporciona las funciones de la *capa de red* y redes LAN como Ethernet o Token-ring proporcionan las funciones de las *capas de enlace y física*. Esta regla tiene algunas excepciones.

Si comparamos SNMP al modelo DARPA, notaremos que el modelo DARPA usa cuatro capas para implantar la función de comunicación. En el modelo DARPA, SNMP reside en la capa de proceso/aplicación. Sin embargo mientras la capa host a host de DARPA proporciona comunicación de extremo a extremo, SNMP usa las propiedades de direccionamiento de puerto y chequeo de suma de UDP; esto no proporciona control de error octeto por octeto. IP proporciona las funciones de la capa INTERNET, tal como direccionamiento y fragmentación, necesario para liberar un mensaje SNMP. Finalmente, la capa de interfaz de red interactua con el hardware en la red.

Desde una perspectiva práctica, el modelo SNMP trabaja como se muestra en la figura 4-2. Esto incluye un sistema administrador que utiliza al administrador SNMP, un agente SNMP, recursos administrados y los mensajes SNMP para la administración de información vía cinco unidades de datos del protocolo (PDUs) de SNMP. La aplicación administradora envía las PDUs *Get*, *Get-next*, o *Set*. El sistema administrado retorna una PDU *GetResponse*. El agente puede crear y enviar una PDU *Trap* cuando las condiciones predefinidas se cumplan. Estas PDUs las veremos más adelante a detalle.

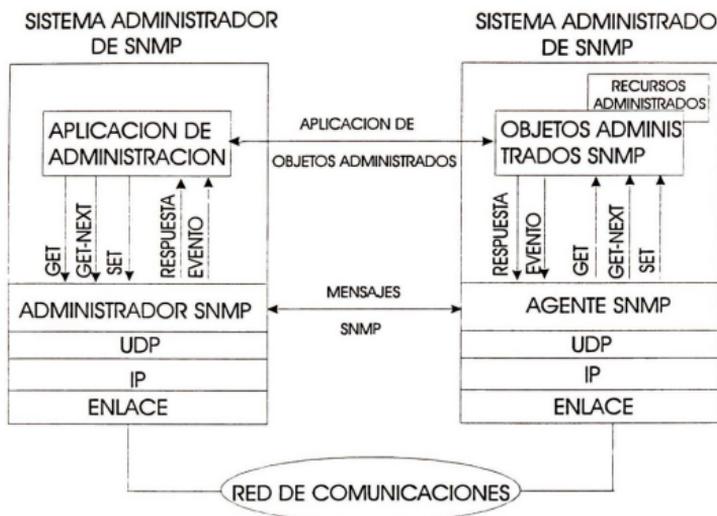


Figura 4-2. Arquitectura de SNMP

4.2 OPERACION DE SNMP

Para que un proceso SNMP de los descritos anteriormente pueda ocurrir en dispositivos físicos, dos conjuntos de procesos lógicos ocurren en estos elementos físicos: las relaciones que son especificadas entre varias entidades administradas de red, y el camino de la información administrada que es comunicado a la red. Por ejemplo, un enrutador podría tener un chip físico que contenga el software que actúa como un agente SNMP.

4.2.1 RELACIONES DE ADMINISTRACION DE RED

La norma SNMP, RFC 1157, y el modelo administrativo SNMP, RFC 1351, definen un número de términos. Muchas de estas definiciones describen relaciones entre entidades administradas:

- *Estaciones para la administración de red:* son dispositivos que ejecutan las aplicaciones de administración que controlan y monitorean los elementos de red.

- *Elementos de red*: son dispositivos tales como *host*, puentes, enrutadores y *hubs* que contienen un agente y realizan las funciones de administración de red que la estación administradora de red requiere.
- *SNMP*: permite a las estaciones administradoras de red y a los agentes en los elementos de red comunicarse.
- *Entidades de aplicación SNMP*: Estas residen en una estación administradora o un nodo administrado, y utiliza el protocolo SNMP como un mecanismo de comunicación.
- *Entidades del protocolo*: Estas son procesos que implementa el protocolo SNMP, soportando así las entidades de aplicación SNMP.
- *Comunidad SNMP*: Identifica un agente SNMP con un conjunto arbitrario de entidades de aplicación SNMP. El administrador de red asigna a la comunidad un nombre (llamado nombre de comunidad) el cual es esencialmente una palabra clave con derechos y privilegios asociados. Una aplicación administrada con múltiples nombres de comunidad podrían corresponder a múltiples comunidades.
- *Autenticación de mensajes SNMP*: Mensajes enviados de una entidad de aplicación a una comunidad SNMP específica. El mensaje contiene el nombre de la comunidad de interés.
- *Esquema de autenticación*: Método por el cual un mensaje SNMP es identificado como perteneciente a una comunidad SNMP específica.
- *Vista MIB*: Subconjunto de objetos MIB, que pertenecen a un elemento de red contenidos en varios subárboles.
- *Modo de acceso*: Determina el nivel de acceso a los objetos que está permitido a una entidad de aplicación particular. Las opciones son de sólo lectura y de lectura escritura.
- *Perfil de la comunidad*: Asocia el modo de acceso SNMP con la vista MIB de SNMP; el perfil de la comunidad representa privilegios específicos de acceso para las variables en una vista MIB.
- *Políticas de acceso SNMP*: Asocian una comunidad SNMP con un perfil de comunidad SNMP. La política de comunidad representa el perfil de comunidad específico que un agente permite tener a otro miembro de la comunidad.
- *Agente proxy SNMP*: Proporciona funciones de administración en favor de elementos de la red que de otra manera son inaccesibles.

4.2.2 IDENTIFICANDO Y COMUNICANDO INSTANCIAS DE OBJETO

Los tipos de objetos administrados SMI tienen un identificador único de objeto (OID) los nombra y localiza en el árbol de objetos. Una instancia de un tipo objeto es una ocurrencia de ese tipo, y tiene un valor asignado. Por ejemplo, el tipo de objeto *sysDescr* {1.3.6.1.2.1.1.1.0} puede tener un valor de “puente remoto modelo 2265M”.

Supongamos que una estación administradora de red desea recuperar una instancia de un objeto específico. La estación administradora utilizará SNMP para comunicarle este requerimiento al agente. Ahora, supongamos que múltiples instancias de ese objeto son posibles. Por ejemplo, digamos que la tabla de ruteo de un enrutador contiene un número de entradas. ¿Cómo podría la estación de administración de red recuperar solo el valor de la tercera entrada en la tabla? El RFC 1157 especifica esta tarea. Para estas operaciones de SNMP, un “nombre de variable” identifica de manera única a cada instancia de tipo de objeto. Este nombre consiste de dos partes de la forma *x.y*. La porción *x* es el tipo de objeto definido en la MIB, y la porción *y* es un fragmento del OID que identifica la instancia deseada. El siguiente ejemplo aclarará esto.

Considere un objeto escalar que tiene una instancia. Los objetos contenidos en el grupo de sistema son todos escalares. Por ejemplo, El tipo de objeto *sysServices* tiene un OID {1.3.6.1.2.1.1.7} y ocurre una vez. La porción *x* del nombre de la variable es el OID, y la porción *y* esta asignada a cero(0). Se puede derivar esto, siguiendo hacia abajo del árbol OID al objeto *sysServices* y agregando la instancia apropiada del sufijo (con el sufijo, o porción *y*).

iso	org	dod	internet	mgmt	mib	systems	sysServices	Instance
1	3	6	1	2	1	1	7	0

Así, el nombre de la variable para *sysServices* es {1.3.6.1.2.1.1.7.0}.

El nombre de variable para un objeto columnar es más complicado por que éste puede identificar la localización de una columna en un renglón de la tabla. Considere el tipo de objeto tabla de la dirección IP, *ipAdEntBcastAddr*, que especifica el valor del bit menos significativo(LSB) de la dirección de difusión(broadcast address). Para comenzar, seguimos en el árbol OID hacia abajo a *ipAdEntBcastAddr*:

iso	org	dod	internet	mgmt	mib	ip	ipAddrTable	ipAddrEntry	ipAdEntBcastAddr
1	3	6	1	2	1	4	20	1	4

El OID es {1.3.6.1.2.1.4.20.1.4}, consiste del grupo IP {1.3.6.1.2.1.4}, la tabla de direcciones IP(20), el *ipAddrEntry* (1), y el tipo de objeto *ipAdEntBcastAddr* (4). Enseguida, la ruta destino es agregada como un sufijo, que es, a.b.c.d. (más una dirección IP y notación decimal punteada). El nombre de la variable para *ipAdEntBcastAddr* asociado con la dirección IP será {1.3.6.1.2.1.4.20.1.4.a.b.c.d}.

Un ejemplo final es de la tabla de conexión, *tcpConnTable*. Suponga que deseamos recuperar el estado de la conexión entre el puerto 575 en la dirección local {a.b.c.d} y el puerto 441 en la dirección remota {w.x.y.z}. El OID para *tcpConnState* es {1.3.6.1.2.1.6.13.1.1}. El sufijo y se expresa como {a.b.c.d.575.w.x.y.z.441}. Por lo tanto, el nombre completo de la variable sería

{1.3.6.1.2.1.6.13.1.1.a.b.c.d.575.w.x.y.z.441}

Los siguientes ejemplos muestran nombres de variables específicos para tipos de objetos escalares y columnares:

- Descripción de los servicios del sistema:
sysServices ::=
{1.3.6.1.2.1.1.7.0}
- Velocidad de la interfaz:
ifSpeed.3 ::=
{1.3.6.1.2.1.2.2.1.5.3}
- Estado de una conexión TCP, entre el puerto local e, la dirección local {a.b.c.d}, y el puerto remoto j, la dirección remota {f.g.h.i}:
tcpConnState.a.b.c.d.e.f.g.h.i.j ::=
{1.3.6.1.2.1.6.13.1.1.a.b.c.d.e.f.g.h.i.j}
- Verificación de una lista UDP que está operando en el puerto especificado de la dirección IP a.b.c.d:
udpLocalAddress.a.b.c.d.e. ::=
{1.3.6.1.2.1.7.5.1.1.a.b.c.d.e}
- Número de mensajes SNMP enviados a este dispositivo con desconocimiento del nombre de la comunidad (un escalara):
snmplnBadCommNames ::=
{1.3.6.1.2.1.11.4.0}

Con este respaldo en los métodos de identificación de instancias de objetos, discutiremos las unidades de datos del protocolo SNMP (PDUs) que llevan los requerimientos y respuestas de información entre el administrador y el agente. Los PDUs usan los ejemplos de las instancias de objetos mostradas para identificar la información específica de la red administrada que el administrador está buscando.

4.3 UNIDADES DE DATOS DEL PROTOCOLO SNMP (PDUs)

Comenzaremos a hablar de las PDUs describiendo la posición del mensaje SNMP en un *cuadro* transmitido. El cuadro es la unidad de información transmitida entre nodos

de red. Por ejemplo, un formato de cuadro IEEE 802.5 define la transmisión entre nodos *Token-Ring*, y un formato ANSI T1.617 define la transmisión entre nodos *Frame Relay*. (Checar el cuadro de UDP).

La cabecera y remolque definida por el protocolo LAN o WAN delimita el cuadro (Ver figura 4-3). El dato transmitido es llamado un *datagrama del Protocolo Internet (IP)*. EL datagrama IP es una unidad de información contenida en sí misma, enviada del host fuente al destino. Dentro del datagrama está una dirección IP destino que guía el datagrama al receptor propuesto. Enseguida, la cabecera del Datagrama de Usuario del Protocolo (UDP) identifica la capa más alta del proceso del protocolo (SNMP) que procesará el datagrama, y proporcionará el control de error utilizando un chequeo de suma. El mensaje SNMP es la parte más profunda del cuadro, llevando los datos actuales del administrador al agente y viceversa.

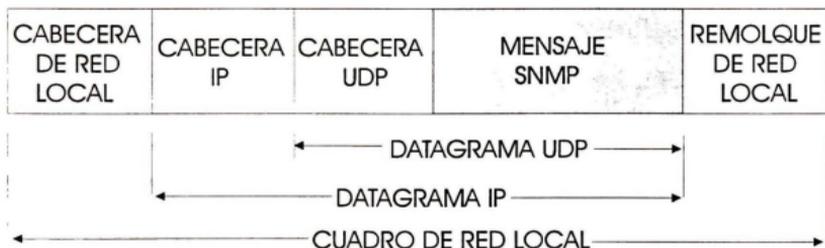


Figura 4-3, Mensaje SNMP en un cuadro de transmisión

Cuando la IP es demasiado larga para meterla en un cuadro, ésta es fragmentada en varios cuadros para transmitirla en la red. Por ejemplo, un datagrama que contiene 2500 octetos requiere dos cuadros *Ethernet*, cada uno de estos puede contener 1500 octetos de datos de la capa más alta. La estructura general de cada cuadro, como se muestra en la figura 4-3 queda igual.

El mensaje SNMP en sí mismo es dividido en dos secciones: un identificador de versión mas el nombre de la comunidad, y una PDU. La primer sección, el identificador de versión y el nombre de la comunidad son referenciados algunas veces como la *cabecera de autenticación SNMP*. Hay cinco diferentes tipos de PDU: *GetRequest*, *GetNextRequest*, *GetResponse*, *SetRequest*, y *Trap*. Las PDUs *Get*, *Set*, y *Response* tienen un formato común (Vea la figura 4-4), mientras el formato del PDU *Trap* es único (Vea la figura 4-8).

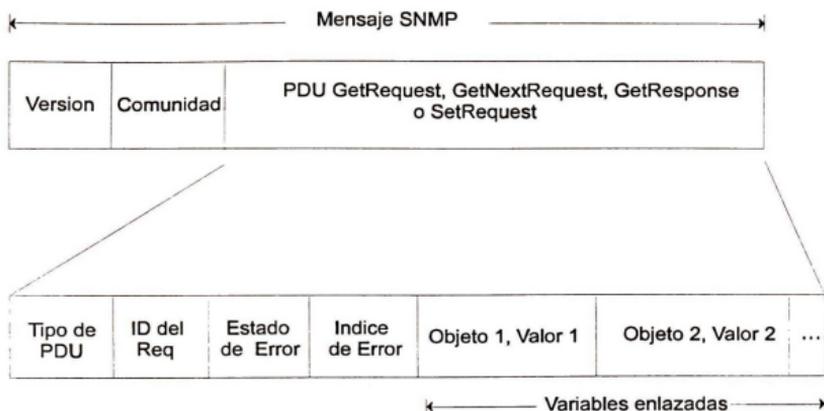


Figura 4-4. Estructura de las PDUs GET, GETNEXT, SET y RESPONSE de SNMP

El número de versión (un tipo INTEGER) asegura que el agente y el administrador están utilizando la misma versión de protocolo SNMP. Los mensajes que contienen diferente número de versión entre agente y administrador son descartados. El nombre de la comunidad (un tipo OCTET STRING), valida con el administrador antes de permitirle el acceso al agente. El nombre de la comunidad, junto con la dirección IP del administrador, es almacenado en el perfil de la comunidad del agente. Si hay diferencia entre los valores del administrador y el agente para el nombre de la comunidad, el agente enviará un mensaje *Trap* de autenticación de falla al administrador. Si el número de versión y el nombre de la comunidad son iguales a los almacenados en el agente, la PDU SNMP se comienza a procesar.

4.3.1 FORMATO DE LAS PDU GET, SET Y RESPONSE

Las PDUs *GetRequest*, *GetNextRequest*, *SetRequest* y *GetResponse* comparten un formato común (vea la Figura 4-4). El primer campo, tipo de PDU, especifica el tipo de PDU que contiene el mensaje.

PDU	Valor del campo tipo de PDU
GetRequest	0
GetNextRequest	1
GetResponse	2
SetRequest	3
Trap	4

El campo *Request ID* es un tipo ENTERO que relaciona el requerimiento del administrador a la respuesta del agente. El campo *Error Status* es un tipo ENTERO enumerado que indica operación normal o una de cinco condiciones de error. Los posibles valores son:

Error	Valor	Significado
NoError	0	Operación propia entre Administrador/agente
TooBig	1	El tamaño del PDU <i>GetResponse</i> requerido excede una limitación local
NoSuchName	2	El nombre del objeto requerido no es igual al nombre disponible en la vista relevante de la MIB.
BadValue	3	Un <i>SetRequested</i> contiene un tipo, longitud, y valor inconsistente para la variable.
ReadOnly	4	No definido en el RFC 1157 (pero algunos fabricantes sufren de este error)
GenErr	5	Otros errores, no definidos explícitamente, han ocurrido.

Cuando ocurre un error, el campo *error Index* identifica la entrada en la variable ligada a la lista que causo el error. Por ejemplo, si un error de sólo lectura(ReadOnly) ocurrió, éste retornará un Error Index = 4.

Una variable enlazada (*VarBind*) une el nombre de la variable con su valor. Una *VarBindList* es una lista creada por tal unión. Note que en los campos de variables enlazadas de los PDUs SNMP, la palabra *objeto* identifica el nombre de variable (codificando el OID del tipo objeto más la instancia) para que un valor sea comunicado. Además nótese que los PDUs *GetRequest* o *GetNextRequest* utilizan un valor NULL, el cual es un tipo de dato especial.

4.3.2 UTILIZACION DE LA PDU GETREQUEST

El administrador utiliza la PDU *GetRequest* para recuperar el valor de uno o más objetos de un agente. En muchos casos, estos objetos son escalares no columnares. Para generar la PDU *GetRequest*, el administrador asigna PDU Type =0, un ID de requerimiento definido localmente, e inicializa *ErrorStatus* y *ErrorIndex* a 0. Una lista de variables enlazadas, contiene las variables requeridas, y sus valores NULL correspondientes, completan la PDU. Bajo condiciones libres de error, el agente genera una PDU *GetResponse*, a la que se le asigna PDU Type = 2, el mismo valor del ID del requerimiento, *Error Status* = *noError*, y *Error Index* = 0. Las variables enlazadas contienen ahora los valores asociados con cada una de las variables marcadas en la PDU *GetRequest* (vea la figura 4-5). Recordemos que el termino *variable* se refiere a una instancia de un objeto administrado.

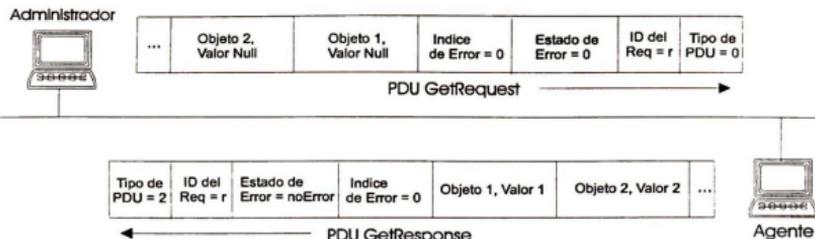


Figura 4-5. Transmisión de PDUs GetRequest/GetResponse

Cuatro condiciones de error son posibles:

- Si una variable en el campo de variables enlazadas no es exactamente igual a un objeto disponible, el agente retorna una PDU *GetResponse* con *Error Status = noSuchname*, y con *Error Index* indicando el índice de la variable en cuestión.
- Si una variable es un tipo agregado, tal como un objeto renglón, el agente retorna una PDU *GetResponse* con *Error Status = noSuchName*, y *Error Index* indicando el índice de la variable en cuestión.
- Si el tamaño de la PDU *GetResponse* apropiada excede una limitante local, entonces el agente retorna una PDU *GetResponse* de forma idéntica, con *Error Status = tooBig*, y *Error Index = 0*.
- Si el valor de una variable requerida no es recuperado por ninguna otra razón, entonces el agente retorna una PDU *GetResponse*, con *Error Status = genErr*, y *Error Index* indicando el índice de la variable en cuestión.

4.3.3 UTILIZACION DE LA PDU GETNEXTREQUEST

El administrador utiliza la PDU *GetNextRequest* para recuperar uno o más objetos y sus respectivos valores de un agente. En muchos casos, estos múltiples objetos residirán en una tabla. Como se ve en la figura 4-6, para generar la PDU *GetNextRequest*, el administrador asigna PDU *Type = 1*, un ID de requerimiento definido localmente, e inicializa *ErrorStatus* y *ErrorIndex* a 0. Una lista de variables enlazadas (*VarBindList*) conteniendo los OIDs y los valores NULL correspondientes, completan la PDU. Estos OIDs pueden ser cualquier OID (que puede ser una variable) que inmediatamente preceden a la variable y valor retornado. Bajo condiciones libres de error, el agente genera una PDU *GetResponse*, que es asignada PDU *Type = 2*, el mismo valor del ID del requerimiento, *Error Status = noError*, y *Error Index = 0*. La variable de enlace contiene el nombre y

valor asociado con el sucesor lexicográfico de cada uno de los OIDs anotados en la PDU *GetNext Request*.

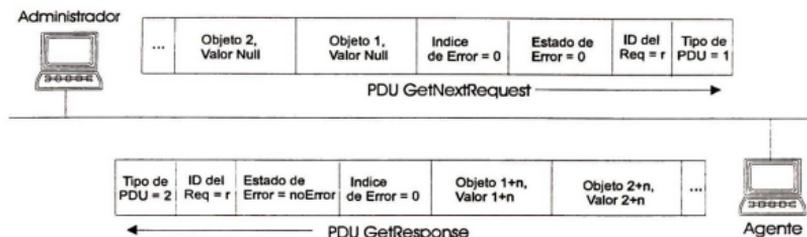


Figura 4-6. Transmisión de PDUs *GetNextRequest/GetResponse*

La principal diferencia entre la PDU *GetRequest* y *GetNextRequest* es la palabra lexicográfica. Esto significa que *GetNextRequest* recupera el valor del objeto siguiente al, que se recuperó anteriormente, en la vista de la MIB del agente.

Tres condiciones de error son posibles:

- Si una variable en el campo de variables de enlace no precede lexicográficamente el nombre de un objeto que puede ser recuperado (esto es, un objeto disponible para operaciones *Get* y relevante en la vista de la MIB), el agente retorna una PDU *GetResponse* con *Error Status = noSuchName*, y con *Error Index* indicando el índice de la variable en cuestión. Esta condición es llamada “*running off the end of the MIB View*”.
- Si el tamaño de la PDU *GetResponse* excede una limitante local, el agente retorna una PDU *GetResponse* de forma idéntica, con *Error Status = tooBig*, y *Error Index = 0*.
- Si el valor del sucesor lexicográfico de una variable requerida en el campo de variables ligadas, no es recuperado por ninguna otra razón, el agente retorna una PDU *GetResponse*, con *Error Status = genErr*, y *Error Index* indicando el índice de la variable en cuestión.

4.3.4 UTILIZACION DE LA PDU SETREQUEST

El administrador utiliza la PDU *SetRequest* para asignar un valor a un objeto residente en un agente. Como se ve en la figura 4-7, para generar la PDU, el administrador asigna *PDU Type = 3*, un ID de requerimiento definido localmente, y *ErrorStatus* y

ErrorIndex inicializados en 0. Una lista de variables enlazadas (*VarBindList*), conteniendo las variables especificadas y sus valores correspondientes, completan la PDU. Cuando el agente recibe la PDU *SetRequest*, ésta altera los valores de los objetos nombrados al valor en la variable enlazada. Bajo condiciones libres de error, el agente genera una PDU *GetResponse* de forma idéntica, excepto que la *PDU Type = 2*, *Error Status = noError*, y *Error Index = 0*.

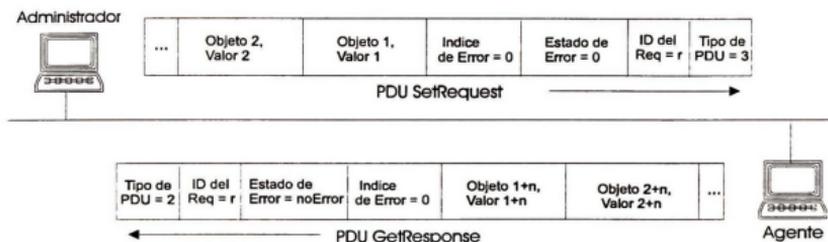


Figura 4-7. Transmisión de PDUs SetRequest/GetResponse

Cuatro condiciones de error son posibles:

- Si una variable en el campo de variables enlazadas no está disponible para operaciones de inicialización (*Set*) en la vista relevante de la MIB, el agente retorna una PDU *GetResponse* de forma idéntica, con *Error Status = noSuchname*, y con *Error Index* indicando el índice del nombre de objeto en cuestión.
- Si el valor de una variable nombrada en el campo de variables enlazadas no conforma al tipo, longitud y valor ASN.1 requeridos, el agente retorna una PDU *GetResponse* de forma idéntica, con *ErrorStatus = bad-Value* y *Error Index* indicando el índice de la variable en cuestión.
- Si el tamaño de la PDU *GetResponse* apropiada excede una limitante local, el agente retorna una PDU *GetResponse* de forma idéntica, con *Error Status = tooBig*, y *ErrorIndex = 0*.
- Si el valor de una variable no puede ser alterado por ninguna otra razón, el agente retorna una PDU *GetResponse* de forma idéntica, con *ErrorStatus = genErr* y *Error Index* indicando el índice de la variable en cuestión.

4.3.5 FORMATO DE LA PDU TRAP

La PDU *Trap* tiene un formato distinto de los otros cuatro PDUs SNMP, como se ve en la figura 4-8. El primer campo indica La PDU *Trap* y contiene *PDU Type = 4*. El campo

enterprise identifica el registro de autoridad de la empresa administradora bajo el cual la *trampa* fue definida. Por ejemplo, El prefijo OID {1.3.6.1.4.1.110} identifica a *Enredos Computacionales S.A.*, como la empresa que envió el *trap*. El campo de dirección de agente, que contiene la dirección IP del agente, proporciona identificación adicional. Si un protocolo de transporte no IP es utilizado, el valor 0.0.0.0 es regresado.

Otro campo es *Tipo genérico de trampa*, el cuál provee información más específica en el evento reportado. Hay siete valores definidos (tipos Enteros enumerados) para este campo:

Trap	Valor	Significado
ColdStart	0	La entidad emisora del protocolo (capa más alta de la administración. de red) se reinició, indicando que la configuración del agente o la implementación de la entidad puede ser alterada.
WarmStart	1	El emisor del protocolo se reinició, pero la configuración del agente o la implementación de la entidad del protocolo a sido alterada.
LinkDown	2	Una liga de comunicación a fallado. La interfaz afectada es identificada como el primer elemento en el campo de variables enlazadas: nombre y valor de la instancia <i>ifIndex</i> .
LinkUp	3	Una liga de comunicación se ha establecido. La interfaz afectada es identificada como el primer elemento en el campo de variables enlazadas: nombre y valor de la instancia <i>ifIndex</i> .
AuthenticationFailure	4	El agente a recibido del administrador un mensaje SNMP impropriadamente autenticado; esto es, el nombre de la comunidad es incorrecto.
EgpNeighborLoss	5	Un punto colindante EGP no está activo.
EnterpriseSpecific	6	Una trampa no genérica a ocurrido, está es identificada adicionalmente por el campo <i>tipo específico de trap</i> , y el campo Empresa (<i>Enterprise</i>).

Dos campos adicionales completan la PDU *Trap*. El campo *TimeStamp* contiene el valor del objeto *sysUpTime*, representando la cantidad de tiempo entre la última reinicialización del agente, y la generación de la trampa. El último campo contiene las variables enlazadas.

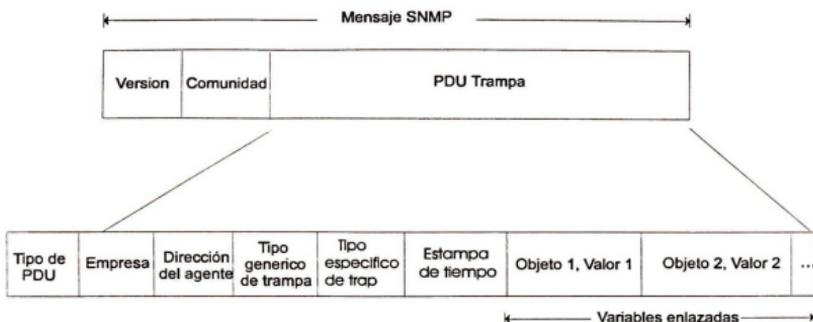


Figura 4-8. Estructura de la PDU Trap de SNMP

4.3.6 UTILIZACION DE LA PDU TRAP

El agente utiliza la *PDU Trap* para alertar al administrador que un evento predefinido a ocurrido. Para generar la *PDU Trap*, el agente asigna *PDU Type = 4*, y llena en los campos *Enterprise*, *Agent Address*, *Generic Trap*, *Specific Trap Type*, *Time-stamp* y la lista de variables enlazadas.

Por definición y convención, las trampas son aplicaciones específicas. Así mismo, será difícil cubrir el rango de usos para está PDU. El RFC 1215, "Una convención para definir trampas para usarse con SNMP", ofrece algunas guías para su uso. La figura 4-9 ilustra como un agente utiliza una trampa para comunicar un evento significativo al administrador.

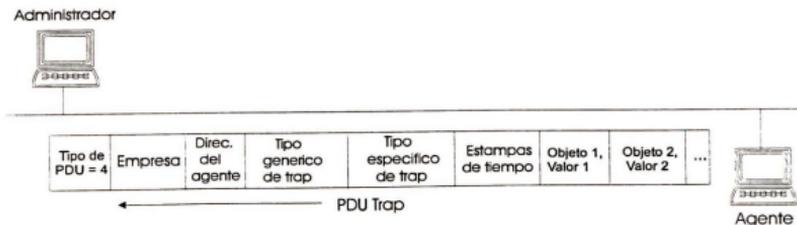
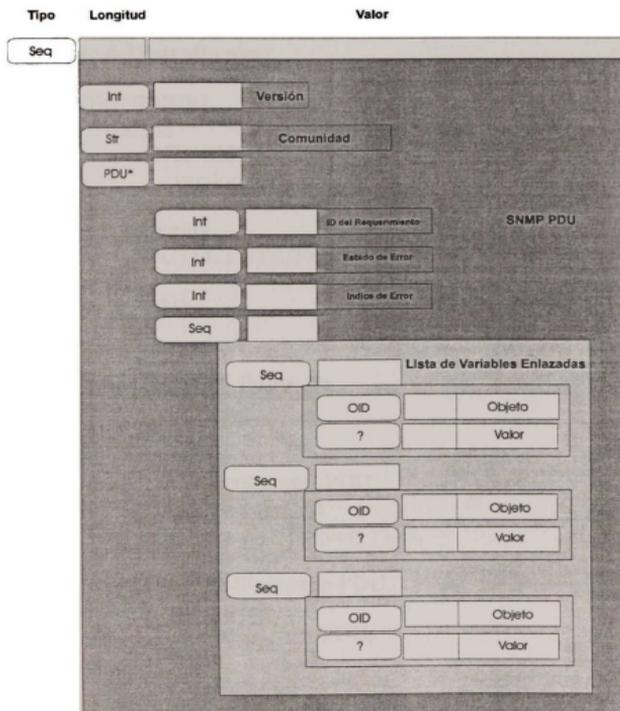


Figura 4-9. Transmisión del PDU Trap

4.3.7 CODIFICACION DE LA PDU DE SNMP

La PDU de SNMP está codificada utilizando la clase contexto específico, con una etiqueta que identifica a la PDU. Los campos *Longitud* y *Valor* son construidos para transportar una estructura particular y cantidad de información. Ahora que se ha discutido la estructura de las PDUs de SNMP, se pueden revisar estas codificaciones con más detalle.

La figura 4-10 muestra el ejemplo de una codificación *Tipo-Longitud-Valor* (TLV) de una PDU de SNMP. Nótese que la codificación entera comienza con un tipo SEQUENCE OF. La versión es un tipo INTEGER, y el nombre de la comunidad es un tipo OCTET STRING. Un tipo *contexto-especifico* que indica la PDU y su longitud. Tres tipos INTEGER proporcionan *Request ID*, *ErrorStatus*, y *ErrorIndex*. La lista de variables enlazadas, consiste de múltiples codificaciones SEQUENCE OF, que completan la PDU.



* Tipo de contexto-especifico que identifica la PDU

Figura 4-10. Codificación TLV de un PDU de SNMP

4.4 DEFINICION ASN.1 DE SNMP

Para concluir la plática de las operaciones del protocolo SNMP, la definición que se muestra más delante, es la definición ASN.1 de SNMP (RFC 1157). De especial interés son los constructores de las PDUs de SNMP. Estos constructores resumen las variables utilizadas en las PDUs, mas el valor que estas variables pueden asumir.

Definición ASN.1 de SNMP

```

RFC1157-SNMP DEFINITIONS ::= BEGIN
IMPORTS
    ObjectName, Objectsyntax, NetworkAddress, IpAddress, TimeTicks
    FROM RFC1155-SMI;

-- mensaje del mas alto nivel
Message ::=
    SEQUENCE{
        versión                -- version-1 para este RFC
        INTEGER{
            version-1(0)
        },
        community              -- nombre de comunidad
        OCTECT STRING,
        data                   -- e.g., si los PDUs son triviales
        ANY                    -- la autenticación a sido usada
    }

-- Unidades de Datos del Protocolo
PDUs ::=
    CHOICE{
        get-request
        GetRequest-PDU,
        get-next-request
        GetNextRequest-PDU,
        get-response
        GetResponse-PDU,
        set-request
        SetRequest-PDU,
        trap
        Trap-PDU
    }

-- PDUs
GetRequest-PDU ::=

```

```

[0]
    IMPLICIT PDU

GetNextRequest-PDU ::=
    [1]
        IMPLICIT PDU
GetResponse-PDU ::=
    [2]
        IMPLICIT PDU
SetRequest-PDU ::=
    [3]
        IMPLICIT PDU
PDU ::=
    SEQUENCE{
        request-id
            INTEGER,
        error-status          -- ignorado algunas veces
            INTEGER{
                noError(0),
                tooBig(1),
                noSuchName(2),
                badValue(3),
                ReadOnly(4),
                genErr(5)
            },
        error-index          -- ignorado algunas veces
            INTEGER,
        variable-bindings   -- los valores son ignorados algunas veces
            VarBindList
    }
Trap-PDU ::=
    [4]
        IMPLICIT SEQUENCE {
            enterprise       -- generando tipo de objeto
                            -- trampa, vea sysObjectID en [5]

            OBJECT IDENTIFIER,
            agent-addr       -- generando la dirección del objeto
                NetworkAddress, -- trampa
            generic-trap     -- tipo genérico de trampa
            INTEGER{
                coldStart(0),
                warmStart(1),
                linkDown(2),
                linkUp(3),
                authenticationFailure(4),
                egpNeighborLoss(5),

```

```
        enterpriseSpecific(6)
    },
    specific-trap      -- código específico del nivel presente
                      INTEGER, -- si el trap genérico no es específico de
                              la empresa
    time-stamp        -- lapso de tiempo entre la última
                      TimeTicks, -- reinicialización de la red
    variable-bindings -- información interesante
                      VarBindList
}

-- variables enlazadas
VarBind :=
    SEQUENCE{
        name
            ObjectName,
        value
            ObjectSyntax
    }

VarBindList :=
    SEQUENCE OF
        VarBind

END
```

CAPITULO 5

**PROPUESTAS DE
ADMINISTRADOR Y DE
AGENTE**

5.1 PROPUESTA DE AGENTE

Como se vio en el capítulo 1, el agente es uno de los tres componentes importantes en la administración de redes. En este capítulo se verá a detalle que es, como funciona y el esquema propuesto de un agente. Hay que mencionar que el desarrollo de un agente es difícil y complejo, lo cual consume tiempo y muchos recursos.

Hay diferentes tipos de agentes, un tipo común es el agente de monitoreo, el cual simplemente monitorea los recursos administrados y envía los datos administrados recolectados al administrador que lo está solicitando. Algunos agentes pueden ejecutar ambas operaciones, las de control y las de monitoreo, mientras otros sólo pueden analizar datos administrados.

Hay además agentes “no persistentes”, los cuales son iniciados por un administrador, realizan una sola operación, y entonces terminan. Los agentes de este tipo son frecuentemente utilizados por el administrador, para realizar análisis matemáticos de computo intensivo.

Los puntos de análisis que se tocarán para describir todo lo que es un agente son los siguientes:

- funcionalidad,
- operaciones,
- desarrollo,
- servicios y
- arquitectura

Cada uno de los puntos mencionados anteriormente, complementa a los otros en la estructura general, para conocer a fondo el esquema de un agente para la administración de red. La meta aquí es poder construir un agente genérico que requiera la mínima entrada de datos del usuario, y que ésta sea concisa y fácil.

5.1.1 FUNCIONALIDAD

Un agente debe ser capaz de responder a los requerimientos de los administradores, de otros agentes y de los objetos administrados.

5.1.2 OPERACIONES

Las siguientes clases de operaciones deben ser soportadas por el agente:

AUTODESCRIPCION

Un agente debe ser capaz de describirse a sí mismo, Ante cualquier otra entidad administradora, ya sea un administrador, otro agente o un objeto administrado dentro del mismo nodo de red o en cualquier otro. Esto con la finalidad de que las entidades administradoras puedan conocer (descubrir) que es capaz de hacer o para que tipo de requerimientos, está preparado este agente en particular. Adicionalmente, esta operación permite sólo a los administradores “enlazar” dinámicamente a cualquier agente.

Las operaciones de autodescripción son las siguientes:

- *DescribeMyself₂*: Permite al agente describirse a sí mismo y sus capacidades.
- *GetMOList₂*: Lista de objetos que están siendo administrados actualmente. Esta operación básicamente atraviesa el árbol de objetos administrados, y regresa una lista de objetos. Se puede obtener un parámetro (MO) que permita al requisitante limitar o controlar el atravesar el árbol de objetos administrados.

El parámetro *MO* es un especificador que funciona para especificar uno o más objetos administrados. La estructura depende del protocolo utilizado. Este puede ser una lista de variables SNMP. En este caso, *MO* puede ser utilizado para especificar cómo atravesar el árbol de objetos administrados; por ejemplo, en SNMP, éste puede ser usado como un parámetro para la operación *Get-Next*. *MO_Name* contiene una lista de nombres de los objetos administrados actualmente existentes.

OPERACIONES COMUNES DE ADMINISTRACION

Estas operaciones comunes representan al protocolo de administración (en este caso, SNMP). Estas incluyen operaciones para obtener información de los recursos administrados, inicializar información en los recursos administrados, enviar comandos a los recursos administrados, y crear y borrar recursos administrados. Hay que notar que el crear y borrar objetos administrados se refiere a crear y borrar entidades abstractas que representan a los recursos administrados, no a los recursos reales en sí. Adicionalmente estas operaciones pueden ser utilizadas por los objetos administrados para enviar notificaciones de eventos a los administradores. Estas operaciones deben ser soportadas en todo agente administrado.

Las operaciones comunes de administración son las siguientes:

- *Trap₂*: Permite a un objeto administrado enviar una notificación al administrador.
- *Get₂*: Regresa el valor de atributos específicos de los objetos administrados especificados.

- *Set*_i: Inicializa a un valor específico de cada uno de los atributos especificados en cada objeto administrado.

5.1.3 DESARROLLO

Hay muchas decisiones que deben ser tomadas en el desarrollo de agentes, éstas son las siguientes:

- Servicios a ofrecer e
- Interacción con el ambiente (recursos requeridos de hardware o software, la interfaz de usuario, entre otros.)

Parte de las razones para la dificultad en el desarrollo de los agentes, es que al diseñarlos no se tienen separados ni estas decisiones, ni ningún otro componente. Por ejemplo, hay un conjunto de servicios que es requerido por todos o la gran mayoría de los agentes.

5.1.4 SERVICIOS

El conjunto de servicios que es requerido por todos o la gran mayoría de los agentes, son los siguientes:

- Aceptar acciones de monitoreo y requerimientos de control
- Ejecutar estos requerimientos
- Regresar los resultados
- Notificar los eventos predeterminados de interés, y
- Comunicarse con otras entidades de administración, ya sea un administrador, otro agente, un objeto administrado y otras aplicaciones de administración.

5.1.5 ARQUITECTURA

En esta sección, se describe cada uno de los componentes de la arquitectura propuesta, así como la información necesaria para automatizar la creación de cada componente. Los componentes de la arquitectura propuesta de un agente genérico se muestran en la figura 5-1.

Cuando un requerimiento llega al agente, lo hace por el componente de comunicaciones; a su vez, el requerimiento es pasado al componente verificador de requerimientos, donde es checado. El requerimiento entonces es enviado al componente apropiado donde éste es ejecutado. En el modulo de error si es el caso, se aplica la acción correspondiente para eliminar los errores, o descartar el requerimiento. En el caso del componente interfaz de objeto administrado, checará el objeto correspondiente y cualquier resultado obtenido o generado, será enviado al requisitante nuevamente, a través del componente de comunicaciones.

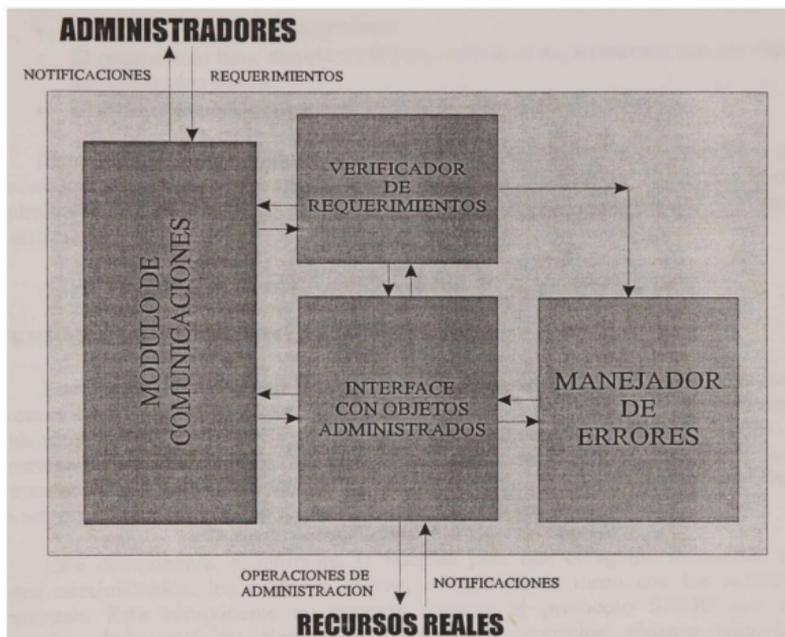


Figura 5-1. Arquitectura propuesta de un agente

COMPONENTE DE COMUNICACIONES

Este componente es el principal del agente. Ordena los requerimientos y pasa la información solicitada al componente apropiado, reconoce el protocolo de administración SNMP que se está utilizando, proporciona los servicios de comunicaciones a otros componentes, tal como el servicio para permitir al agente recibir requerimientos, retornar información al requisitante, y adelantar las notificaciones de los objetos administrados a los

administradores. El protocolo SNMP es utilizado para todo lo que este componente necesita conocer. Además permite conocer cuales servicios son proporcionados por el agente.

COMPONENTE VERIFICADOR DE REQUERIMIENTOS

Este componente verifica cada requerimiento que llega para estar seguro que las siguientes condiciones se cumplen:

- El agente soporta el requerimiento
- El requisitante tiene autorización para realizar el requerimiento con los objetos administrados dados.
- El o los objetos administrados indicados existan y/o sean validos.

Para verificar estas condiciones, este componente debe tener conocimiento de que servicios son soportados por el agente, así como qué clases de objeto administrado pueden ser administradas. Para esto, no es necesario conocer que protocolo de administración se está utilizando.

INTERFAZ CON EL OBJETO ADMINISTRADO

Este componente contiene los objetos administrados, los cuales son abstracciones de los recursos administrados. Cada objeto administrado “representa” a un solo recurso para propósitos de administración. Hay que recordar que un recurso administrado puede o no ser un recurso físico. Por ejemplo, un sistema administrador tendrá un objeto administrado representando una aplicación, la cual en si misma contiene varios objetos administrados, cada uno representa un proceso diferente el cual es parte de esta aplicación.

Este componente, proporciona la interfaz para que el agente interactúe con los objetos administrados, los cuales a su vez, interactúan en turno con los recursos que representan. Este componente no necesita conocer el protocolo SNMP que se está utilizando, dado que no tiene que comunicarse con los objetos administrados necesariamente a través de él. Los objetos administrados necesitan comunicarse con recursos externos, pero el método o protocolo que ellos utilizan, para hacer esta comunicación la decisión es hecha por el desarrollador de objetos administrados, y ésta no es relevante para el diseño del agente.

La única información que el desarrollador de agentes puede sustituir para crear este componente automáticamente es la lista de operaciones de administración que serán soportadas.

En el caso de SNMP, los objetos administrados como tales no existen. Lo cual da como resultado, que los "objetos administrados" en SNMP son simplemente variables. Un especificador de objetos administrados (*MO_Specifier*) es simplemente una lista de variables.

COMPONENTE MANEJADOR DE ERROR

Este componente oculta algunos de los detalles del protocolo SNMP del resto del agente. Traduce códigos internos de error a un formato reconocible por un administrador de red, el cual puede entonces diagnosticar y posiblemente corregir los problemas. El conocimiento del protocolo de administración SNMP, es utilizado por el agente si es necesario, para la creación de este componente.

5.1.6 CONCLUSIONES DE LA PROPUESTA DE AGENTE

Una vez vistos los diferentes componentes de un agente, se puede apreciar que la ventaja obvia de tener componentes modulares, es que cada componente no es dependiente en los detalles de implementación de los otros componentes. Por ejemplo, para codificar y automatizar el componente manejador de error, solo se necesita conocer el protocolo de administración SNMP. Sea o no utilizado el agente por ejemplo, los servicios o accesos definidos por el usuario no tienen efecto en el código para este componente. Este rechazo de código implica otra ventaja: incrementa la formalidad. El mismo código manejador de error es utilizado en todos los agentes con el mismo protocolo sin hacer caso de otros servicios ofrecidos, ni de las clases soportadas de objetos administrados. Si este código es probado y se encuentra que es estable en una implementación particular, se puede asumir que el código trabajará en otras implementaciones, los detalles de implementación de otros componentes no afectan a éste. Una vez que el código es estable, este además puede ser optimizado, resultando en agentes más rápidos y más eficientes.

Todo lo anterior permite considerar esta propuesta de arquitectura de agente, como una de las más simples y comprensibles que se han desarrollado en el campo de la administración de redes utilizando el protocolo SNMP. Y sin lugar a dudas pudiera ser fácilmente implementada utilizando otros protocolos de administración de red, tales como ICMP. En el siguiente capítulo se explica como funciona el prototipo correspondiente a esta arquitectura propuesta, y el lenguaje en que se desarrolló.

5.2 PROPUESTA DE ADMINISTRADOR

El administrador es otro de los tres componentes importantes en la administración de redes. En esta sección se verá a detalle que es, como funciona y el esquema propuesto de un administrador.

Un administrador esta localizado en el Host o algún servidor dentro de la red. Su función principal es preguntar a los agentes para conocer cierta información requerida. Haciendo uso de este método de preguntas, los administradores están capacitados para administrar desde uno hasta varios cientos de agentes. El administrador debe preguntar periódicamente a cada agente que esta bajo su control la información que sea conveniente, lo cual toma tiempo. Ese tiempo puede ser desde un segundo hasta varios días, dependiendo de la misión de cada agente administrado y el nodo en que se encuentre.

Los puntos de análisis que se tocarán para describir todo lo que es un administrador, son los siguientes:

- funcionalidad,
- operaciones,
- desarrollo,
- servicios y
- arquitectura

Cada uno de los puntos mencionados anteriormente, complementa a los otros en la estructura general, para conocer el esquema de un administrador de red. La meta aquí es poder construir un administrador que requiera la mínima entrada de datos del usuario, y que éste sea simple y fácil.

5.2.1 FUNCIONALIDAD

Un administrador debe ser capaz de responder a los requerimientos de otros administradores, de cualquier tipo de agentes y de los objetos administrados en cada agente.

5.2.2 OPERACIONES

Las operaciones soportadas por el administrador son las mismas operaciones que tiene ya definidas el protocolo SNMP. Ya que se vale de éstas, para generar todas las preguntas a los agentes y recibir de ellos las respuestas, así como las trampas generadas.

5.2.3 DESARROLLO

Hay decisiones que deben ser tomadas en el desarrollo de un administrador, éstas son las siguientes:

- Servicios a ofrecer e
- Interacción con el ambiente (recursos requeridos de hardware o software, la interfaz de usuario, entre otros.)

Parte de las razones para la dificultad en el desarrollo de un administrador, es que al diseñarlo no se tiene realmente documentación o información al respecto de éste. En otras palabras, el administrador es abstracto con relación al protocolo utilizado en el mundo de la administración de redes. Sin embargo para proponer un administrador, hay que considerar un conjunto de servicios que sea requerido por los agentes.

5.2.4 SERVICIOS

El conjunto de servicios que es requerido a un administrador por todos o la gran mayoría de los agentes, son los siguientes:

- Generar acciones de monitoreo y requerimientos de control
- Interpretar y registrar estos requerimientos
- Registrar y almacenar las respuestas de los requerimientos
- Proporcionar respuestas a las notificaciones de eventos predeterminados (trampas), enviados por los agentes y
- Comunicarse con otros administradores.

5.2.5 ARQUITECTURA

En esta sección, se describe cada uno de los componentes de la arquitectura propuesta, así como la información necesaria para automatizar la creación de cada componente. Los componentes de la arquitectura propuesta de un administrador simple se muestran en la figura 5-2.

Cuando se genera un requerimiento en el administrador, lo hace por el componente de comunicaciones y lo envía. Al recibir la respuesta a ese requerimiento, lo hace también con este componente. Enseguida la respuesta que llega se chequea en el componente verificador de respuestas y trampas, para conocer su contenido. En caso de que exista algún error en la respuesta, entonces el componente manejador de error valida el o los errores,

para indicarle a los componentes de comunicaciones y verificador de respuestas, que generen la solicitud de reenvío de la respuesta o la acción que corresponda.

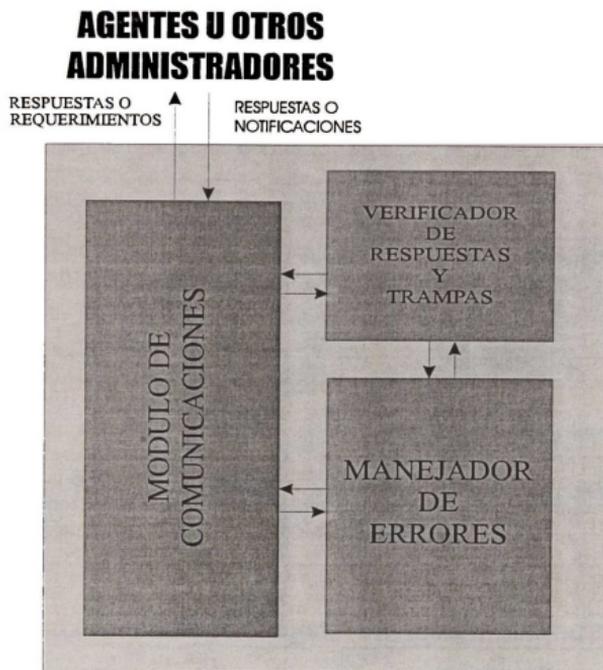


Figura. 5-2. Arquitectura propuesta de un administrador

COMPONENTE DE COMUNICACIONES

Este componente es el principal del administrador. Ordena las respuestas y pasa la información obtenida al componente apropiado, está estrechamente ligado con el protocolo de administración SNMP para todo lo que este componente necesita generar y conocer, proporciona los servicios de comunicaciones a otros componentes, tal como el servicio para permitir al administrador recibir respuestas, en el caso de recibir una notificación, retornar la información de respuesta al agente que la generó. Una característica de este componente, es que puede conocer cuales servicios proporcionan cada uno de los agentes que interactúan con el administrador.

COMPONENTE VERIFICADOR DE RESPUESTAS

Este componente verifica cada requerimiento que se envía o cada respuesta que se recibe para estar seguro que las siguientes condiciones se cumplen:

- El Administrador genera adecuadamente el requerimiento o notificación
- El agente que esta siendo encuestado soporta el requerimiento
- Los objetos administrados indicados en los requerimientos, sean efectivamente los validos.

Para verificar estas condiciones, este componente debe tener conocimiento de que servicios son soportados por cada agente que el administrador cuestiona.

COMPONENTE MANEJADOR DE ERROR

Este componente al igual que en los agentes, oculta algunos de los detalles del protocolo SNMP del resto del administrador. Traduce códigos internos de error a un formato idéntico al del protocolo SNMP, cuando se trata de una notificación. La cual puede entonces ser diagnosticada y posiblemente corregir los problemas que informa. Apoya a los componentes de comunicaciones y verificador de respuestas

5.2.6 CONCLUSIONES DE LA PROPUESTA DE ADMINISTRADOR

Una vez vistos los diferentes componentes propuestos de un administrador, se puede apreciar la misma ventaja de tener componentes modulares: incrementa la formalidad. Si este código es probado y se encuentra que es estable en una implementación particular, se puede asumir que el código trabajará en otras implementaciones, los detalles de implementación de otros componentes no afectan al código. Una vez que el código es estable, este además puede ser optimizado, resultando en administradores cada vez más completos, seguros, rápidos y eficientes.

Todo lo anterior permite considerar esta propuesta de arquitectura de administrador, como una de las más simples, transportable y comprensible que se ha desarrollado en el campo de la administración de redes utilizando el protocolo SNMP. En el siguiente capítulo se explica como funciona el prototipo correspondiente a esta arquitectura propuesta, y todos los términos en que se desarrolló. por ejemplo, el lenguaje utilizado.

CAPITULO 6

PROTOTIPOS

Anteriormente se presento una propuesta de arquitectura para un agente genérico, y una para un administrador. En este capítulo se describe cómo se desarrolló todo lo relacionado a los prototipos. A continuación se listan las partes consideradas para describir el trabajo realizado:

- Lenguaje de programación seleccionado
- Estructura de clases.
- Desarrollo del prototipo del administrador
- Desarrollo del prototipo del agente
- Códigos Fuente.

Estas partes describen en su conjunto, el trabajo realizado en los prototipos y permiten que sea más comprensible cada parte de los mismos.

6.1 LENGUAJE DE PROGRAMACION

El asunto de seleccionar un lenguaje de programación para desarrollar los prototipos, requirió que primero se pensará en definir el paradigma de programación a utilizar. Esto es, programación estructurada o programación orientada a objetos. Esta selección del paradigma surge debido a que muchos estudiosos del protocolo SNMP aseguran que éste no tiene bien definidos sus componentes como objetos y todo lo que los rodea.

Sin embargo, otros muchos estudiosos del protocolo SNMP aseguran que éste si tiene bien definidos sus componente como objetos. Esto es, que se puede representar con el paradigma de la orientación a objetos. Entrando a detalle en el tema se descubre que la polémica existe por que, en los principios del protocolo no era posible, verlo compuesto por objetos. Pero la misma evolución del protocolo ahora permite que si haya una utilización orientada a objetos del protocolo SNMP. Es muy importante recordar que este protocolo evoluciona a gran velocidad, ya que recibe aportaciones de mejora de todas partes del mundo.

Toda esta investigación permitió seleccionar el paradigma orientado a objetos, al cuál es considerado más útil para desarrollar aplicaciones genéricas y fácilmente transportables de una plataforma a otra. Una vez con el paradigma seleccionado, se analizaron los lenguajes de éste para seleccionar a los más comunes y comerciales. Los lenguajes que cumplieron con las características antes mencionadas son C++ y JAVA. Ahora la tarea era analizar a estos dos lenguajes con base en los siguientes aspectos:

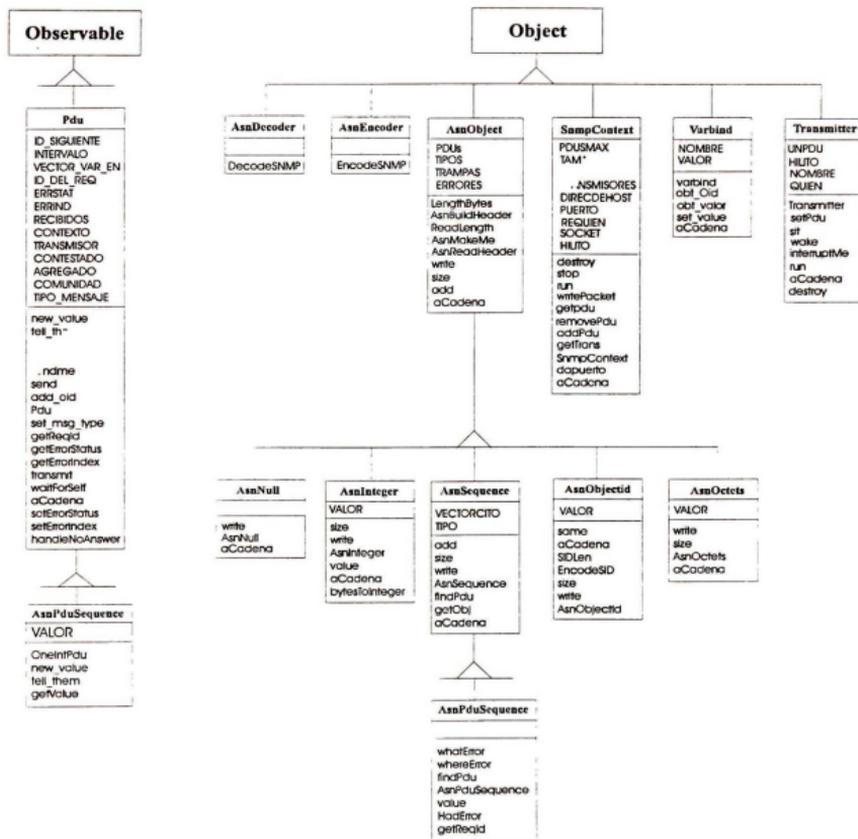
- *Sencillez*: Simplicidad para desarrollar aplicaciones con el lenguaje. Donde se aprecia que JAVA es más elegante y de más alto nivel en general.
- *Capacidades de red*: Biblioteca de funciones o métodos para utilizar más rápida y eficientemente la red. Donde se aprecia una fuerte semejanza.
- *Capacidades gráficas*: Biblioteca de funciones o métodos para desarrollar más rápida y eficientemente las interfaces gráficas en un sistema. Donde se aprecia una ventaja significativa de JAVA.
- *Costo*: Adquirir el compilador del lenguaje y la respectiva bibliografía. El compilador de JAVA es gratis y multiplataformas, lo cual aventaja al de C++. En cuanto a la bibliografía hay una fuerte semejanza en costos.
- *Documentación*: Cantidad de libros, artículos y manuales que existen en el mercado, así como, el grado que tiene de complejidad. En este caso, C++ cuenta con mejor y más documentación.
- *Presencia a futuro*: Posicionamiento del lenguaje en el mercado, lo cual ayuda a garantizar la vigencia de los sistemas desarrollados. En este caso, los dos lenguajes cada día están más fuertes, ya que son fuertemente respaldados por el mercado del cómputo.
- *Grado de experiencia*: Nivel actual de conocimiento del lenguaje.

Analizando a los lenguajes en cada uno de los aspectos antes mencionados excepto el último, se seleccionó al lenguaje JAVA para desarrollar los prototipos. Y excepto el último porque no se tenía ningún conocimiento de JAVA, pero existía la inquietud de aprender que ofrecía realmente como lenguaje, más haya de las aplicaciones para Web.

6.2 ESTRUCTURA DE CLASES

En esta parte del desarrollo de los prototipos, se tomaron como base los capítulos 2, 3 y 4 para estructurar las clases que se utilizaron en éstos. En otras palabras, el grupo de clases se construyó según los tipos de datos, secuencias y cómo se manejan todos estos.

A continuación se muestra la herencia de clases y se explica la finalidad de cada una de ellas.



Object: Clase propia del lenguaje Java, de la cuál se hereda.

AsnDecoder: Decodifica el flujo de entrada y obtiene de éste el PDU que está contenido.

AsnEncoder: Codifica paquetes SNMP y los envía por el flujo de salida.

AsnObject: Define y estructura todos los objetos según la notación ASN.1, así como leer y crear las secuencias y objetos que están contenidos en éstas. Es sin duda una de las clases más importantes para generar el prototipo.

- SnmpContext*: Crea el contexto para enviar y recibir paquetes SNMP de información, entre el agente y el administrador.
- varbind*: Crea variables enlazadas que contienen el identificador de objeto con su respectivo valor.
- Transmitter*: Crea, destruye y manipula los hilos asociados a las PDUs.
- AsnNull*: Crea objetos de tipo nulo, así como asigna el valor nulo a los objetos cuando es solicitado.
- AsnInteger*: Crea objetos de tipo entero, manipula su tamaño, y proporciona el valor de éstos.
- AsnSequence*: Construye y manipula la secuencia de información de la PDU.
- AsnObjectId*: Crea y manipula el identificador de cada objeto
- AsnOctets*: Crea, manipula y convierte las cadenas de octetos.
- AsnPduSequence*: Manipula la secuencia de cada elemento que conforma la PDU. Esta clase trabaja muy estrechamente con la clase *AsnSequence*.
- Observable*: Clase propia del lenguaje Java, de la cuál se hereda.
- Pdu*: Define, crea, estructura y manipula las PDUs independientemente de su contenido. Es sin duda una de las clases más importantes para generar el prototipo.
- OneIntPdu*: Construye una PDU que permite consultar información de tipo entero a los agentes.

6.3 PROTOTIPO DEL ADMINISTRADOR

El prototipo del administrador utiliza todo el conjunto de clases mostrado anteriormente, por lo cual no se detalla más al respecto.

El prototipo del administrador funciona de la siguiente manera: al arrancar el programa pide el número de puerto del agente a cuestionar, el nombre del nodo de red donde se encuentra el agente, el nombre de la comunidad que va a identificar la información que el administrador y el agente se enviarán, y por último el identificador del objeto (OID) a requerir.

Una vez que se capturó toda la información inicial antes mencionada, se crea el contexto utilizando como parámetros el nombre del nodo y el número de puerto que está

usando el agente. Aquí se genera el socket datagrama que se utilizará y un hilo para que se tenga autonomía y concurrencia en las operaciones.

Esta operación estará repitiéndose indefinidamente, y dentro de ésta, se realiza lo siguiente: Se define y se crea el paquete de datagrama para recibir por el socket los paquetes que lleguen. Aquí cabe mencionar que si el socket no recibe ningún paquete, el hilo(proceso) actual permanecerá bloqueado hasta que se reciba.

Como en este momento ya se construyó el contexto para establecer comunicación con el agente y no se pueden tener respuestas cuando no ha habido requerimientos, entonces el administrador continuará su ejecución y comenzará a generar requerimientos haciendo uso del hilo principal.

Para generar los requerimientos, el administrador necesita saber qué tipo de requerimiento debe realizar. Esto es, si realizará una operación *Get*, *Get-Next* o *Set*. Y otro aspecto importante es que, también necesita saber sobre que tipo de dato ya sea INTEGER, OCTET-STRING, COUNTER, etc. Se realizará la operación previamente solicitada.

Una vez que se determina que tipo de dato se va a requerir y que tipo de requerimiento se va a realizar, se invoca a la clase correspondiente pasando como parámetros, al contexto que se genero anteriormente, la comunidad que se está empleando para identificar la comunicación entre el agente y el servidor, el identificador del objeto que se va a requerir, y el identificador correspondiente al administrador.

Dentro de la clase correspondiente, se crea un nuevo paquete de datos de la PDU. Al crear esta nueva PDU, primero se crea un nuevo hilo que corresponderá a esta nueva PDU, con la finalidad de que exista la concurrencia necesaria entre las diferentes PDU's que esta manejando el administrador. Una vez creado el hilo, se sincroniza con los otros que ya existen. Con todo lo anterior, a creado en estructura el PDU, se procede a enviarlo.

Para enviar la PDU primero se codifican los campos correspondientes a la PDU, como se vió en el capítulo 5. Ya construida la PDU, se envía (escribe) por el flujo de salida que será convertido y enviado por el paquete datagrama creado para este fin. Aquí es importante mencionar que al crear el paquete datagrama, se le proporciona el nombre del nodo donde se encuentra el agente y el número de puerto que está utilizando. Para SNMP el número de puerto por default es el 161.

Finalmente el paquete datagrama es enviado por el socket y sólo queda esperar que llegue la respuesta. Mientras que todo esto sucede, los hilos ya existentes se conmutan entre sí para evitar algún posible problema de inanición entre ellos. Cuando llega un paquete, el hilo que se quedó bloqueado, esperando la recepción de paquetes en el socket, se reactiva para recibirlo. El paquete se convierte en un flujo de entrada y es decodificado. En el proceso para decodificar, el flujo será descompuesto en las secuencias que se vieron en el capítulo 5. Esto es, se empezará por leer su cabecera, conociendo el tipo y la longitud de la secuencia que lo forma. A continuación, este flujo es procesado según la clase correspondiente a cada tipo, lo cual nos permite conocer todos y cada uno de los elementos que conforman la secuencia (PDU) de entrada y sus respectivos valores, entre los cuales al

final de la secuencia está el identificador (OID) del objeto, del cual se solicitó conocer su valor actual.

Finalmente el administrador puede seguir enviando y recibiendo información solicitada al agente.

6.4 PROTOTIPO DEL AGENTE

Al igual que el prototipo del administrador, el prototipo del agente utiliza todo el conjunto de clases mostrado anteriormente, por lo cual no se detallará más al respecto.

El prototipo del agente funciona de la siguiente manera: al arrancar el programa selecciona el número de puerto a utilizar, por default SNMP utiliza el 161, y el nombre de la MIB que utilizará para responder a los requerimientos del administrador. Una vez que se capturó toda la información inicial antes mencionada, se crea el contexto. Aquí se genera el socket datagrama que se utilizará y un hilo para que se tenga autonomía y concurrencia en las operaciones.

Esta operación estará repitiéndose indefinidamente, y dentro de éste, se realiza lo siguiente: Se crea el paquete de datagrama para poder recibir por el socket los paquetes que lleguen. Aquí cabe mencionar que si el socket no recibe ningún paquete, este hilo permanecerá bloqueado hasta que reciba alguno.

Cuando llegue un paquete, se realizará exactamente la misma decodificación que se explico para este fin en el prototipo del administrador. Con la principal diferencia que, ahora el paquete solicita o informa al agente acerca del valor correspondiente a un OID.

Como ya existe un requerimiento o una inicialización, el agente lo utiliza para conocer el nombre y la dirección del nodo del administrador y el número del puerto que está utilizando. El agente continúa su ejecución y comienza a generar la respuesta correspondiente, haciendo uso del hilo principal. Para generar la respuesta, el agente necesita saber que tipo de requerimiento se le hizo, esto es, saber que tipo de dato, ya sea INTEGER, OCTET-STRING, COUNTER, etc., fue el que definió el requerimiento.

Una vez que se determina qué tipo de dato se le solicitó, se invoca al método de la clase correspondiente, pasando como parámetros al contexto que se generó anteriormente, la comunidad que se está empleando para identificar la comunicación entre el agente y el administrador, el identificador del objeto que se está requiriendo, y el identificador correspondiente al agente. Dentro de la clase correspondiente se crea un nuevo paquete de datos(PDU). Al crear esta PDU de respuesta, primero se crea un nuevo hilo que corresponderá a toda la construcción y envío de ésta, con la finalidad de que exista la concurrencia necesaria entre los diferentes hilos que están conformando al agente. Una vez creado el hilo, se sincroniza con los otros que ya existen.

Para enviar el PDU de respuesta, primero se codifica con los campos correspondientes al PDU como se vió en el capítulo 5. Ya construido el PDU respuesta, se escribe en el flujo de salida, el cuál será enviado al paquete datagrama creado para este fin. Aquí es importante mencionar que al crear el paquete, se le proporcionan el nombre del nodo donde se encuentra el administrador y el número de puerto que está utilizando, lo cual fue obtenido con anterioridad del flujo de entrada. Para el administrador no hay un número de puerto por default como en el caso del agente, por lo que cada vez que inicia debe obtener su puerto.

Finalmente el paquete es enviado por el socket y sólo queda esperar que llegue éste al administrador. Al terminar el proceso de recepción y envío, el agente puede seguir esperando requerimientos con información solicitada por el administrador.

6.5 CODIGOS FUENTE

Para desarrollar el código fuente utilizando el lenguaje Java, primero se debía diseñar el grupo de clases que representaran la codificación de sintaxis abstracta uno (ASN.1) y segundo diseñar el grupo de clases que representaran la codificación del protocolo SNMP. Para finalmente diseñar las clases para los prototipos de administrador y de agente, los cuales utilizaran los dos grupos antes mencionados.

Desarrollar estos grupos es una tarea compleja y que lleva tiempo, por lo que primeramente se realizó una búsqueda en diferentes medios, para saber si existía algún desarrollo al respecto. Encontrándose dos soluciones particulares desarrolladas en Java, de las cuales una codificaba la ASN.1 y la otra codificaba al protocolo SNMP, sin embargo las dos son soluciones particulares parcialmente desarrolladas y no van más allá de eso. Al platicar con los autores, ambos otorgaron su permiso para utilizar y modificar su código, con la única condición de mencionarlos en el producto final que se desarrollara. Ya con el permiso, hay que mencionar que la tarea de analizar, comprender, probar y modificar estas soluciones particulares para integrarlas, fue una tarea ardua y de mucho tiempo, ya que en ambos casos no existía o no se nos proporcionó ninguna documentación al respecto. Pero apegándonos a la notación ASN.1, al protocolo y a todas sus características se logró comprender que hace y para que sirve cada clase de estas soluciones particulares y utilizar lo que nos servía para codificar nuestras propuestas. Los autores de los dos grupos de clases que tomamos como base son Tim Panton y Jordan Hargrave, a los cuales se les agradece la aportación de su código y sus pocas pero al fin útiles asesorías, ya que gracias a sus proyectos y al trabajo que se ha realizado integrándolos, se han fusionado realmente tres investigaciones para construir los prototipos de las propuestas de este trabajo.

Cómo ya se vió en la sección 6.2 de este capítulo, los prototipos de nuestra propuesta utilizan 18 clases diferentes, incluyendo la del administrador y del agente. En su conjunto, estas clases tienen más de 2600 líneas de código y para hacer uso de ellas en su totalidad, el usuario sólo necesita proporcionar tres datos de entrada al prototipo de administrador y un dato al prototipo de agente, lo cuál habla de la simplicidad de las propuestas.

Una vez terminados los prototipos había que probarlos para saber si efectivamente se cumplía la finalidad de éstos, que era integrarlos a un sistema de administración de redes de uno o varios fabricantes. Después de algunas correcciones, el primero en funcionar fue el prototipo de administrador que se probó administrando un agente comercial en un servidor Windows NT 4.0, después administrando un agente comercial en un servidor de impresión "Axis" y finalmente, un agente comercial en una estación de trabajo SparcStation de SUN. Y estas mismas pruebas se hicieron con dos administradores comerciales para validar los resultados. Después funcionó el prototipo del agente, el cual se probó corriendo en una computadora con procesador Intel y después en una computadora Macintosh primero respondiendo a nuestro prototipo de administrador, lo anterior se representa en la figura 6-1. Lo cual nos da como resultado, que los dos prototipos codifican y decodifican apegados a la notación ASN.1 y al protocolo, lo que les da una integración total con cualquier producto comercial para la administración de redes.

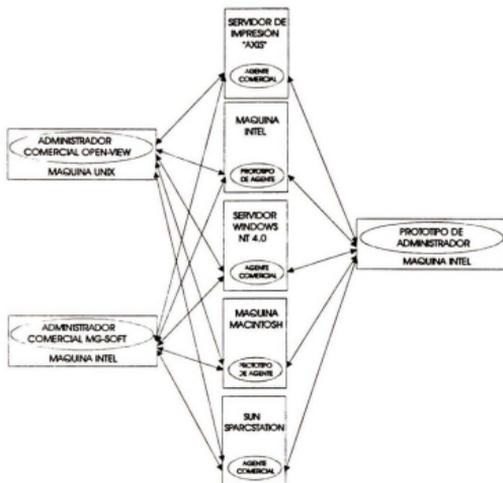


FIGURA 6-1. PRUEBAS REALIZADAS ENTRE LOS PROTOTIPOS Y PRODUCTOS COMERCIALES.

Para finalizar este capítulo, todos los códigos fuente de los prototipos, se muestran en el apéndice C. El orden de aparición es exactamente el mismo, que como fueron descritos anteriormente en este capítulo.

CONCLUSIONES

CONCLUSIONES

En el presente trabajo se seleccionó el estándar de Internet para la administración de redes que es el más utilizado en el mundo, lo cuál nos dio un amplio campo de aplicación para desarrollar nuestra propuesta. También, se utilizó la notación ASN.1 y las reglas básicas de codificación para describir, nombrar, codificar y conocer los valores de los objetos que fueron administrados. Por otro lado, se conocieron las PDUs del protocolo SNMP y los mecanismos para que nuestro prototipo de administrador y de agente, se comuniquen con otros componentes para la administración de red satisfactoriamente. Se presentaron y se describieron las arquitecturas de las propuestas de administrador y de agente a detalle, de las cuales se desarrollaron las clases de los prototipos y la codificación en el lenguaje Java. Finalmente se realizaron las pruebas de nuestros prototipos, integrándolos con diferentes administradores y agentes comerciales de diferentes plataformas y dispositivos, lo que resultó satisfactorio puesto que, hay una comunicación transparente entre todos éstos y no se tienen limitaciones en el uso de diferentes plataformas.

INTERNET es el estándar más utilizado en el mundo en redes de datos y SNMP es el protocolo asociado a éste, razón por la que se desarrolló la propuesta bajo este estándar. En lo personal creo que es algo complicado inicialmente comprender y utilizar SNMP. Una vez que se logran representar todos los datos y tipos del protocolo SNMP adecuadamente, lo cual toma tiempo, difícilmente pueden existir errores en la construcción de los mensajes de información, son como una plantilla.

La estructura de la información administrada es sin duda una parte fundamental y necesaria para cualquier investigación o desarrollado en el área de sistemas para la administración de red. La complejidad para estructurar información administrada es alta y sobre todo se debe manejar con mucho cuidado, de lo contrario, afecta todo el proceso de administración y éste no funciona.

La base de datos de la información administrada, tiene una relación estrecha con la SMI. Esta base de datos organiza jerárquicamente los objetos administrados, y como los objetos son definidos en su totalidad a través de macros especificadas, utilizando la notación ASN.1 propia para la administración de redes. Esta base de datos es muy grande y específica para la administración de redes, hay que seguir las normas ya establecidas en los RFCs para modificarla. Sin embargo en el árbol jerárquico de Internet, se pueden agregar objetos de prueba dentro de su rama de investigación y posteriormente, si este objeto es útil, se integra a un grupo de objetos de la MIB.

Se constató que la administración de redes, efectivamente es una rama desarrollada por muy poca gente en el mundo. Lo anterior se basa en que, sólo existen ciertos grupos de desarrollo y sus trabajos son dirigidos a partes específicas de la administración de redes. Aun más selecto es el grupo de organizaciones o personas que utilizan Java para desarrollar las pruebas de sus investigaciones, o sus productos comerciales, sin embargo cada día este número aumenta.

Las propuestas de agente y de administrador, son más simples en su funcionamiento que algunos de los productos comerciales o que los desarrollos hechos por investigadores, ya que requieren un mínimo de datos de entrada y realizan todos sus procesos independientemente del humano. Se demostró que las propuestas desarrolladas y los prototipos correspondientes, se integran a sistemas para la administración de redes comerciales funcionando de manera adecuada. Además, se pueden ejecutar en múltiples plataformas sin agregar o eliminar líneas de código, gracias a la máquina virtual de Java que es la que traduce el código fuente en instrucciones propias de cada plataforma.

Gracias a la simplicidad y capacidad mostrada por las propuestas y sus prototipos, este trabajo ha sido sometido a evaluación en dos congresos internacionales [C-1] [C-3], dos congresos nacionales [C-2] [C-4] y ha asistido a varias invitaciones para ser mostrado, durante semanas de la informática en diversas instituciones de educación superior.

Las perspectivas a futuro del presente trabajo se mencionan en el orden de prioridad que se considera es el ideal.

- Primera, profundizar y complementar el planteamiento de las propuestas y sus arquitecturas, para desarrollar a un mediano plazo una metodología para la construcción de componentes para la administración de redes. Y proporcione las capacidades de construir aplicaciones para la administración de redes fácilmente.
- Segunda, que los dos prototipos tengan la capacidad de manejar, la totalidad de los tipos de datos y de mensajes que soporta el protocolo SNMP. Lo cuál es relativamente simple, ya que ahora se tiene la infraestructura base para hacerlo.
- Tercera, que se desarrolle e integre a los prototipos de administrador y de agente un compilador de MIBs, para que al ejecutarse o activarse, puedan manejar la información administrada recibida o solicitada según el caso.

- Cuarta, desarrollar e integrar al prototipo del agente, el componente manejador de objetos administrados, el cual debe trabajar independientemente del protocolo SNMP y tener la capacidad para soportar una gran variedad de objetos.
- Y por último, el desarrollo de la interfaz gráfica para cada uno de los prototipos. La cuál al igual que éstos, se propone sea desarrollada en el lenguaje Java. Ya que las capacidades para desarrollo gráfico de Java son muy grandes y relativamente simples.

APENDICE A

JAVA

En este apéndice se verá que es el lenguaje de programación Java. Y comienza con la pregunta necesaria.

A.1 ¿QUE ES JAVA?

Algunas personas ven a Java sólo como una moda mas de INTERNET. Otros lo ven como el futuro de INTERNET y todas las aplicaciones que lo rodean. Java es un nuevo lenguaje de programación orientado a objetos. Cuenta con las ventajas de la programación orientada a objetos y varias más que se han producido durante su proceso de desarrollo.

Sin duda alguna, Java esta cambiando la manera en que la gente y personas de todas las áreas de la computación utilizan el INTERNET, las redes y las aplicaciones en red. Java emplea muchos elementos comunes de otros lenguajes orientados a objetos, como C++, pero contiene mejoras que facilitan la programación.

Al igual que cualquier otro lenguaje, Java cuenta con una sintaxis particular, una estructura para programas y muchas aplicaciones de soporte.

El Java Developer's Kit (JDK) contiene todas las herramientas necesarias para crear aplicaciones o applets mediante el lenguaje de programación Java, incluyendo las siguientes:

- javac El compilador de Java
- jdb El depurador de Java
- javadoc El programa de documentación de Java
- java La Máquina Virtual de Java (JVM)
- appletviewer El visor de applets de Java

Tal vez algunos de los componentes de Java no parezcan muy obvios, como la Máquina Virtual de Java. Al usar Java para programar, uno no accesa la Máquina Virtual de Java directamente.

A.2 COMPUTACION DISTRIBUIDA

Java es revolucionario por varias razones, pero una de las más importantes es cómo cambia la manera en que usamos las computadoras. Java está diseñado alrededor del concepto de las redes y la conectividad. La finalidad de Java es producir aplicaciones muy completas para red y esta realidad se hace cada día más palpable, ya que la evolución en las restricciones de las redes van desapareciendo y el lenguaje Java puede mostrar su verdadero potencial en las redes. Todo lo anterior es totalmente independiente del gran impacto que Java a tenido en INTERNET con sus miniaplicaciones llamadas Applets, que pueden incorporarse dentro de sitios Web y ser ejecutadas desde una página base.

La programación multihilo es una de las características más poderosas del lenguaje Java. Ya que Java no deja fuera de su especificación el multihilo, lo que lo hace más simple de utilizar los hilos y su sincronización es la más elegante hasta ahora.

A.3 SIMPLICIDAD

Java se basa en su mayor parte en C++, por desgracia, C++ es un lenguaje complejo. Por ejemplo, C++ le da al programador control directo sobre la administración de memoria de una aplicación. Esta capacidad significa que el programador tiene que ser muy cuidadoso acerca de la asignación de nueva memoria a una aplicación, estableciendo la manera en que la aplicación empleará dicha memoria y limpiando la memoria no utilizada cuando la aplicación termina.

La asignación de memoria y la recolección de basura con frecuencia dan como resultado errores muy complicados y consumidores de tiempo en los programas de C++. Afortunadamente, Java maneja por el programador los detalles de la asignación de memoria, lo cual ahorra muchas horas de compleja depuración.

Este manejo automático de la memoria es una gran ventaja. Cualquiera que haya programado en C++ sabe que el aspecto más complicado del lenguaje son los apuntadores. Los apuntadores son utilizados en C++ para llevar cuentas de las ubicaciones de memoria y lo que se almacena en éstas. Y mientras más grandes son las aplicaciones en esa o mayor proporción crece la complejidad en el manejo de los apuntadores. Para facilitar la programación, Java esta diseñado explícitamente para eliminar los apuntadores.

A.4 DESVENTAJAS

El lenguaje Java presenta dos desventajas principales con respecto a los otros lenguajes orientados a objetos:

- La administración de memoria simplificada de Java provoca que algunas de las aplicaciones sean más lentas que si se desarrollaran en C++. Lo cual deja en desventaja temporal al Java con respecto al C++.
- Se debe planear adecuadamente su aplicación antes de codificarla, así la orientación a objetos trabajará para el desarrollador, no en su contra. En cuanto comience a desarrollar mas aplicaciones y a reutilizar objetos, se observa claramente que la planeación le brinda beneficios y a la larga le ahorra tiempo.

Pero aún con estas desventajas, Java es una herramienta maravillosa. Ya que desde el principio Java fue diseñado para corregir algunos problemas de los lenguajes de programación de alto nivel.

A.5 INDEPENDIENTE DE LA PLATAFORMA

El código Java está diseñado para ser independiente de la plataforma y esto se logra por que cuenta con varias características. Cuando se escribe una aplicación en Java, se está escribiendo un programa para ejecutarse en la Máquina Virtual de Java. Esta Máquina Virtual es la que permite ser independiente de la plataforma.

La Máquina Virtual es otra pieza de software. Entonces la Máquina Virtual interpreta y ejecuta el programa de Java que se ha escrito. Existe una Máquina Virtual para cada uno de los sistemas operativos de computadoras CISC y RISC. (Win95, WinNT, OS/2, Macintosh, Solaris, HP-UX, entre otros), además existen máquinas Virtuales para los navegadores de INTERNET como Netscape y Explorer.

Con esta ventaja, cualquier programa escrito en código Java, puede ejecutarse en cualquier computadora que ejecute una versión de la Máquina Virtual. Lo anterior sin necesidad de modificar una sola línea de código. Cuando se compila un programa de Java el compilador genera el código de byte (bytecode), independiente de la plataforma. Entonces, cualquier Máquina Virtual Java puede ejecutar dicho código de byte.

A.6 APARIENCIA CONSISTENTE

La Máquina Virtual también le permite a Java conservar una apariencia y manejo consistentes con la plataforma. Por ejemplo, cuando se ejecuta una aplicación en una máquina de Windows, los menús y las barras de herramientas de su aplicación Java se ven como componentes estándar de Windows. Si se ejecuta en un sistema Sun, los botones, los menús, entre otros. Tendrán la apariencia y la funcionalidad del sistema Sun, y así sucesivamente.

APENDICE B

PRODUCTOS COMERCIALES

En la actualidad las grandes empresas mundiales de las redes y conectividad cuentan con sus respectivos productos para la administración de redes. Sin embargo, esto no es del todo satisfactorio para las empresas y los usuarios de sus productos.

Las mismas empresas reconocen, que la administración de redes es compleja y costosa. Que actualmente sus productos necesitan integrarse manualmente entre ellos, o en definitiva, desarrollar una aplicación de administración de redes específica para una empresa en particular o para un hardware en particular.

A continuación se mencionarán y se describirán brevemente los productos para la administración de red más importantes.

Solstice de Sun Microsystems: El Solstice es un sistema diseñado para ayudar a administrar su entorno de red optimizando los costos de administración, y realiza administración centralizada para redes de mas de 100 nodos. Este sistema se enfoca en poder administrar redes de diferentes tamaños. Plataformas pequeñas pueden administrar sitios pequeños a un costo apropiado, pero no escalar a administrar los ambientes grandes.

Solstice cuenta con aplicaciones de usuario, agentes que ayudan para una administración automatizada y una interface programática que permite a los desarrolladores crear nuevas aplicaciones y agentes para la administración.

Algunas de las herramientas con que cuenta el Solstice son las siguientes:

- Solstice Site Manager
- Solstice SunNet Manager
- Solstice Domain Manager
- Cooperative Consoles
- Enterprise Manager
- Enterprise Agents
- FireWall-1

Hasta el momento, solo existe la versión para el sistema operativo Solaris, en computadoras SPARC. Lo cual es una limitante y por que no desirlo, una desventaja con los otros productos.

OpenView de Hewllet Packard: El HP OpenView ayuda a los administradores a tomar control de sus redes, simplificando las tareas y reduciendo los costos de la computación cliente / servidor.

El OpenView integra trece aplicaciones de administración, que se utilizan para computadoras personales, servidores, dispositivos de red, impresoras y UPSs. Proporciona

una caja de soluciones con una sola interface de administración, la cual descubre, mapea, configura y patea la red. Al instalarse realiza todo lo anterior en un máximo de 30 minutos.

La capacitación y entrenamiento del personal es mínima, gracias a su estructura e interface consistentes. Y por último el tiempo empleado para diagnosticar y corregir problemas es muy poco.

Algunas de las herramientas con que cuenta el OpenView son las siguientes:

- Workgroup Node Manager
- Desktop management tools
- Profesional suite server management
- Traffic and Network Health Monitoring
- HP Network Device management
- HP Printer Management
- Remote Management
- Uninterruptible Power Supply Management

De los productos en el mercado, el OpenView es uno de los líderes y sin lugar a dudas uno de los sistemas administradores de red, con mejores capacidades para integrar a su sistema aplicaciones de otros fabricantes.

SystemView de IBM: El SystemView es un sistema integrado que puede controlar ambientes de red multiproveedores, (AIX, HP-UX, OS/2, DOS, SunOS, y Windows) y multiprotocolos. Optimizando costos y ofreciendo un nivel de productividad, eficiencia, y escalabilidad sin precedente.

SystemView esta constituido por los siguientes elementos:

- Performance Management
- Operations, Print Management
- Configuration Management
- problem Management
- Network Management
- Storage Manager
- Network management Applications

Zero de McAfee: El Zero es una aplicación líder en el mercado de la administración de redes. La cual realiza las siguientes funciones:

- Bloqueo de escritorio integrado
- Configuración centralizada
- Administración de escritorio

- Inventario de Hardware
- Distribución de Software
- menús y registros
- Control remoto de escritorio
- Herramienta para el análisis de costos.

Además cuenta con herramientas que automatizan las funciones críticas y acelera el trabajo de recorrer la red, disminuyendo los costos. La consola central de Zero controla computadoras personales con DOS, Windows 3.1, Windows 95 y Windows NT Workstation, conectadas a un servidor de archivos Netware o Windows NT. Y esto asegura la accesibilidad a las estructuras de bases de datos.

Zero cuenta con soporte nativo para Netware y Windows NT Server, lo cual le da transparencia y costos bajos, mientras hace más poderosos a los administradores, dándoles el control de toda su red.

ManageWise de Novell: El ManageWise es un conjunto integrado de servicios de administración que permiten al administrador de red en general monitorear y controlar su ambiente heterogéneo de redes. ManageWise permite administrar fácilmente los servidores Netware, los servicios de impresión y los dispositivos SNMP. Adicionalmente puede administrar servidores Windows NT con la adición del ManageWise Agent for Windows NT Server.

Se puede analizar tráfico en la red, controlar y administrar de manera remota las redes y las estaciones de trabajo, administrar aplicaciones de red, y prevenir infiltración de paquetería y virus. Todo desde una estación de trabajo.

ManageWise automáticamente mapea todos los dispositivos de red y proporciona un inventario detallado de cada estación de trabajo y servidor en la red. ManageWise soporta un amplio rango de estándares industriales y más de 100 aplicaciones de terceros disponibles en la industria.

ManageWise soporta los sistemas operativos DOS, Macintosh, Windows y OS/2. Y generalmente Novell recomienda la instalación de una copia del producto por servidor. Managewise se puede adquirir en versiones de 5 a 1000 usuarios, por lo que es una solución para redes de cualquier tamaño.

CiscoVision de Cisco Systems: El CiscoVision es una suite de herramientas integradas para la configuración y el diagnóstico de redes de pequeño a mediano tamaño, basadas en computadoras personales o grupos de trabajo remotos.

CiscoVision da a los usuarios todo lo que ellos necesitan para administrar ambientes de red. La disponibilidad de CiscoVision permite ahora a los usuarios obtener una solución para la administración de redes barata, con respecto a los otros productos similares.

La integración de CiscoVision con las otras plataformas existentes como HP OpenView esta mas que probada, y de forma fácil. Por lo que los administradores no tendrán que interrumpir sus sistemas de administración actual, para actualizarlo.

Las características de CiscoVision son las siguientes:

- Plataforma de administración SNMP completa
- Configuración de dispositivos integrada
- Monitoreo de dispositivos integrados
- Pila de enlace TCP/IP
- Soporte estándar para la protección de ambiente

Finalmente, se mencionan algunos de los beneficios que tiene CiscoVision. Estos son:

- Incluye todo lo que el cliente necesita para configurar, monitorear, y administrar los dispositivos Cisco en su red.
- Activa todos los enrutadores CiscoPro y switches de grupos de trabajo de la red para ser administrados por un solo sistema, ahorrando tiempo, dinero y esfuerzo.
- Se instala rápida y fácilmente utilizando las convenciones y los estándares de Windows.
- Fácil de usar con mínimo entrenamiento, gracias a su interfaz gráfica.

Por último es importante mencionar que en la actualidad todas estas empresas, ya están desarrollando sus nuevas aplicaciones para la administración de redes basadas en Java. Incluso algunas de éstas ya han liberado las primeras versiones de sus aplicaciones. Esta renovación de sus productos, dicho por las mismas empresas pretende mejorar las capacidades de los productos, principalmente la de integración entre ellos.

APENDICE C

CODIGO FUENTE

```

=====
// Codigo EXPERIMENTO 5.
// Relacionado con el desarrollo del administrador
// DOMINGO 21 de Septiembre de 1997
// Carlos Alberto Guizar Gomez
// *** ADMINISTRADOR QUE REALIZA SOLO UN PDU ***
=====

import snmp.stack.*;
import snmp.pdu.*;
import java.io.*;
import java.net.*;
import java.lang.*;
import java.util.*;

class admin5
{
    private static SnmpContext contexto;
    public static int leporte=0, c=0, puerto=0, con=0, selagen, seltipos;
    public static int version = 0, pdu = 0, requestID = 0, errStatus = 0, errIndex = 0;
    public static byte puer[]= new byte[4], nomage[] = new byte[1], tipos[]= new byte[1],
        comuni[] = new byte[6], posfijo[] = new byte[5];
    public static StringBuffer OID = new StringBuffer();
    public static String comunidad, maq, maqui, oid, eltipo,
        OIDFIJO = new String("1.3.6.1.2.1."); //ES EL OID FIJO + UN PREFIJO
    public static char buf3[] = new char[3]; //ESTO SE PUEDE HACER MAS FLEXIBLE

    /*****
    METODO: RECIBIR()
    OBJETIVO: CREA EL CONTEXTO Y ESPERA LA RESPUESTA DEL AGENTE AL
    REQUERIMIENTO
    PARAMETROS:NINGUNO
    *****/
    public static void recibir() throws Exception
    {
        System.out.println("/*/*/* CREANDO EL CONTEXTO Y LLENANDOLO /*/*/* ");
        try
        {
            contexto = new SnmpContext(maqui,puerto,13);
            leporte = contexto.dapuerto(); // OBTIENE EL PUERTO LOCAL DEL ADMIN
            System.out.println("EL PUERTO LOCAL DEL ADMINISTRADOR ES: "+leporte);
        } catch(java.io.IOException e)
        {
            System.out.println("EXCEPCION DE E/S AL CREAR EL CONTEXTO: "+
            e.getMessage());
        }
    }
}

```



```

public static void main(String args[]) throws Exception
{
    System.out.println("DAME EL NUMERO DE PUERTO DEL AGENTE: ");
    while((c = System.in.read())!='\n')
    {
        puer[con++] = (byte) c;
    }
    String l = new String(puer, 0);           //PUERTO EN CADENA
    if(l.charAt(0) == '0')
    {
        l.getChars(1, 4, bufs, 0);          //PARA QUITAR UN CERO A LA IZQ.
        String ll = String.valueOf(bufs, 0, 3);
        puerto = Integer.parseInt(ll);      //PUERTO EN ENTERO
        System.out.println("ENTERO DEL PUERTO: "+puerto);
    }
    else
    {
        puerto = Integer.parseInt(l);       //PUERTO EN ENTERO
        System.out.println("ENTERO DEL PUERTO: "+puerto);
    }
    /*puerto = 161;                        // SIMPLIFICACION ADMINISTRATIVA
    System.out.println("ENTERO DEL PUERTO: "+puerto); */

    c = 0; con = 0;                        // LIMPIA LAS VARIABLES PARA CACHAR CARACTERES.
    System.out.println("DAME EL NUMERO DEL AGENTE, 1) HUITZI, 2) PC15: ");
    while((c = System.in.read())!='\n')
    {
        nomage[con++] = (byte) c;
    }
    maq = new String(nomage, 0);
    selagen = Integer.parseInt(maq);        //cadena a entero
    if(selagen == 1)
    {
        maqui = new String("huitzilopochtli");
    }
    if(selagen == 2)
    {
        maqui = new String("PC15");
    }
    System.out.println("NOMBRE DEL AGENTE EN CADENA: "+maqui);

    c = 0; con = 0;
    System.out.println("DAME EL TIPO DE OID's A REQUERIR: (1)Enteros,
        (2)Cadenas, (3)MTiempo ");
    while((c = System.in.read())!='\n')
    {
        tipos[con++]=(byte)c;

```

```
}
eltipo = new String(tipos,0);
seltipos = Integer.parseInt(eltipo);           //cadena a entero
if(seltipos == 1)
{
    System.out.println("TIPO DE DATO A REQUERIR, SELECCIONADO ES:
                        ENTERO ");
}
if(seltipos == 2)
{
    System.out.println("TIPO DE DATO A REQUERIR, SELECCIONADO ES:
                        CADENAS ");
}
if(seltipos == 3)
{
    System.out.println("TIPO DE DATO A REQUERIR, SELECCIONADO ES:
                        MARCAS DE TIEMPO ");
}

/*c = 0; con = 0;    // LIMPIA LAS VARIABLES PARA CACHAR CARACTERES.
System.out.println("DAME EL POSFIJO DEL IDENTIFICADOR DE OBJETO: ");
while((c = System.in.read())!='\n')
{
    posfijo[con++] = (byte) c;
}
String oidpos = new String(posfijo, 0);       //OID POSFIJO EN CADENA
*/

String oidpos = new String("1.7.0");
OID.append(OIDFIJO); OID.append(oidpos);
oid = OID.toString();
System.out.println("IDENTIFICADOR DE OBJETO en string: "+oid);

recibir(); //CREA EL CONTEXTO Y ESPERA LA RESPUESTA AL
           REQUERIMIENTO
pedir();   // CONSTRUYE UN REQUERIMIENTO PARA EL AGENTE

} //FIN DEL MAIN

} // FIN DE LA CLASE ADMIN5
```

```

=====
// Codigo EXPERIMENTO 4.
// Relacionado con el desarrollo del AGENTE
// Viernes 26 de Septiembre de 1997
// Carlos Alberto Guizar Gómez
// ***** A G E N T E QUE RECIBE PDU CON UN OID *****
=====

import snmp.stack.*;
import snmp.pdu.*;
import java.io.*;
import java.net.*;
import java.lang.*;
import java.util.*;

public class agent4
{
    private static SnmpContext contexto;
    public static byte nomb[]= new byte[11];
    public static String oid, eltipo;
    public static int c=0, con=0;

    /**
     * METODO QUE RECIBE EL MENSAJE SNMP QUE EL ADMINISTRADOR ENVIO
     */
    public static void recibemen() throws Exception
    {
        System.out.println(" ");
        System.out.println("*. *+CREANDO EL CONTEXTO Y LLENANDOLO.+*.* ");
        try
        {
            //PUERTO=0 Y MAQ="" PORQUE NO SE CONOCE ESTA INFORMACION
            contexto = new SnmpContext("",0,6); //CREA EL CONTEXTO
        } catch(java.io.IOException e)
        {
            System.out.println("EXCEPCION DE E/S AL CREAR EL CONTEXTO: "+
            e.getMessage());
        }
    }

    } // FIN DE LA FUNCION RECIVEMEN

    /**
     * METODO QUE ENVIA MENSAJE SNMP QUE EL AGENTE RESPONDERA AL
     * ADMINISTRADOR
     */
    public static void enviamen() throws Exception
    {

```



```
//=====
// CLASE           : ASNDECODER
// AUTOR           : CARLOS GUIZAR
// VERSION ORIGINAL: TIM PANTON
//
//=====
```

```
package snmp.stack;
```

```
import java.io.*;
import java.util.*;
```

```
public class AsnDecoder extends Object
```

```
{
  /*****
  METODO: DECODESNMP()
  OBJETIVO: DECODIFICA EL FLUJO DE ENTRADA Y RETORNA UN PDU.
  INVOCADO: POR RUN() DE CONTEXT,
  *****/
  public AsnPduSequence DecodeSNMP(InputStream in, int requien) throws IOException
  {
    int valor = 0;

    AsnSequence dummy = new AsnSequence();
    //SE OBTIENEN LOS OBJETOS
    AsnObject asnTopSeq = dummy.AsnReadHeader(in, requien);
    AsnPduSequence Pdu = (AsnPduSequence) asnTopSeq.findPdu();

    return Pdu;
  }
} // FIN DE LA CLASE ASNDECODER
```

```
//=====
// CLASE           : ASNENCODER
// AUTOR           : CARLOS GUIZAR
// VERSION ORIGINAL: TIM PANTON
//
//=====
```

```
package snmp.stack;
```

```
import java.io.*;
import java.util.*;
```

```
public class AsnEncoder extends Object
```

```

{
/*****
METODO: ENCODESNMP()
OBJETIVO: CODIFICA PAQUETES SNMPv1.
INVOCADO: POR SENDME(comuni)
*****/
public ByteArrayOutputStream EncodeSNMP(Enumeration ve, String comunidad,
    int rId, int errstat, int errind, byte tipo_mensaje, int requien) throws IOException
{
    ByteArrayOutputStream bout;
    AsnObject      asnTopObject, asnPduObject, asnVbObject, asnVbObject;

    // CREA LA AUTENTIFICACION DEL PDU
    asnTopObject = new AsnSequence();
    asnTopObject.add(new AsnInteger(0));          // VERSION DE PDU
    asnTopObject.add(new AsnOctets(comunidad));   // COMUNIDAD

    // CREA LA SECUENCIA PDU EMPESANDO POR
    // TIPO DE REQUERIMIENTO
    asnPduObject = asnTopObject.add(new AsnSequence(tipo_mensaje)); //CONSTRUYE
    SECUENCIA SEGUN MENSAJE
    asnPduObject.add(new AsnInteger(rId));       // ID DEL REQUERIMIENTO
    asnPduObject.add(new AsnInteger(errstat));   // ESTADO DE ERROR
    asnPduObject.add(new AsnInteger(errind));    // INDICE DE ERROR

    // CREA LA SECUENCIA DE VarbindList
    asnVbObject = asnPduObject.add(new AsnSequence());

    // AGREGA LAS VARIABLES ENLAZADAS
    while(ve.hasMoreElements())                 // LOS OBJETOS QUE HALLA !!!
    {
        // SE AGREGAN AL PDU.
        asnVbObject = asnVbObject.add(new AsnSequence());
        varbind vb = (varbind) ve.nextElement();
        asnVbObject.add(vb.obt_Oid());           // OBJETO A CODIFICAR
        asnVbObject.add(vb.obt_valor());        // VALOR QUE TIENE EL OBJETO
    }

    // ESCRIBE EL OBJETO SNMP EN EL FLUJO
    bout = new ByteArrayOutputStream();
    asnTopObject.write(bout);
    System.out.println(" EN ASNENCODER, EL CONTENIDO EN EL BYTEARRAY
        OUTPUT ES: "+bout+" "+
            "^^ ^^ ^^ ^^ "+"||||||||||||||||||||||||||||||||||||||||");
    return bout; // AHI VA LO QUE RETORNA
}
} // FIN DE LA CLASE ASNENCODER

```

```
//=====
// CLASE           : ASNINTEGER
// AUTOR           : CARLOS GUIZAR
// VERSION ORIGINAL: TIM PANTON
//
//=====

package snmp.stack;

import java.io.*;
import java.util.*;

//=====
// ENTERO CON SIGNO ASN.1 DE 32 BITS
//=====
public class AsnInteger extends AsnObject
{
    protected int valor;

    /*****
    METODO: SIZE()
    OBJETIVO: OBTIENE EL NUMERO DE BYTES OCUPADOS POR EL ENTERO.
    INVOCADO: POR SIZE() DE ASNSEQUENCE
    *****/
    public int size()
    {
        int contador, vacio = 0x00, signo = 0x00;

        if(valor < 0)
        {
            vacio = 0xFF; signo = 0x80;
        }

        // ENTERO DE 32 BITS.. CAMBIA A 56 PARA ESCRIBIR LONGITUD DE 64 BITS
        for(contador=24; contador>0; contador-=8)
        {
            if(((valor >> contador) & 0xFF) != vacio) break;
        }
        // CHECA EL BIT DE SIGNO.. ASEGURA QUE EL BIT MSB EN 1 ES NEGATIVO,
        // POSITIVO ES 0.
        // (0x00000080 = 0x00 0x80) 0xFFFFFFFF01 => 0xFF 0x01
        // (0x0000007F = 0x7F) 0xFFFFFFFF80 => 0x80

        if(((valor >> contador) & 0x80) != signo)
        {
            contador += 8;
        }
    }
}
```

```

    return (contador>>3)+1;
}

/*****
ENTERO DE SALIDA
*****/
public void write(OutputStream out) throws IOException
{
    int contador, vacio = 0x00, signo = 0x00;

    if(valor < 0)
    {
        vacio = 0xFF;
        signo = 0x80;
    }

    // OBTIENE CONTADOR
    for(contador=24; contador>0; contador-=8)
    {
        if(((valor >> contador) & 0xFF) != vacio)
        {
            break;
        }
    }
    if(((valor >> contador) & 0x80) != signo)
    {
        contador += 8;
    }

    /*****
    CONSTRUYE CABECERA Y ESCRIBE EL VALOR
    *****/
    AsnBuildHeader(out, ASN_INTEGER, (contador>>3)+1);
    for(; contador>=0; contador-=8)
    {
        out.write((byte)((valor >> contador) & 0xFF));
    }
}

/*****
METODO: ASNINTEGER()
OBJETIVO: CODIFICA PAQUETES SNMPv1.
INVOCADO: POR ENCODESNMP(ENUM, STRING, INT, INT, INT),
          VARBIND(String,int),
*****/
public AsnInteger(int v)
{

```

```

this.valor = v;
}

```

```

/*****
METODO: ASNINTEGER(IN, LARGO)
OBJETIVO: OBTIENE UN VALOR DEL FLUJO DE ENTRADA.
INVOCADO: POR ASNMAKEME()),
*****/
public AsnInteger(InputStream in, int largo) throws IOException
{
    byte datos[] = new byte[largo];
    if(largo != in.read(datos,0,largo))
    {
        throw new IOException("NO HAY BASTANTES DATOS");
    }
    int val = bytesToInteger(datos);
    if((datos[0] & 0x80) != 0)
    {
        // ESTO ES NEGATIVO
        val = val - (1<<(largo<<3));
    }
    this.valor = val;
    System.out.println("EN ASNINTREGER, EL VALOR ES: "+val);
}

```

```

/*****
METODO: VALUE()
OBJETIVO: RETORNA EL VALOR QUE SE TIENE.
INVOCADO: POR GETREQID(), WHATERROR(), WHEREERROR(),
*****/
public int value() { return valor; }

```

```

/*****/
public String toString() { return (String.valueOf(valor)); }

```

```

/*****
METODO: BYTESTOINTEGER()
OBJETIVO: CONVIERTE DE BYTES A ENTEROS.
INVOCADO: POR ASNINTEGER(IN, LARGO),
*****/
protected int bytesToInteger(byte[] datos)
{
    int valor = 0;
    byte mascara = (byte) 0x7f;

    for(int n=0; n<datos.length; n++)
    {

```

```

byte ba = datos[n];
int v = (int) (mascara & ba);
if(ba<0)
{
    v+=128;
}
valor *= 256; valor += v;
}
return valor;
}

} // FIN DE LA CLASE ASNINTEGER

```

```

//=====
// CLASE           : ASNNULL
// AUTOR           : CARLOS GUIZAR
// VERSION ORIGINAL: JORDAN HARGRAVE
//
//=====

```

```

package snmp.stack;
import java.io.*;
import java.util.*;

```

```

public class AsnNull extends AsnObject
{
    /***/
    public void write(OutputStream out) throws IOException
    {
        AsnBuildHeader(out, ASN_NULL, 0);
    }
}

```

```

/***/
METODO: ASNNULL(IN, LARGO)
OBJETIVO: .
INVOCADO: POR ASNMAKEME(),
/***/

```

```

public AsnNull(InputStream in, int longitud)
{
}

```

```

/***/
METODO: ASNNULL()
OBJETIVO: ASIGNA VALOR NULO.
INVOCADO: POR VARBIND(oid), GETOBJ(int),
/***/

```

```

public AsnNull()
{
; //NADA QUE PONER
}

/*****/
public String toString()
{
return(new String("ASNNULL"));
}

} //FIN DE LA CLASE ASNNULL

//=====
// CLASE : ASNOBJECT
// AUTOR : CARLOS GUIZAR
// VERSION ORIGINAL: TIM PANTON
//
//=====

package snmp.stack;
import java.io.*;
import java.util.*;

//=====
// CLASE BASE ASN.1
//=====
abstract public class AsnObject extends Object
{
static final byte ASN_BOOLEAN =(byte)(0x01);
static final byte ASN_INTEGER =(byte)(0x02);
static final byte ASN_BIT_STR =(byte)(0x03);
static final byte ASN_OCTET_STR =(byte)(0x04);
static final byte ASN_NULL =(byte)(0x05);
static final byte ASN_OBJECT_ID =(byte)(0x06);
static final byte ASN_SEQUENCE =(byte)(0x10);
static final byte ASN_SET =(byte)(0x11);

static final byte ASN_UNIVERSAL =(byte)(0x00);
static final byte ASN_APPLICATION =(byte)(0x40);
static final byte ASN_CONTEXT =(byte)(0x80);
static final byte ASN_PRIVATE =(byte)(0xC0);

static final byte ASN_PRIMITIVE =(byte)(0x00);
static final byte ASN_CONSTRUCTOR =(byte)(0x20);

```

```

static final byte ASN_LONG_LEN      =(byte)(0x80);
static final byte ASN_EXTENSION_ID =(byte)(0x1F);
static final byte ASN_BIT8          =(byte)(0x80);

static final byte INTEGER           =(byte)ASN_INTEGER;
static final byte STRING            =(byte)ASN_OCTET_STR;
static final byte OBJID             =(byte)ASN_OBJECT_ID;
static final byte NULLOBJ          =(byte)ASN_NULL;
static final byte IPADDRESS        =(byte)(ASN_APPLICATION | 0);
static final byte COUNTER          =(byte)(ASN_APPLICATION | 1);
static final byte GAUGE             =(byte)(ASN_APPLICATION | 2);
static final byte TIMETICKS        =(byte)(ASN_APPLICATION | 3);
static final byte OPAQUE           =(byte)(ASN_APPLICATION | 4);

static final byte SNMP_TRAP_COLDSTART      =(byte)(0x0);
static final byte SNMP_TRAP_WARMSTART     =(byte)(0x1);
static final byte SNMP_TRAP_LINKDOWN      =(byte)(0x2);
static final byte SNMP_TRAP_LINKUP        =(byte)(0x3);
static final byte SNMP_TRAP_AUTHFAIL      =(byte)(0x4);
static final byte SNMP_TRAP_EGPNEIGHBORLOSS =(byte)(0x5);
static final byte SNMP_TRAP_ENTERPRISESPECIFIC =(byte)(0x6);

public static final byte SNMP_ERR_NOERROR      =(byte)(0x0);
public static final byte SNMP_ERR_TOOBIG      =(byte)(0x1);
public static final byte SNMP_ERR_NOSUCHNAME  =(byte)(0x2);
public static final byte SNMP_ERR_BADVALUE    =(byte)(0x3);
public static final byte SNMP_ERR_READONLY    =(byte)(0x4);
public static final byte SNMP_ERR_GENERR      =(byte)(0x5);
public static final byte SNMP_VERSION_1      =(byte)0;

public static final byte GET_REQ_MSG          =(byte)(ASN_CONTEXT |
ASN_CONSTRUCTOR | 0x0);
public static final byte GETNEXT_REQ_MSG     =(byte)(ASN_CONTEXT |
ASN_CONSTRUCTOR | 0x1);
public static final byte GET_RSP_MSG         =(byte)(ASN_CONTEXT |
ASN_CONSTRUCTOR | 0x2);
public static final byte SET_REQ_MSG         =(byte)(ASN_CONTEXT |
ASN_CONSTRUCTOR | 0x3);
public static final byte TRP_REQ_MSG         =(byte)(ASN_CONTEXT |
ASN_CONSTRUCTOR | 0x4);
public static final byte CONS_SEQ            =(byte)(ASN_SEQUENCE |
ASN_CONSTRUCTOR);

int cuantos = 0;

```

```
/******  
// RETORNA LA LONGITUD DEL TAMAÑO DEL CAMPO PARA LA LONGITUD  
// DE UN PAQUETE.  
// RETORNA:  
// 1 SI LA LONGITUD SE ADECUARA EN FORMA CORTA (0x00-0x7f)  
// 2+ SI LA LONGITUD >= 0x80  
/******  
public int LengthBytes(int longitud)  
{  
    int mascara, contador;  
  
    if(longitud < 0x80)  
    {  
        // FORMA CORTA.. 1 BYTE  
        return 1;  
    }  
    else  
    {  
        // FORMA LARGA.. BYTE DE PREFIJO + BYTES DE LONGITUD  
        mascara = 0xFF000000;  
        for(contador=4; (longitud&mascara)==0; contador--)  
        {  
            mascara >>= 8;  
        }  
        return contador+1;  
    }  
}  
  
/******  
CABECERA DE SALIDA ASN  
*****  
public void AsnBuildHeader(OutputStream out, byte tipo, int longitud) throws  
IOException  
{  
    int contador;  
  
    // BYTE DE TIPO  
    System.out.println("(AsnObject)TIPO= "+tipo);  
    out.write(tipo);  
  
    // BYTES DE LONGITUD  
    System.out.println("(AsnObject)LONGITUD= "+longitud);  
    contador = LengthBytes(longitud);  
  
    if(contador > 1)  
    {  
        // ESCRIBE EN FORMA LARGA EL BYTE DE PREFIJO
```

```
--contador;
byte tmp = (byte)(0x80 | (byte)contador);
out.write(tmp);
}

while(contador!=0)
{
// ESCRIBE BYTES DE LONGITUD
out.write((byte)((longitud >> (--contador << 3)) & 0xFF));
}
System.out.println("(AsnObject)VALOR = ");
}

/*****/
int readLength(InputStream in) throws IOException
{
int longitud = 0;
byte largo = (byte) in.read();

if((0x80 & largo) != 0)
{
// FORMA LARGA
int contador = (0x7f & largo);
if(contador < 4)
{
byte data[] = new byte[contador];
int n = in.read(data,0,contador);
if(n != contador)
{
throw new IOException("NO HAY BASTANTES DATOS");
}
else
{
for(n=0; n<contador; n++)
{
int val = (0x7f & data[n]);
if((0x80 & data[n]) != 0)
{
val += 128;
}
longitud = (longitud << 8) + val;
}
}
}
}
else
{

```

```
// FORMA CORTA
longitud = (int) (0x7f & largo);
}
return (longitud);
}

/*****
METODO: ASNMAKEME( IN, TIPO, LARGO)
OBJETIVO: CREA EL TIPO ESPECIFICADO.
INVOCADO: POR ASNREADHEADER(),
*****/
public AsnObject AsnMakeMe(InputStream in, byte tipo, int largo, int requien) throws
IOException
{
    AsnObject objetito = this;
    switch (tipo)
    {
        case CONS_SEQ :
            objetito = new AsnSequence(in,largo,requien);
            System.out.print("SE CONTRUYO UN _>_(48) CONS_SEQUENCE QUE
                CONTIENE: ");

            cuentos++;
            break;
        case GET_RSP_MSG :
            objetito = new AsnPduSequence(in,largo,requien);
            System.out.print("SE CONTRUYO UN _>_(-94) GET_RSP_MSG QUE
                CONTIENE: ");

            cuentos++;
            break;
        case GET_REQ_MSG :
            objetito = new AsnPduSequence(in,largo,requien);
            System.out.print("SE CONTRUYO UN _>_(-96) GET_REQ_MSG QUE
                CONTIENE: ");

            cuentos++;
            break;
        case ASN_INTEGER :
            objetito = new AsnInteger(in,largo);
            System.out.print("SE CONTRUYO UN _>_(2) ASN_INTEGER CONTIENE: ");
            cuentos++;
            break;
        case TIMETICKS :
            System.out.print("SE CONTRUYO UN _>_(67) TIMETICKS QUE CONTIENE: ");
            cuentos++;
            break;
        case COUNTER :
            System.out.print("SE CONTRUYO UN _>_(65) COUNTER QUE CONTIENE: ");
            cuentos++;
    }
}
```

```

        break;
    case GAUGE :
        objetito = new AsnUnsInteger(in,largo);
        System.out.print("SE CONTRUYO UN _>_(66) ASNUNSINTEGER QUE
            CONTIENE: ");

        cuantos++;
        break;
    case ASN_OBJECT_ID :
        objetito = new AsnObjectId(in,largo);
        System.out.print("SE CONTRUYO UN _>_(6) ASN_OBJECT_ID QUE
            CONTIENE: ");

        cuantos++;
        break;
    case IPADDRESS :
        System.out.print("SE CONTRUYO UN _>_(64) IPADDRESS CONTIENE: ");
        cuantos++;
        break;
    case ASN_OCTET_STR :
        objetito = new AsnOctets(in,largo);
        System.out.print("SE CONTRUYO UN _>_(4) ASN_OCTECT_STR QUE
            CONTIENE: ");

        cuantos++;
        break;
    case ASN_NULL :
        objetito = new AsnNull(in,largo);
        System.out.print("SE CONTRUYO UN _>_(5) ASN_NULL QUE CONTIENE: ");
        cuantos++;
        break;
    default :
        System.out.println(" .. .. TIPO INCORRECTO "+ tipo);
        break;
}
System.out.println(objetito+
    "<-----"
    "EL VALOR DE CUENTOS ES: "+cuantos+
    ");

return objetito;
}

/*****
METODO: ASNREADHEADER()
OBJETIVO: LEE EL FLUJO Y OBTIENE LA CABECERA.
INVOCADO: POR DECODESNMP(), ASNSEQUENCE(),
*****/
public AsnObject AsnReadHeader(InputStream in, int requien) throws IOException
{
    byte tipo;

```

```

int longi, obtubo;
AsnObject retorno = null;

// BYTE DE TIPO
tipo = (byte) in.read();           // TIPO OBTENIDO DEL FLUJO DE ENTRADA
System.out.println("AL LEER CABECERA, (ABAJO)TIPO = " + (int) tipo);

// BYTE DE LONGITUD
longi = readLength(in);           // LONGITUD OBTENIDA DEL FLUJO DE
ENTRADA
System.out.println("AL LEER CABECERA, (ABAJO)LONGITUD = " + (int) longi);

// LO QUE SE OBTUBO
byte cuerpo[] = new byte[longi];
obtubo = in.read(cuerpo,0,longi);
if(obtubo == longi)
{
    ByteArrayInputStream buf = new ByteArrayInputStream(cuerpo);
    retorno = AsnMakeMe(buf,tipo,longi,requien); // LLEGA EL OBJETO OBTENIDO
}
return retorno;                   //Y AQUI SE VA
}

/*****/
abstract public void write (OutputStream out) throws IOException ;

/*****/
METODO: SIZE()
OBJETIVO: RETORNA UN VALOR DE CERO, AL OBJETO.
INVOCADO: POR SIZE() DE ASNSEQUENCE.
*****/
public int size()
{
    return 0;
}

/*****/
METODO: ADD()
OBJETIVO: AGREGA UN HIJO A LA SECUENCIA.
INVOCADO: POR ENCODESNMP() NOPASA()
*****/
public AsnObject add(AsnObject child)
{
    return child;
}

```

```

/*****
METODO: FINDPDU()
OBJETIVO: RECURSIVAMENTE MIRA PARA UN OBJETO PDUSEQUENCE
SI NO SOBRESERIBE - ENTONCES NO ES UNA SECUENCIA -.
INVOCADO: POR DECODESNMP(),
*****/
public AsnObject findPdu()
{
    return null;
}

/*****/
abstract public String toString();

} // FIN DE LA CLASE ASNOBJECT

//=====
// CLASE           : ASNOBJECTID
// AUTOR           : CARLOS GUIZAR
// VERSION ORIGINAL: JORDAN HARGRAVE
//
//=====

package snmp.stack;

import java.io.*;
import java.util.*;

// ENTEROS SIN SIGNO DE 32-bit CODIFICADOS EN EL SIGUIENTE FORMATO:
// MSB..LSB
// 76543210
// xyyyyyyy
// \ \ /
// . \ /
// . \ /
// . +----- BITS DEL OID
// +------ 0 SI ESTE ES EL ULTIMO BYTE, 1 SI SIGUEN BYTES ADICIONALES
// .
// sid DE SALIDA, EN LA FORMA: abcdefgh.ijklmnop.qrstuvwxy.zABCDEF

public class AsnObjectId extends AsnObject
{
    // VALOR POR DEFAULT CON PROPOSITO DE PRUEBA.
    private int valor[] = {1,3,6,1,4,1,311,1,1,3,1}; // VALOR DEL OID DEL AGENTE.
}

```

```

/*****/
public boolean same(AsnObjectId aquel)
{
    boolean resultado = false;
    if(valor.length == aquel.valor.length)
    {
        int n=0;
        for(n=0; n<valor.length; n++)
        {
            if(this.valor[n] != aquel.valor[n])
            {
                break;
            }
        }
        resultado = (n == valor.length);
    }
    if(!resultado)
    {
        System.out.println("LOS NOMBRES NO SON LOS MISMOS (IGUALES)");
        System.out.println("ESTE= "+ toString() +" AQUEL= "+ aquel.toString());
    }
    return resultado; //AQUI RETORNA EL OID RESULTADO
}

/*****/
public String toString()
{
    String resultado = new String("");
    for(int n=0; n<valor.length-1; n++)
    {
        resultado = resultado + valor[n] + ",";
    }
    resultado = resultado + valor[valor.length-1];
    return resultado; //AQUI RETORNA EL OID CONVERTIDO EN CADENA
}

/*****
OBTIENE LA LONGITUD DEL SUBIDENTIFICADOR DEL OID
*****/
private int SIDLen(int valor)
{
    int cuento;

    for(cuento=1; (valor>>=7)!=0; cuento++);
    return cuento;
}

```

```

/*****
CODIFICA EL SUBIDENTIFICADOR DEL OID
*****/
private void EncodeSID(OutputStream out, int valor) throws IOException
{
    byte mascara = (byte)0x0F;
    int cuenta = 0;

    // MASCARA SUPERIOR ES DE 4 BITS
    mascara = 0xF;

    // CICLO MIENTRAS EL VALOR Y LA MASCARA SON CERO
    for(cuenta=28; cuenta>0; cuenta-=7)
    {
        if(((valor >> cuenta) & mascara) != 0) break;
        mascara = 0x7f;
    }

    // MIENTRAS CUENTA, VALOR DE SALIDA. SI ESTE NO ES EL ULTIMO BYTE,
    OUTPUT
    // 0x80 | valor.
    for(; cuenta>=0; cuenta-=7)
    {
        out.write((byte)(((valor >> cuenta) & mascara) | (cuenta>0 ? 0x80 : 0x00)));
        mascara = 0x7f;
    }
}

/*****
METODO: SIZE()
OBJETIVO: OBTIENE EL TAMAÑO TOTAL DEL OID.
INVOCADO: POR SIZE() DE ASNSEQUENCE
*****/
public int size()
{
    int val, idx, longitud;

    // First entry = OID[0]*40 + OID[1];
    longitud = SIDLen(valor[0]*40 + valor[1]);
    for(idx=2; idx<valor.length; idx++)
    {
        longitud += SIDLen(valor[idx]);
    }
    return longitud;
}

```

```

/*****
DATOS DE SALIDA
*****/
public void write(OutputStream out) throws IOException
{
    int idx;

    // CABECERA DE SALIDA
    AsnBuildHeader(out, ASN_OBJECT_ID, size());

    // BYTES DE DATOS DE SALIDA
    // First entry = OID[0]*40 + OID[1];
    EncodeSID(out, valor[0]*40 + valor[1]);
    for(idx=2; idx<valor.length; idx++)
    {
        EncodeSID(out, valor[idx]);
    }
}

/*****/
public AsnObjectId()
{
}

/*****/
METODO: ASNOBJECTID(IN, LONGITUD)
OBJETIVO: OBTIENE DEL FLUJO DE ENTRADA LOS VALORES DE ESTE TIPO.
INVOCADO: POR ASNMAKEME(),
*****/
public AsnObjectId(InputStream in, int longitud) throws IOException
{
    // OBTIENE NUESTRO DATO
    byte datos[] = new byte[longitud];
    if(longitud != in.read(datos,0,longitud)) //LEE EL FLUJO DE ENTRADA
    {
        throw new IOException("NO HAY SUFICIENTES DATOS");
    }

    // SIDS = SUBIDENTIFICADORES
    // AHORA DECIDE CUANTOS SID NECESITAREMOS
    // CONTAR LOS BYTES CON 0 EN EL BIT SUPERIOR - ENTONCES AGREGA 1
    int sids = 1; // EL PRIMER BYTE TIENE 2 sids EN ESTE
    for(int off=0; off<longitud; off++)
    {
        if(datos[off] >= 0)
        {
            sids++;
        }
    }
}

```

```

    }
  }
  // SE RECERVA ALGUN ESPACIO PARA LOS SIDS
  // CADA SUBIDENTIFICADOR REPRESENTA UN NIVEL EN LA MIB.
  valor = new int[sids];      //VALOR TIENE LA LONGITUD DE SIDS

  // DECODIFICA LOS PRIMEROS DOS
  valor[0] = datos[0] / 40;
  System.out.println("EL CONTENIDO DE VALOR[0] ES= "+valor[0]);
  valor[1] = datos[0] % 40;
  System.out.println("EL CONTENIDO DE VALOR[1] ES= "+valor[1]);

  // AHORA, DECODIFICA EL RESTO
  int off = 1;
  for(int idx=2; idx<valor.length; idx++)
  {
    int tval = 0;
    do
    {
      tval = tval << 7;
      tval |= (datos[off] & 0x7f);
    } while(datos[off++] < 0);
    valor[idx] = tval;
  }

  System.out.println("CONTENIDO DE VALOR[] en ASNOBJECTID(IN, LONG) ES=
    "+valor);
}

/*****
METODO: ASNOBJECTID()
OBJETIVO: CON EL OID RECIBIDO, IDENTIFICA SUS NUMEROS.
INVOCADO: POR VARBIND(String OID),
*****/
public AsnObjectId(String cadena)
{
  int pos, opos, cuenta, n;
  char c;

  // OBTIENE UN NUMERO DE DOTS
  cuenta = 1;
  for(pos = cadena.indexOf('.'); pos>=0; cuenta++) //RECORRE AENCONTRAR EL
    PUNTO

  {
    pos = cadena.indexOf('.',pos+1);
    //AQUI SE MANEJAN LAS POSICIONES DE LOS #S DEL OID, ENCONTRANDO
    EL PUNTO VA DE DOS EN DOS.
  }
}

```

```

    }
    valor = new int[cuenta];
    opos = 0;
    try
    {
        for(n=0; n<cuenta; n++)
        {
            String num;
            pos = cadena.indexOf('.',opos);
            if(pos > 0)
            {
                num = cadena.substring(opos,pos);
            }
            else
            {
                num = cadena.substring(opos);
            }
            Integer val = Integer.valueOf(num); //CONVIERTE EN ENTERO LA CADENA
                                                OBTENIDA

            valor[n] = val.intValue();
            opos = pos+1;
        }
    } catch (java.lang.NumberFormatException e)
    {
        System.out.println("OID INCORRECTO EN ASNOBJECTID: " + cadena +
        e.getMessage());
    }

    System.out.println("EL CONTENIDO DE VALOR[] EN ASNOBJECTID(STR) ES=
    "+valor);

} //FIN DEL METODO
} // FIN DE LA CLASE ASNOBJECTID

//=====
// CLASE           : ASNOCTETS
// AUTOR           : CARLOS GUIZAR
// VERSION ORIGINAL: JORDAN HARGRAVE
//
//=====

package snmp.stack;

import java.io.*;
import java.util.*;

```

```

public class AsnOctets extends AsnObject
{
    byte valor[];

    /**
    public void write(OutputStream out) throws IOException
    {
        int idx;

        // CABECERA DE SALIDA
        AsnBuildHeader(out, ASN_OCTET_STR, valor.length);

        // DATOS DE SALIDA
        for(idx=0; idx<valor.length; idx++)
        {
            out.write(valor[idx]);
        }
    }

    /**
    public int size()
    {
        return valor.length;
    }

    /**
    METODO: ASNOCTETS(SS[])
    OBJETIVO: OBTIENE EL VALOR DE CADA CARACTER QUE LLEGO.
    INVOCADO: POR ASNOCTETS(String)
    *****/
    public AsnOctets(char ss[])
    {
        int idx;

        valor = new byte[ss.length];
        for(idx=0; idx<ss.length; idx++)
        {
            valor[idx] = (byte)ss[idx];
        }
    }

    /**
    METODO: ASNOCTETS()
    OBJETIVO: CONVERTE A UN ARREGLO DE CARACTERES MANDANDO AL
    METODO DE ARRIBA.
    INVOCADO: POR ENCODESNMP()
    *****/

```

```

public AsnOctets(String cadena)
{
    this(cadena.toCharArray());
}

/*****/
public AsnOctets(byte s[])
{
    valor = s;
}

/*****/
METODO: ASNOCTETS()
OBJETIVO: DEL FLUJO DE ENTRADA, OBTENGO UNA CADENA.
INVOCADO: POR ASNMAKEME(),
/*****/
public AsnOctets(InputStream in, int longitud) throws IOException
{
    valor = new byte[longitud];
    if(longitud == in.read(valor,0,longitud))
    {
        String str = new String(valor,0);
        System.out.println("EN ASNOCTETS, EL TEXTO ES = "+ str);
        for(int j=0;j<2000000;j++);           // CICLO PARA DORMIR (RETARDO)
    }
    else
    {
        throw new IOException("NO HAY SUFICIENTES DATOS");
    }
}

/*****/
public String toString()
{
    return (new String(valor, 0));
}

} // FIN DE LA CLASE ASNOCTETS

//=====
// CLASE           : ASNPDUSEQUENCE
// AUTOR           : CARLOS GUIZAR
// VERSION ORIGINAL: JORDAN HARGRAVE
//
//=====

```

```
package snmp.stack;
import java.io.*;
import java.util.*;

public class AsnPduSequence extends AsnSequence
{
    /**
     * METODO: WHATERROR()
     * OBJETIVO: OBTIENE EL OBJETO Y RETORNA SU VALOR
     * INVOCADO: POR LLENALO(SEQ) DE PDU,
     */
    int whatError()
    {
        AsnInteger estat = (AsnInteger) getObj(1);
        return (estat.value());
    }

    /**
     * METODO: WHEREERROR()
     * OBJETIVO: OBTIENE EL OBJETO Y RETORNA SU VALOR
     * INVOCADO: POR LLENALO(SEQ) DE PDU,
     */
    int whereError()
    {
        AsnInteger estat = (AsnInteger) getObj(2);
        return (estat.value());
    }

    // OBSERVA RECURSIVAMENTE POR UN OBJETO PDUSEQUENCE HASTA
    // OBTENER UNO
    /**
     */
    public AsnObject findPdu()
    {
        return this;
    }

    /**
     * METODO: ASNPDUSEQUENCE(IN, LEN)
     * OBJETIVO: CREA EL TIPO ESPECIFICADO, INVOCANDO SU SUPERCLASE.
     * INVOCADO: POR ASNMAKEME(),
     */
    public AsnPduSequence(InputStream in, int len, int requien) throws IOException
    {
        super(in,len,requien);
    }
}
```

```

/*****/
public int value()
{
    AsnSequence varBind = (AsnSequence) getObj(3);
    AsnSequence varPair = (AsnSequence) varBind.getObj(0);
    AsnInteger val = (AsnInteger) varPair.getObj(1);
    int valore = val.value();
    return valore;
}

/*****/
boolean HadError()
{
    return (SNMP_ERR_NOERROR != whatError());
}

/*****
METODO: GETREQID()
OBJETIVO: DE UN PAQUETE OBTIENE SU ID DEL REQ. Y RETORNA SU
VALOR.
INVOCADO: POR RUN() DE CONTEXTO,
*****/
int getReqId()
{
    AsnInteger rid = (AsnInteger) getObj(1);
    return(rid.value());
}

} // FIN DE LA CLASE ASNPDUSEQUENCE.

//=====
// CLASE           : ASNSEQUENCE
// AUTOR           : CARLOS GUIZAR
// VERSION ORIGINAL: TIM PANTON
//
//=====

package snmp.stack;

import java.io.*;
import java.util.*;

public class AsnSequence extends AsnObject
{
    private Vector vectorcito;
    private byte tipo;

```

```

/*****
METODO: ADD(HIJO)
OBJETIVO: AGREGA EL OBJETO AL VECTOR.
INVOCADO: POR ASNSEQUENCE(IN, LEN, REQUIEN),
*****/
public AsnObject add(AsnObject hijo)
{
    vectorcito.addElement(hijo);
    return hijo;
}

/*****
METODO: SIZE()
OBJETIVO: OBTIENE EL TAMAÑO DE LOS HIJOS.
INVOCADO: POR ?.
*****/
public int size()
{
    // METODO PERMANENTEMENTE USADO DURANTE LOS N HILOS
    // PARA AJUSTAR LOS TAMAÑOS DE LOS DATOS.

    Enumeration listadehijos = vectorcito.elements();
    int    cnt, tamaño = 0;

    while(listadehijos.hasMoreElements())
    {
        // OBTIENE EL TAMAÑO DEL HIJO
        cnt = ((AsnObject)listadehijos.nextElement()).size();

        // AGREGA ESPACIO PARA CODIFICACION DE BYTE DE TIPO Y LONGITUD
        cnt += (1+LengthBytes(cnt));
        tamaño += cnt;
    }
    return tamaño;
}

/*****/
public void write(OutputStream out) throws IOException
{
    // CABECERA DE SALIDA
    AsnBuildHeader(out, tipo, size());

    // HIJOS DE SALIDA
    Enumeration listadehijos = vectorcito.elements();
    while(listadehijos.hasMoreElements())
    {

```

```

    AsnObject hijo = ((AsnObject)listadehijos.nextElement());
    hijo.write(out);
}
}

/*****
METODO: ASNSEQUENCE()
OBJETIVO: INVOCA A SI MISMO ABAJO.
INVOCADO: POR ASNSEQUENCE(IN, LEN),
*****/
public AsnSequence()
{
    this(CONS_SEQ);    //48
}

/*****
METODO: ASNSEQUENCE(BYTE)
OBJETIVO: CREA UN VECTOR.
INVOCADO: POR ASNSEQUENCE(IN, LEN),
*****/
public AsnSequence(byte oddtype)
{
    tipo = oddtype;
    vectorcito = new Vector(5,2);
}

/*****
METODO: ASNSEQUENCE(IN, LEN)
OBJETIVO: INVOCA A SI MISMO ARRIBA Y AGREGA EL OBJETO OBTENIDO.
INVOCADO: POR ASNMAKEME(), ASNPDUSEQUENCE(IN, LEN)
*****/
public AsnSequence(InputStream in, int len, int requien) throws IOException
{
    this();
    AsnObject a = null;
    //UNA RECURSION PARA CONOCER CONTENIDO DE TODA LA SECUENCIA
    while(null != (a = AsnReadHeader(in, requien)))
    {
        add(a);
    }
}

/*****
METODO: FINDPDU()
OBJETIVO: OBSERVA RECURSIVAMENTE POR UN OBJETO PDUSEQUENCE
INVOCADO:
*****/

```

```

public AsnObject findPdu()
{
    AsnObject objres = null;

    Enumeration listadehijos = vectorcito.elements();
    while(listadehijos.hasMoreElements())
    {
        AsnObject hijo = ((AsnObject) listadehijos.nextElement());
        objres = hijo.findPdu(); //ESTE METODO ES DE ASNOBJECT
        if(objres != null)
        {
            break;
        }
    }
    return objres;
}

/*****
METODO: GETOBJ(INT)
OBJETIVO: DEL VECTOR CREA UN OBJETO Y LO RETORNA.
INVOCADO: POR GETREQID(), WHATERROR(), WHEREERROR(), LLENALO(),
          SET_VALUE(),
*****/
public AsnObject getObj(int offset)
{
    AsnObject res = new AsnNull();
    try
    {
        res = (AsnObject) vectorcito.elementAt(offset);
    } catch (ArrayIndexOutOfBoundsException exc)
    {
    }
    return res;
}

/*****/
public String toString()
{
    return "";
}

} // FIN DE LA CLASE ASNSEQUENCE

```

```

//=====
// CLASE           : ONEINTPDU
// AUTOR           : CARLOS GUIZAR
// VERSION ORIGINAL: TIM PANTON
//
//=====

package snmp.pdu;
import snmp.stack.*;
import java.util.*;

public class OneIntPdu extends Pdu
{
    Integer valor;

    /*****
    METODO: ONEINTPDU()
    OBJETIVO: INVOCA A LA SUPERCLASE.
    INVOCADO:
    *****/
    public OneIntPdu(SnmpContext con, int requien)
    {
        super(con, requien);
    }

    /*****
    METODO: ONEINTPDU(con, comuni, oid, requien, o)
    OBJETIVO: LLAMA AL CONSTRUCTOR DE PDU, AGREGA UN OID Y ENVIA LA
    COMUNIDAD
    INVOCADO: POR ONEINTPDU(con, comuni, pid, requien),
    *****/
    public OneIntPdu(SnmpContext con, String comuni, String oid, int requien, Observer o)
    {
        super(con, requien);           //NOS VAMOS A LA CLASE PDU

        if(o != null)
        {
            System.out.println("...EL OBSERVADOR ES DIFERENTE DE NULL...");
            addObserver(o);
        }
        add_oid(oid, requien);         //AGREGANDO EL OBJETO
        send(comuni);                  //ENVIA COMUNIDAD E INICIALIZA EL PDU
    }

    /*****
    METODO: ONEINTPDU(con, comuni, oid, requien)
    OBJETIVO: SUPRIME EL HILO E INVOCA A SI MISMO ARRIBA

```

```

INVOCADO: POR EL ADMIN5, AGENT4
*****/
public OneIntPdu(SnmpContext con, String comuni, String oid, int requien)
{
    this(con,comuni,oid,requien,null);
}

/*****
METODO: NEW_VALUE(n, res)
OBJETIVO: ESTO REQUIERE MODIFICACIÓN
INVOCADO: POR LLENALO(),
*****/
protected void new_value(int n, varbind res, int requien)
{
    System.out.println("///EL VALOR DE n EN ONEINTPDU NEW_VALUE() ES: "+n);
    System.out.println("EL VALOR DE res EN ONEINTPDU NEW_VALUE() ES: "+res);

    if(requien == 13)
    {
        AsnInteger va = (AsnInteger) res.obt_valor(); //AQUI PASA ALGO CON NULL, NO
                                                    LO CONVIERTE
        System.out.println("///("+requien+")EL VALOR DE VA EN ONEINTPDU() ES: "+va);
        if(n == 0)
        {
            valor = new Integer(va.value());
            System.out.println("EL VALOR EN ONEINTPDU NEW_VALUE() ES: "+valor);
        }
    }
    if(requien == 6)
    {
        AsnNull va = (AsnNull) res.obt_valor();
        System.out.println("///("+requien+")EL VALOR DE VA EN ONEINTPDU() ES: "+va);
        if(n == 0)
        {
            valor = new Integer(0); //(va.value());
            System.out.println("EL VALOR EN ONEINTPDU NEW_VALUE() ES: "+valor);
        }
    }
}

/*****/
protected void tell_them()
{
    notifyObservers(valor);
}

```

```

/*****/
public Integer getValue()
{
    return valor;
}

} // FIN DE LA CLASE ONEINTPDU

//=====
// CLASE           : PDU
// AUTOR           : CARLOS GUIZAR
// VERSION ORIGINAL: TIM PANTON
//
//=====

package snmp.stack;
import java.util.*;
import java.io.*;

public abstract class Pdu extends Observable
{
    private static int id_siguiete = 1, intervalo_intento[] = {500,1000,2000,5000,5000};
    protected Vector vector_vars_enlaza;
    int id_del_req, errstat, errind, recibidos;
    SnmpContext contexto;
    Transmitter transmisor = null;
    boolean contestado, agregado = false, obtubo = false;
    String comunidad;
    byte tipo_mensaje;

    /*****
    METODO: NEW_VALUE(int, varbind)
    OBJETIVO: .
    INVOCADO: POR LLENALO(SEQ),
    *****/
    abstract protected void new_value(int n, varbind res, int requieren);

    /*****
    METODO: TELL_THEM()
    OBJETIVO: .
    INVOCADO: POR LLENALO(SEQ),
    *****/
    abstract protected void tell_them();

    /*****

```

METODO: SETTRANS()
 OBJETIVO: ASIGNA EL HILO CREADO.
 INVOCADO: POR ADDPDU(p) DE CONTEXTO

*****/

```
void setTrans(Transmitter t)
{
    transmisor = t;
}
```

METODO: LLENALO(SEQ,INT)
 OBJETIVO: LLENA LA SECUENCIA QUE FORMA EL PDU
 INVOCADO: POR RUN() DEL CONTEXTO,

*****/

```
void llenalo(AsnPduSequence seq, int requien)
{
    if(contestado)
    {
        System.out.println("SE OBTUVO UNA SEGUNDA RESPUESTA AL
            REQUERIMIENTO ");
        return;
    }
    setErrorStatus(seq.whatError()); //LE PONE VALOR AL ESTADO DE ERROR
    setErrorIndex(seq.whereError()); //LE PONE VALOR AL INDICE DE ERROR

    AsnSequence varBind = (AsnSequence) seq.getObj(3);
    int n=0;
    for(Enumeration e = vector_vars_enlaza.elements(); e.hasMoreElements();n++)
    {
        AsnSequence va = (AsnSequence) varBind.getObj(n);
        varbind vb = (varbind) e.nextElement();
        System.out.println("(1)** LA VARIABLE ENLAZADA VB EN LLENALO() ES:
            "+vb);

        AsnObject res = (AsnObject) vb.set_value(va);
        System.out.println("(2)** EL OBJETO RESULTANTE RES EN LLENALO() ES:
            "+res);

        setChanged(); // DEL OBSERVADOR
        new_value(n, vb, requien);
    }

    tell_them(); //AQUI SE VA A ONEINTPDU
    clearChanged(); //DEL OBSERVADOR
    synchronized(this)
    {
```

```

    obtubo = true;
    contestado = true;
    notify();
    transmisor.interruptMe();
}
}

```

```

/*****
METODO: SENDME()
OBJETIVO: CODIFICA Y ESCRIBE EL PAQUETE.
INVOCADO: POR TRANSMIT()
*****/

```

```

boolean sendme(String comunidad, int requieren) throws java.io.IOException
{
    AsnEncoder enc = new AsnEncoder();
    Enumeration vbs = vector_vars_enlaza.elements();

    ByteArrayOutputStream bay = enc.EncodeSNMP(vbs, comunidad, id_del_req, errstat,
erind, tipo_mensaje, requieren);
    byte [] paquete = bay.toByteArray(); //EL FLUJO A ARREGLO DE BYTES
    System.out.println("|||||||||||||" + "EL PAQUETE CODIFICADO ES: " + paquete);
    contexto.writePacket(paquete);

    System.out.println(" REQUERIMIENTO ENVIADO: " + id_del_req+
        "RECIBIDOS = "+recibidos+
        "EN EL HILO: "+Thread.currentThread());
    recibidos ++;
    return (recibidos > 3);
}

```

```

/*****
METODO: SEND()
OBJETIVO: INICIALIZA EL PDU Y LO AGREGA.
INVOCADO: POR ONEINTPDU(con, comuni, oid, requieren, o),
*****/

```

```

public boolean send(String comuni)
{
    if(agregado & (transmisor != null))
    {
        comunidad = comuni;
        transmisor.setPdu(this);
        transmisor.wake();
    }
    return agregado;
}

```

```

/*****

```

```

public void add_oid(varbind variable)
{
    vector_vars_enlaza.addElement(variable);
}

/*****
METODO: ADD_OID()
OBJETIVO: CREA UNA VARIABLE ENLAZADA CON EL OID Y LA AGREGA.
INVOCADO: POR ONEINTPDU(con, comuni, oid, requien, o),
*****/
public void add_oid(String oid, int requien)
{
    varbind vb = new varbind(oid, requien); //LA CREA
    System.out.println("<<< LA VARIABLE ENLAZADA DEL OID ES: "+vb);
    vector_vars_enlaza.addElement(vb); //AGREGA AL VECTOR EL OID Y VALOR
} // OBTENIDOS.

/*****
METODO: PDU()
OBJETIVO: INICIALIZA TODOS LOS CAMPOS DEL PDU.
INVOCADO: POR ONEINTPDU(con, comuni, oid, o)
*****/
public Pdu(SnmpContext con, int requien)
{
    contexto = con;
    id_del_req = id_siguiete; //EN SNMP PDU ES EL (1).
    errstat = AsnObject.SNMP_ERR_NOERROR; //EN SNMP PDU ES EL (2).
    errind = 0x00; //EN SNMP PDU ES EL (3).
    id_siguiete++;
    vector_vars_enlaza = new Vector(1,1); //EN SNMP PDU ES EL (4). SOLO MANEJA
    UN DATO
    agregado = contexto.addPdu(this, requien); //AQUI SE HACE TODO LO BUENO

//EN ESTE CASO ES UN GET, PUEDE SER SET, GETNEXT, RESPONSE O TRAP.
if(requien == 13)
{
    tipo_mensaje = AsnObject.GET_REQ_MSG; // PARA UN REQUERIMIENTO -96h
}
if(requien == 6)
{
    tipo_mensaje = AsnObject.GET_RSP_MSG; // PARA UNA RESPUESTA -94h
}
contestado = true;

    System.out.println("EL REQUERIMIENTO A REALIZAR ES: "+tipo_mensaje);
} // FIN DEL METODO PDU()

```

```

/*****/
public void set_msg_type(byte tipo)
{
    tipo_mensaje = tipo;
    System.out.println("EL TIPO CON QUE SE INICIALIZO EL MENSAJE ES:
                        "+tipo_mensaje+ "oooooooooooo");
}

/*****/
public int getReqId()
{
    return id_del_req;
}

/*****/
public int getErrorStatus()
{
    return errstat;
}

/*****/
public int getErrorIndex()
{
    return errind;
}

/*****/
METODO: TRANSMIT()
OBJETIVO: SI NO ESTA CONTESTADO ENVIAME Y A DORMIR.
INVOCADO: POR RUN() DE TRANSMITTER
*****/
public void transmit(int requien)
{
    int n=0;
    contestado=false;

    while(!contestado && n<interva_intento.length)
    {
        try
        {
            sendme(comunidad,requien);
        } catch (java.io.IOException e)
        {
            System.out.println("TRANSMIT() DE PDU, UNA EXCEPCION DE E/S: " +
                               e.getMessage());
        }
    }
}

```

```
    }
    try
    {
        Thread.sleep(interva_intento[n]);
    } catch (java.lang.InterruptedException e)
    {
        // IGNORAR ESTO
    }
    n++;
}

if(!contestado)
{
    handleNoAnswer();
}
if(!contexto.removePdu(id_del_req))
{
    System.out.println("FALLO AL REMOVER EL PDU CON ID DEL REQ:
                        "+id_del_req);
}
}

/*****/
public synchronized boolean waitForSelf(long retardo)
{
    if(!obtuvo)
    {
        try
        {
            wait(retardo);
        } catch (InterruptedException ix)
        {
            ;
        }
    }
    return contestado;
}

/*****/
public boolean waitForSelf()
{
    long del = 0;
    for(int i=0; i<interva_intento.length; i++)
    {
        del += interva_intento[i];
    }
    return waitForSelf(del);
}
```

```

}

/*****/
public String toString()
{
    return getClass().getName()
        + "["+ "ID DEL REQ= "+ id_del_req+ ", TIPO_MENSAJE= "+ tipo_mensaje
        + ", VARS_ENLAZADAS= "+ vector_vars_enlaza + "];"
}

/*****
METODO: SETERRORSTATUS()
OBJETIVO: INICIALIZA EL ESTADO DE ERROR.
INVOCADO: POR HANDLENOANSWER()
*****/
protected void setErrorStatus(int err)
{
    errstat = err;
}

/*****
METODO: SETERRORINDEX()
OBJETIVO: INICIALIZA EL INDICE DE ERROR.
INVOCADO: POR HANDLENOANSWER()
*****/
protected void setErrorIndex(int ind)
{
    errind = ind;
}

/*****
METODO: HANDLENOANSWER()
OBJETIVO: CUANDO LA RESPUESTA NO ES RECIBIDA DESPUES DE TODOS
LOS QUE SE ESPERABAN
LA APLICACION ES NOTIFICADA DE ESTO.
INVOCADO: POR TRANSMIT()
*****/
private void handleNoAnswer()
{
    setErrorStatus(AsnObject.SNMP_ERR_GENERR);
    setErrorIndex(0);

    setChanged(); //DEL OBSERVADOR
    tell_them();
    clearChanged(); //DEL OBSERVADOR

    synchronized(this)

```

```

    {
        notify();                // NOTIFICA AL HILO QUE FUE SINCRONIZADO
    }
}

} //FIN DE LA CLASE PDU

```

```

=====
// CLASE           : SNMPCONTEXT
// AUTOR           : CARLOS GUIZAR
// VERSION ORIGINAL: TIM PANTON
//
=====

```

```

package snmp.stack;
import java.net.*;
import java.io.*;
import java.util.*;

public class SnmpContext extends Object implements Runnable
{
    final static int MAXPDU = 20;
    final static int TAMAXSO = 1300; // TAMAÑO MAXIMO QUE RECIBE EL SOCKET;

    Pdu []           lospdus;
    Transmitter []  transmisores;
    InetAddress     direcdehost;
    int              age_adm_puerto, elpuerte, requien, admipue, elpuerto, banda;
    DatagramSocket  soc;
    Thread           hilito;
    InetAddress     admidir;

    /**
    public synchronized void destruir() //oy()
    {
        if(hilito != null)
        {
            hilito.destroy();
            hilito = null;
            for(int i=0;i<MAXPDU; i++)          // REPITE MAXPDU VECES
            {
                if(transmisores[i] != null)
                {
                    transmisores[i].destroy();
                }
            }
        }
    }
}

```

```

    }
  }
  System.out.println("VOY A DESTRUIR EL SOCKET DATAGRAMA, YA..... ");
  soc.close();
}

```

```

/*****

```

```

public synchronized void parar()
{
  if(hilito != null)
  {
    hilito.stop();
    hilito = null;
  }
}

```

```

/*****

```

METODO: RUN()
OBJETIVO: CREA DATAGRAMPACKET Y RECIBE DEL SOCKET, SI NO HAY PAQUETE SE BLOQUEA. SI HAY PAQ. LO HACE INPUTSTREAM Y LO DECODIFICA OBTENIENDO UNA SECUENCIA CON UN ID DEL REQ. Y UN PDU PARA LLENARLO.

INVOCADO: POR SNNMPCONTEXT(),

```

*****

```

```

public void run()
{
  while(true) //HILO QUE TIENE MAX PRIORIDAD Y SIEMPRE CORRIENDO
  {
    if(requien == 13)
    {
      AsnDecoder rpdu = new AsnDecoder();
      try
      {
        System.out.println("~~~~~"
          + "!!!!!! WHILE(TRUE) WHILE(TRUE) !!!!!!!"
          + "~~~~~");
        byte [] datos = new byte[TAMAXSO];
        DatagramPacket paqueti = new DatagramPacket(datos,TAMAXSO);

        System.out.println("ESTE ES EL HILO ---> HILITO: "+hilito);
        hilito.yield(); //CAMBIO DE CONTEXTO ENTRE EL HILO ACTUAL
          Y EL SIGUIENTE DISPONIBLE

        soc.receive(paqueti);
        ByteArrayInputStream in = new ByteArrayInputStream (paqueti.getData(), 0,
          paqueti.getLength());

```

```

AsnPduSequence seq = rpdu.DecodeSNMP(in, requien);
if(seq != null)
{
    Integer rid = new Integer(seq.getReqId());
    System.out.println("...EL RID ES > : "+rid);
    Pdu answ = getPdu(rid); // LOCALIZA UN PDU CON EL ID DEL REQ.
    System.out.println("...EL PDU ANSW CONTIENE > : "+answ+" "+
        ".....");
    if(answ != null)
    {
        answ.llenalo(seq, requien);
    }
}
else
{
    System.out.println("ERROR - FALTA SECUENCIA DE ENTRADA");
}

} catch(java.io.IOException e)
{
    System.out.println("RUN() DE SNMPCONTEXT, EXCEPCION DE E/S: " +
e.getMessage());
}
}

if(requien == 6)
{
    AsnDecoder rpdu = new AsnDecoder();
    try
    {
        System.out.println(".....+
            " !!!!!!!!!!! WHILE(TRUE) WHILE(TRUE) !!!!!!!!!!!"+
            ".....");

        byte [] datos = new byte[TAMAXSO];
        DatagramPacket paqueti = new DatagramPacket(datos,TAMAXSO);

        System.out.println("ESTE ES EL HILO ACTUAL ---> HILITO: "+hilito);
        hilito.yield(); //CAMBIO DE CONTEXTO ENTRE EL HILO ACTUAL Y EL
            SIGUIENTE DISPONIBLE
        soc.receive(paqueti); //INSTRUCCION BLOQUEANTE
        admidir = paqueti.getAddress(); //OBTIENE LA IP DEL ADMINISTRADOR
        admipue = paqueti.getPort(); //OBTIENE EL PUERTO DEL ADMINISTRADOR

        System.out.println("vvvVVVvvvVVVvvVVVvvVVVvvVVVvvVVVvvVVVvvVVV"+
            "EL PUERTO DEL ADMIN ES: "+admipue+" LA IP DEL

```

```

ADMINISTRADOR          ES:          "+admidir+"
"vvvVVVvvvVVVvvvVVVvvvVVVvvvVVVvvvVVVvvvVVV");

ByteArrayInputStream in = new ByteArrayInputStream(paqueti.getData(), 0,
paqueti.getLength());
AsnPduSequence seq = rpdu.DecodeSNMP(in, requien); //OBTIENE SECUENCIA
DE OBJETOS
if(seq != null)
{
Integer rid = new Integer(seq.getReqId());
System.out.println("...EL RID ES > : "+rid);

Pdu answ = getPdu(rid); // LOCALIZA UN PDU CON EL ID DEL REQ.
System.out.println(".....EL PDU ANSW CONTIENE > : "+answ+" "+
".....");

if(answ != null)
{
answ.llenalo(seq,requien);
}
}
else
{
System.out.println("ERROR - FALTA SECUENCIA DE ENTRADA");
}
} catch(java.io.IOException e)
{
System.out.println("RUN() DE SNMPCONTEXT, EXCEPCION DE E/S: " +
e.getMessage());
}
}

} // FIN DEL WHILE
} // FIN DEL METODO RUN

/*****
METODO: WRITEPACKET()
OBJETIVO: RECIVE PAQUETE SNMP CODIFICADO CORRECTAMENTE Y LO
ENVIA...
INVOCADO: POR SENDME()
*****/
synchronized void writePacket(byte[] paquet)
{
if(requien == 13)
{
DatagramPacket paquetin = new DatagramPacket(paquet,paquet.length,direcdehost,
age_adm_puerto);

try

```

```

{
    soc.send(paquetin); //ENVIA POR EL SOCKET EL PAQUETE... CON EL PDU
    System.out.println(" ** EL PAQUETE SE HA ENVIADO AL AGENTE ** "
        "*****");
} catch (IOException iox)
{
    System.out.println("AL TRATAR DE ENVIAR POR EL SOCKET, EXCEPCION
        DE E/S: "+ iox);
}
} //FIN DE IF

if(requien == 6)
{
    DatagramPacket paquetin = new DatagramPacket(paquet,paquet.length, admirir,
        admipue);
    try
    {
        soc.send(paquetin); //ENVIA POR EL SOCKET EL PAQUETE... CON EL PDU
        System.out.println("** EL PAQUETE SE ENVIO AL ADMINISTRADOR ** +
            "*****");
    } catch (IOException iox)
    {
        System.out.println("AL TRATAR DE ENVIAR POR EL SOCKET, EXCEPCION
            DE E/S: "+ iox);
    }
} //FIN DE IF
}

/*****
METODO: GETPDU(INTEGER)
OBJETIVO: AQUI SE INVOCA AL METODO GETPDU DE ABAJO
INVOCADO: POR RUN(),
*****/
Pdu getPdu(Integer ReqId)
{
    return getPdu(ReqId.intValue());
}

/*****
METODO: GETPDU(INT)
OBJETIVO: ENCUENTRA EL PDU SOLICITADO Y LO RETORNA.
INVOCADO: POR GETPDU(INTEGER),
*****/
Pdu getPdu(int rid)

```

```

{
  Pdu ret = null;
  for(int i=0; i<MAXPDU; i++)
  {
    if((lospdus[i] != null) && (lospdus[i].id_del_req == rid))
    {
      ret = lospdus[i];
      break;
    }
  }
  return ret;
}

```

```

/*****
METODO: REMOVEPDU()
OBJETIVO: CON EL RID IDENTIFICA EL PDU A BORRAR.
INVOCADO: POR TRANSMIT() DE PDU
*****/

```

```

synchronized boolean removePdu(int rid)
{
  boolean ret = false;
  for(int i=0; i<MAXPDU; i++)
  {
    if((lospdus[i] != null) && (lospdus[i].id_del_req == rid))
    {
      System.out.println("PDU REMOVIDO DE "+(i+1) +" CON ID DEL REQ = "+rid);
      lospdus[i] = null;
      ret = true;
      break;
    }
  }
  return ret;
}

```

```

/*****
METODO: ADDPDU()
OBJETIVO: AGREGA EL PDU QUE RECIVE.
INVOCADO: POR PDU()
*****/

```

```

synchronized boolean addPdu(Pdu p, int requien)
{
  boolean done = false;
  for(int i=0; i<MAXPDU; i++)
  {
    if(lospdus[i] == null) // SI EL ESPACIO EN EL ARREGLO ESTA VACIO
    {
      lospdus[i] = p;
    }
  }
}

```

```

        lospdus[i].setTrans(getTrans(i,requien));
        done = true;

        System.out.println("AGREGO PDU No. "+(i+1) +" CON ID DEL REQ =
            "+p.id_del_req);
        System.out.println("CONTENIDO DEL ARREGLO PDUS = "+lospdus[i]);
        break;
    }
}
return done;
}

/*****
METODO: GETTRANS()
OBJETIVO: CREA NUEVO HILO PARA TRANSMITIR.
INVOCADO: POR ADDPDU(p)
*****/
Transmitter getTrans(int i, int requien)
{
    if(transmisores[i] == null)        // SI EL ESPACIO EN EL ARREGLO NO EXISTE.
    {
        transmisores[i] = new Transmitter("SNMPTRANS: "+i,requien);    // CREA NUEVO
HILO
    }
    return transmisores[i];
}

/*****
METODO: SNMPCONTEXT()
OBJETIVO: CREA SOCKET, HILO Y MAXPDU'S, OBTIENE LA IP DEL AGENTE
INVOCADO: POR ADMIN5,
*****/
public SnmpContext(String host, int puerto, int quien) throws java.io.IOException
{
    if(quien == 13)
    {
        lospdus = new Pdu[MAXPDU];
        age_adm_puerto = puerto;
        transmisores = new Transmitter[MAXPDU];
        requien = quien;

        try
        {
            direcdehost = InetAddress.getByName(host);        //RECIBE NOMBRE PARA
                                                                OBTENER LA IP
            System.out.println("ADMIN> LA DIRECCION DEL HOST ES: "+direcdehost);
            System.out.println("ADMIN> EL VALOR DE QUIEN = "+quien);

```

```

try
{
    soc = new DatagramSocket();
    hilito = new Thread(this);
    hilito.setPriority(hilito.MAX_PRIORITY);
    hilito.start();
} catch (SocketException sex)
{
    System.out.println ("PROBLEMAS AL CREAR SOCKETS " + sex);
}
} catch (UnknownHostException nohost)
{
    System.out.println ("NO ENCONTRA EL HOST DEL AGENTE" + nohost);
}
}

if(quien == 6)
{
    lospdus = new Pdu[MAXPDU];
    transmisores = new Transmitter[MAXPDU];
    requien = quien;

    System.out.println("AGENTE> EL VALOR DE QUIEN = "+quien);
    try
    {
        soc = new DatagramSocket();
        elpuerte = soc.getLocalPort(); //OBTIENE EL PUERTO localmente QUE USARA
                                     EL AGENTE
        //elpuerte = this.dapuerto(); //ESTO ES PARA USAR EL METODO DE ESTA
                                     CLASE
        System.out.println("AGENTE POR EL PUERTO : "+elpuerte);

        hilito = new Thread(this);
        hilito.setPriority(hilito.MAX_PRIORITY);
        hilito.start();
    } catch (SocketException sex)
    {
        System.out.println ("PROBLEMAS AL CREAR SOCKETS " + sex);
    }
}

} // FIN DEL METODO SNMPCONTEXT

/*****/
public int dapuerto()
{
    if(requien == 13)

```

```
{
    elpuerto = soc.getLocalPort();    //OBTIENE EL PUERTO LOCAL
}
if(requien == 6)
{
    elpuerto = 161;                //VALOR POR DEFAULT PARA EL PUERTO
}
return elpuerto;
}

/*****/
public String toString()
{
    char [] cret = new char[MAXPDU];
    for(int i=0; i<MAXPDU; i++)
    {
        if(transmisores[i] != null)
        {
            if(lospdus[i] != null)
            {
                cret[i] = (char) ('A' + (lospdus[i].getReqId() % 26));
            }
            else
            {
                cret[i] = '-';
            }
        }
        else
        {
            cret[i] = '-';
        }
    }
    return new String(cret);
}

} //FIN DE LA CLASE SNMPCONTEXT

//=====
// CLASE          : TRANSMITTER
// AUTOR          : CARLOS GUIZAR
// VERSION ORIGINAL: TIM PANTON
//
//=====

package snmp.stack;
import java.util.*;
```

```
class Transmitter extends Object implements Runnable
```

```
{
    Pdu  unpdu = null;
    Thread hilo;
    String minombre;
    int quien;

    /**
     * METODO: TRANSMITTER()
     * OBJETIVO: CREA NUEVO HILO PARA TRANSMITIR.
     * INVOCADO: POR GETTRANS(INT)
     */
    Transmitter(String nombre, int requien)
    {
        hilo = new Thread(this,nombre);    //CREA EL NUEVO HILO
        hilo.setPriority(hilo.MIN_PRIORITY);
        minombre = nombre;                //DEFINE MINOMBRE
        quien = requien;
        System.out.println("HILO DE TRANSMITTER CREADO: " + minombre);
        hilo.start();
    }

    /**
     * METODO: SETPDU()
     * OBJETIVO: ASIGNA EL PDU.
     * INVOCADO: POR SEND(comuni),
     */
    synchronized void setPdu(Pdu p)
    {
        unpdu = p;
    }

    /**
     * METODO: SIT()
     * OBJETIVO: SI ESA POSICION DEL ARREGLO DE PDU ESTA VACIA, ESPERA.
     * INVOCADO: POR RUN(),
     */
    synchronized void sit()
    {
        while(unpdu == null)
        {
            try
            {
                wait();
            } catch (InterruptedException iw)

```

```
{
;
}
}
```

```
/******
METODO: WAKE()
OBJETIVO: HACE UNA NOTIFICACION Y SE MANTENDRA DADO QUE ESTE
          DESATA SIT().
INVOCADO: POR SEND(comuni),
*****/
synchronized void wake()
{
    notify();
}
```

```
/******
METODO: INTERRUPTME()
OBJETIVO: ESTO INTERRUMPE EL HILO, SI ESTA DURMIENDO(SLEEP), NO
          ESPERANDO (WAIT) ESTO ENVIA UN KICK
INVOCADO: POR LLENALO(SEQ),
*****/
void interruptMe()
{
    hilito.interrupt();
}
```

```
/******
METODO: RUN()
OBJETIVO: CORRE EL HILO, LO SICRONIZA E INICIA LA TRANSMICION.
INVOCADO: POR TRANSMITTER(NOMBRE)
*****/
public void run()
{
    while(true)
    {
        sit();                // HACE UNA ESPERA
        synchronized(this)
        {
            if(unpdu != null)
            {
                unpdu.transmit(quien);    // A TRANSMITIR
                unpdu = null;            // YO DIRIA ESTO SOLO UNA VEZ....
            }
        }
    }
}
```

```

        }
        }
        hilito.yield();
    }
}

/*****/
public String toString()
{
    return getClass().getName()+ "["+ "NOMBRE = "+ minombre+ "];"
}

/*****/
public void destroy()
{
    hilito.destroy();
}

} // FIN DE LA CLASE TRANSMITTER

//=====
// CLASE          : VARBIND
// AUTOR           : CARLOS GUIZAR
// VERSION ORIGINAL: TIM PANTON
//
//=====

package snmp.stack;
public class varbind extends Object
{
    private AsnObjectId nombre;
    private AsnObject valor;

    /*****/
    public varbind(varbind varenlazada)
    {
        nombre = varenlazada.nombre;
        valor = varenlazada.valor;
    }

    /*****/
    METODO:  OBT_OID()
    OBJETIVO:  RETORNA EL NOMBRE DEL OID.
    INVOCADO:  POR ENCODESNMP()
    /*****/
    public AsnObjectId obt_Oid()

```

```
{
    return nombre;
}
```

```
/******
METODO:  OBT_VALOR()
OBJETIVO:  RETORNA EL VALOR DEL OID. (AsnNull, Int, entre otros)
INVOCADO:  POR ENCODESNMP(), NEW_VALUE()de OneIntPdu
*****/
public AsnObject obt_valor()
{
    System.out.println("----- (1),VALOR: "+valor);
    return valor;
}
```

```
/******
METODO:  SET_VALUE(vb)
OBJETIVO:  OBTIENE EL NOMBRE Y VALOR DEL OBJETO EN LA SECUENCIA
INVOCADO:  POR LLENALO(SEQ),
*****/
Object set_value(AsnSequence vb)
{
    nombre = (AsnObjectId) vb.getObj(0);
    valor = vb.getObj(1); //TOMA EL QUE LLEGA EN LA SECUENCIA DEL AGENTE
    System.out.println("----- (2),VALOR: "+valor);
    return valor;
}
```

```
/******
METODO:  VARBIND(OID,REQUIEN)
OBJETIVO:  CREA UN NOMBRE CON EL OID RECIBIDO Y UN VALOR NULO.
INVOCADO:  POR ADD_OID(oid),
*****/
public varbind(String Oid, int requien)
{
    if(requien == 13)
    {
        nombre = new AsnObjectId(Oid);
        valor = new AsnNull();
    }
    if(requien == 6)
    {
        nombre = new AsnObjectId(Oid)
        valor = new AsnInteger(369);
    }
}
```

```
        System.out.println(">>>EN VARBIND<<<, EL NOMBRE ES: "+nombre+" EL  
            VALOR ES: "+valor);  
    }  
  
    /*****  
    public varbind(String Oid, AsnObject val)  
    {  
        nombre = new AsnObjectId(Oid);  
        valor = val;  
    }  
  
    /*****  
    public String toString()  
    {  
        return (new String(nombre.toString() + ": " + valor.toString()));  
    }  
  
} // FIN DE LA CLASE VARBIND
```

APENDICE D

OBJETOS DE LA MIB

GRUPOS DE OBJETOS DE LA MIB

Las figuras en este apéndice son dibujadas para ayudar a entender la organización de la MIB. Como tal, en lugar de solo listar cada objeto con una breve descripción, cada grupo es dibujado con su lista subordinada de objetos con indentaciones jerárquicas de izquierda a derecha de la página. Donde existen tablas en la MIB, se pueden apreciar en estas figuras. Este enfoque ayudará al lector a entender cómo la composición de la MIB puede ser usada para crear una Base de Datos Lógicamente Jerárquica.

Los grupos de la MIB son listados en las siguientes figuras:

- Grupo de Sistema en la figura D-1
- Grupo de Interfaces en la figura D-2
- Grupo de Traducción de Direcciones en la figura D-3
- Grupo IP en la figura D-4
- Grupo ICMP en la figura D-5
- Grupo TCP en la figura D-6
- Grupo UDP en la figura D-7
- Grupo EGP en la figura D-8
- Grupo SNMP en la figura D-9

Grupo de Sistema. El grupo de sistema contiene siete objetos. Su propósito es proporcionar información general acerca de un objeto administrado, por ejemplo, un agente. El grupo de sistema debe ser implementado por todos los objetos, aunque no todos los objetos tienen valores. La figura D-1 muestra la estructura de dicho grupo. Los objetos en este grupo son usados para ejecutar las siguientes funciones:

```
System (mib -2 1)
|___ sysDescr (1)
|___ sysObjectID (2)
|___ sysUpTime (3)
|___ sysContact (4)
|___ sysName (5)
|___ sysLocation (6)
|___ sysServices (7)
```

Figura D-1. Grupo de Sistema

- *sysDescr*: Cadena para describir el objeto, tal como hardware, sistema operativo, etc.
- *sysObjectID*: Identificador de objeto (*OID*) para identificar de manera única el objeto (con valores de jerarquía de nombres) dentro del árbol de nombres.
- *sysUpTime*: Marca de tiempo (*TimeTicks*), con el tiempo desde que el objeto fue iniciado o esta corriendo (reinicializado).
- *sysContact*: Cadena para identificar a la persona y/u organización a contactar para información acerca del objeto.
- *sysName*: Cadena para identificar un nombre (usualmente una persona) que es responsable del objeto.
- *sysLocation*: Cadena que contiene la ubicación del objeto.
- *sysServices*: Entero para describir el servicio o los servicios ofrecidos por el objeto, basado sobre la localización del servicio dentro de una capa.

Grupo de Interfaces. Este grupo es también obligatorio por todos los sistemas. Su propósito es proporcionar información sobre una o más interfaces del objeto. Éste describe el tipo de interfaz, tal como SDLC o Ethernet, etc., y proporcionan estadísticas sobre las operaciones ocurridas en la interfaz. Además proporciona descriptores adicionales.

La figura D-2 muestra la estructura del grupo de interfaces de la MIB. La cuál consiste de un entrada individual y una tabla de 22 columnas. Cada renglón de la tabla contiene información acerca de un interfaz. Consecuentemente, un dispositivo con múltiples interfaces tendrán múltiples renglones de entrada. Las entradas individuales de la interfaz MIB son usadas por los siguientes servicios y funciones:

Interface (mib -2 2)

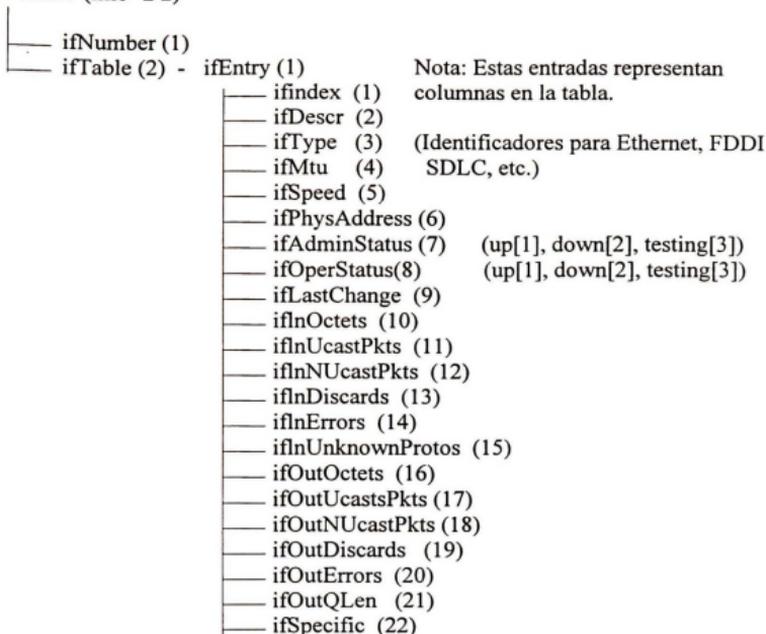
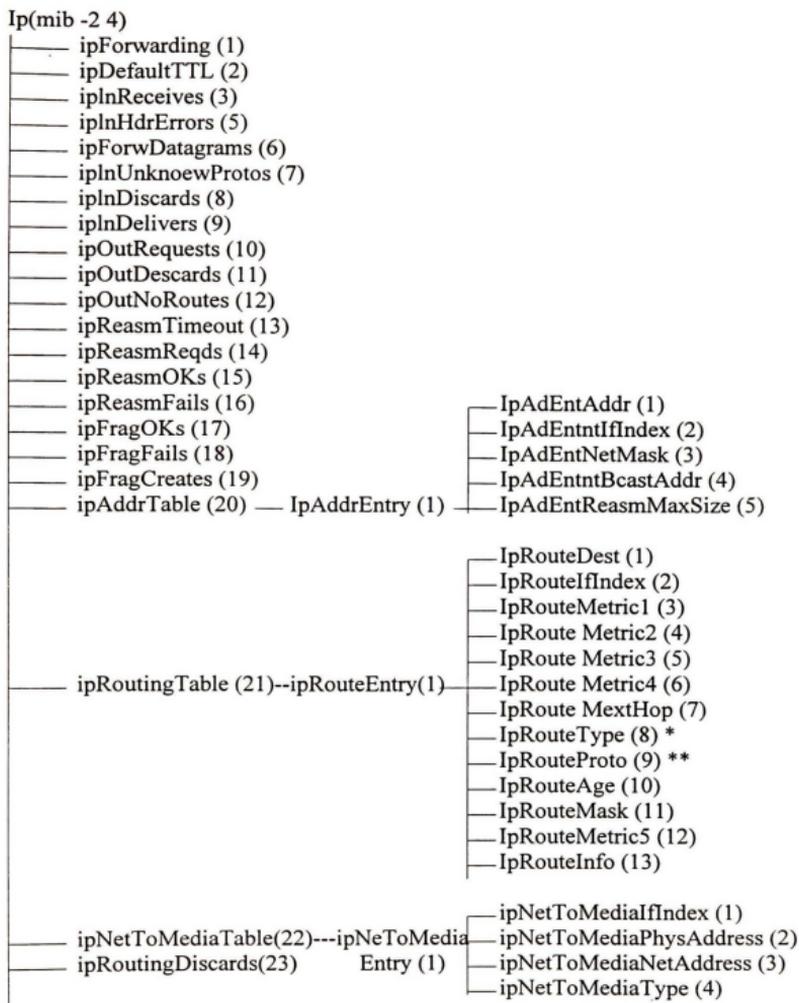


Figura D-2. Grupo de Interfaces.

- *ifNumber*: Valor conteniendo el número de interfaces de red en el sistema.
- *ifIndex*: Valor no ambiguo para cada interfaz, su valor puede tener un rango desde 1 al valor de *ifNumber*.
- *ifDescr*: Información textual que describe la interfaz, típicamente el nombre del producto, el nombre del fabricante y el número de versión de la interfaz.
- *ifType*: Identifica el tipo específico de interfaz y actualmente contiene valores para identificar uno de las 32 tipos de interfaces tales como Ethernet, SDLC, Interfaz de Datos Distribuidos por Fibra (FDDI), etc.
- *ifMtu*: Tamaño máximo del la Unidad de Dato de Protocolo (PDU) que puede ser servido en esta interfaz, especificada en octetos.

- *ifSpeed*: Valor de la capacidad de velocidad de la interfaz en Bits por segundo, si es relevante.
- *ifPhysAddress*: Dirección de la interfaz inmediatamente abajo de la capa de red en una pila típica de protocolo, normalmente es una dirección de IEEE MAC para una interfaz LAN.
- *ifAdminStatus*: Estado de la interfaz, el cual puede ser up = 1, down = 2, o testing = 3.
- *ifOperStatus*: Estado operacional de la interfaz, el cual puede ser up = 1, down = 2, o testing = 3.
- *ifLastChange*: Tiempo en que la interfaz entra a su estado operacional actual, en marcas de tiempo.
- *ifInOctets*: Número total de octetos recibidos en esta interfaz desde la última reinicialización.
- *ifUcastPkts*: Número de paquetes enviado a la siguiente capa del protocolo, sólo paquetes sin transmisión (nonbroadcast).
- *ifInNUcastPkts*: Número de transmisiones o multitransmisión de PDUs enviados a la siguiente capa más alta de protocolo.
- *ifInDiscards*: Número de PDUs descartados en esta interfaz y no enviados a la capa más alta del protocolo. Sin que hayan ocurrido errores pero los PDUs hayan sido descartados por otras razones (por ejemplo problemas de espacio en un buffer).
- *ifInErros*: Número de PDUs que contienen errores, fueron mal formateados, ilegibles, etc. (No enviados a la siguiente capa más alta del protocolo).
- *ifInUnkownProtos*: Número de PDUs descartados porque el PDU es relacionado con un protocolo no soportado o desconocido.
- *ifOutOctets*: Número total de octetos transmitidos en esta interfaz desde la última reinicialización.
- *ifOutUcastPkts*: Número total de PDUs que la siguiente capa más alta del protocolo requirió esta interfaz para transmitir a una dirección no multitransmisión o una dirección no transmisora. Incluye PDUs los cuales pueden haber sido descartados o de otra manera no enviados.
- *ifOutNUcastPkts*: Número total de PDUs que la siguiente capa más alta del protocolo requirió para enviar sobre un transmisor básico o una multitransmisión básica, incluyendo los PDUs que fueron descartados o de otra manera no enviados.



* (Other[1],invalid[2],direct[3],indirec[4])

** (14 entradas, EGP, Rip, OSFP entre otros)

Figura D-4. Grupo IP.

- *ipForwarding*: Indica si esta máquina esta actuando como una compuerta adelantada para datagramas que están llegando.
- *ipDefaultTTL*: Contiene el valor por “default” para el campo tiempo-de-vida en los datagramas IP originados en esta máquina.
- *ipInReceives*: Total de datagramas recibidos de las capas inferiores, incluyendo datagramas erróneos.
- *ipInHdrErros*: Total de datagramas IP descartados debido a errores en los campos de la cabecera de los datagramas.
- *ipInAddrErrors*: Total de datagramas descartados a causa de que la dirección IP fue inválida.
- *ipForwDatagrams*: Indica el número total de datagramas ingresados cuyo destino final no fue la máquina que los recibio. Consecuentemente, no fue hecho el intento para enviarlos a través de la red.
- *ipInDiscards*: Valor del total de datagramas ingresados que fueron descartados, además de que no hayan existido problemas (por ejemplo, problemas de espacio en el buffer)
- *ipInDelivers*: Número total de datagramas ingresados que fueron entregados a otros protocolos, incluyendo protocolos de capa superior y protocolos acompañantes tales como ICMP.
- *IpOutRequests*: Número total de datagramas IP requeridos para ser transmitidos desde el módulo IP de protocolos de capa superior o protocolos acompañantes.
- *ipOutNoRoutes*: Número total de datagramas descartados por que el destino no pudo ser encontrado o las compuertas no estuvieron en estado disponible.
- *ipReasmTimeout*: Número total de segundos en que llegan los fragmentos de datagramas recibidos que esperan ser reensamblados.
- *ipReasmReqds*: Número total de fragmentos que fueron reensamblados en el módulo IP.
- *ipReasmOKs*: Número total de datagramas IP que fueron reensamblados exitosamente.
- *ipReasmFails*: Número total de fallas de reensamble detectadas durante el proceso de reensamble.
- *ipFragOks*: Número total de datagramas que fueron exitosamente fragmentados.
- *ipFragFails*: Número total de datagramas que fueron descartados porque los indicadores de fragmentación no fueron colocados.

- *ipFragCreates*: Número total de fragmentos de datagramas IP generados en este módulo IP.

La primera tabla en el grupo IP es llamada la *ipAddrTable*; ésta consiste de cinco columnas, sus objetos son:

- *ipAdEntAddr*: Valor de la dirección IP para el módulo IP.
- *ipAdEntIfIndex*: Valor índice pertinente para la interfaz; es el mismo valor identificado por *ifIndex* en el grupo de interfaz.
- *ipAdEntNetMask*: Contiene el valor asociado a la máscara de red con la dirección IP para el módulo.
- *ipAdEntBcastAddr*: Contiene el valor del último bit significativo en la dirección de transmisión IP para el módulo IP.
- *ipAdEntReasmMaxSize*: Tamaño del datagrama más largo que puede ser manejado por el módulo IP.

La segunda tabla en el grupo IP es el la tabla de rutas IP. Cada renglón consiste de 13 columnas. Cada renglón contiene una entrada para cada ruta conocida para el módulo. Los objetos en la tabla de ruta IP son usados para realizar lo siguiente:

- *ipRouteDest*: Contiene la dirección IP destino para esta ruta.
- *ipRouteIfIndex*: Interfaz física que contiene el siguiente salto que puede ser alcanzado, el puerto de comunicaciones. Este valor es el mismo que el objeto *ifIndex* en el grupo de interfaces.
- *ipRouteMetric* (1-4): Contiene la métrica de ruteo usada por la ruta [números de saltos, tipo de servicios (TOS), etc.]. Los valores dependen del protocolo de ruteo específicamente utilizado.
- *ipRouteNextHop*: Contiene la dirección IP del siguiente salto de este módulo IP.
- *ipRouteType*: Contiene other = 1, invalid = 2, direc = 3, indirect = 4.
- *ipRouteProto*: Contiene 1 de 14 entradas para describir el protocolo de ruteo utilizado para descubrir la ruta ((EGP), open shortest path first (OSPF), etc.).
- *ipRouteAge*: Valor en número de segundos, desde que la ruta fue actualizada.

- *ipRouteMask*: Contiene la máscara de subred para la ruta.
- *ipRouteInfo*: Una referencia a las definiciones MIB relacionadas con el protocolo de ruteo.

La última tabla en el grupo IP es la tabla de traducción de direcciones IP. Esta contiene cuatro columnas y es utilizada para proporcionar traducción de direcciones físicas y direcciones de red. Es utilizada en conjunto con ARP y proxy ARP para mapeo de direcciones. Las columnas en esta tabla son utilizadas para desempeñar los siguientes objetos:

- *ipNetToMediaIfIndex*: Contiene el valor del número de interfaz, el cual es el mismo valor del objeto *IfIndex* en el grupo de interfaces.
- *ipNetToMediaPhysAddress*: Contiene la dirección física (MAC) de la interfaz.
- *ipNetToMediaNetAddress*: Contiene la dirección IP asociada a la dirección física.
- *ipNetToMediaType*: Describe el tipo de mapeo usado para la traducción de direcciones.
- *ipRoutingDiscards*: Número de entradas de ruteo descartadas.

Grupo ICMP. Utilizado para reportar las operaciones del módulo ICMP en un **host** o compuerta. Como se muestra en la figura D-5, este grupo no contiene tablas, sino objetos escalares. Los objetos desempeñan las siguientes funciones.

- *icmpInMsgs*: Contiene una cuenta del número total de mensajes ICMP recibidos por el módulo ICMP.
- *icmpInErrors*: Registra el número de mensajes ICMP con error.
- *icmpInDestUnreachs*: Contiene el número total de mensajes que no pueden ser entregados porque el destino no fue alcanzado.
- *icmpInTimeExcds*: Contiene un valor representando el número total de mensajes recibidos que excedieron su tiempo de vida.

Icmp (mib - 2 5)

- icmpInMsgs (1)
- icmpInErrors (2)
- icmpInDestUnreachs (3)
- icmpInTimeExcds (4)
- icmpInParmProbs (5)
- icmpInSrcQuenchs (6)
- icmpInRedirects (7)
- icmpInEchos (8)
- icmpInEchoReps (9)
- icmpInTimestamps (10)
- icmpInTimestampRep (11)
- icmpInAddrMasks (12)
- icmpInAddrMaskReps(13)
- icmpOutMsgs (14)
- icmpOutErrors (15)
- icmpOutDestUnreachs (16)
- icmpOutTimeExcds (17)
- icmpOutParmProbs (18)
- icmpOutSrcQuenchs (19)
- icmpOutRedirects (20)
- icmpOutEchos (21)
- icmpOutEchoReps (22)
- icmpOutTimestamps (23)
- icmpOutTimestampsReps (24)
- icmpOutAddrMasks (25)
- icmpOutAddrMaskReps (26)

Figura D-5. Grupo ICMP.

- *icmpInParmProgs*: Registra el número total de mensajes recibidos en los cuales hubo problemas en los parámetros.
- *icmpInSrcQuenchs*: Contiene el valor del número total de fuentes suprimidas de mensajes recibidos.
- *icmpInRedirects*: Refleja el número total de mensajes redireccionados recibidos por el módulo ICMP.

- *icmpInEchos*: Contiene el valor del número total del eco de peticiones de mensajes, recibidos por el módulo ICMP.
- *icmpInEchoReps*: Contiene el valor del número total del eco de respuestas ICMP de los mensajes recibidos por este módulo.
- *icmpInTimestamps*: Representa el valor del número total de estampas de tiempo ICMP en la petición de mensajes recibidos por este módulo.
- *icmpInTimestampReps*: Representa el valor del total de las estampas de tiempo ICMP de los mensajes de respuesta recibidos por este módulo.
- *icmpInAddrMasks*: Representa el número total de mensajes recibidos en la máscara de dirección por este módulo.
- *icmpInAddrMaskRpes*: Representa el número total de mensajes recibidos en la máscara de direcciones de ICMP por este módulo.

Como se indica en la figura D-5., un comparable conjunto de grupos de objetos son utilizados para reportar el número de mensajes enviados por el módulo ICMP.

Grupo TCP. Utilizado para recoger estadísticas y reportes de información sobre cada conexión TCP en un módulo TCP. Éste grupo TCP consiste de varios valores escalares y una tabla (vea la figura D-6). El grupo TCP desempeña las siguientes funciones:

- *tcpRtoAlgorithm*: Identifica el algoritmo utilizado para establecer el valor de timeout de retransmisión de un tráfico desconocido.
- *tcpRtoMin*: Contiene un valor mínimo para el módulo TCP relacionado al timeout de retransmisión.
- *tcpRtoMax*: Complemento para *tcpRtoMin*, contiene el valor máximo de timeout para el módulo TCP.

Tcp (MIB -2 6)

- tcpRtoAlgorithm (1)
- tcpRtoMin (2)
- tcpRtoMax (3)
- tcpMaxConn (4)
- tcpActiveOpens (5)
- tcpPassiveOpens (6)
- tcpAttemptFails (7)
- tcpEstbResets (8)
- tcpCurrEstab (9)
- tcpInSegs (10)
- tcpOutSegs (11)
- tcpRetransSegs (12)
- tcpConnTable(13) — tcpConnEntry (1)
- tcpInErrs (14)
- tcpOutRsts (15)

Nota: Estas entradas representan columnas en esta tabla.
(Los 11 estados de TCP: Closed[1],Listen[2],etc.)

- tcpConnSate (1)
- tcpConnLocalAddress (2)
- tcpConnLocalProt (3)
- tcpConnRemAddress (4)
- tcpConnRemPort (5)

Figura D-6. Grupo TCP.

- *tcpMaxConn*: Establece el número máximo de conexiones que el módulo puede soportar.
- *tcpActiveOpens*: Registra el número de actividades abiertas que han sido soportadas por esta entidad.
- *tcpPassiveOpens*: Refleja el número de aperturas pasivas soportadas por esta entidad.
- *tcpAttemptFails*: Contiene el número de intentos de conexiones fallidos.
- *tcpEstabResets*: Registra las inicializaciones que han ocurrido en esta entidad.
- *tcpCurrEstab*: Registra el número de conexiones TCP en el cual el estado es CLOSE-WAIT o ESTABLISHED.
- *tcpInSeg*: Representa el número total de paquetes recibidos por esta entidad.
- *tcpOutsegs*: Registra el número total de paquetes enviados por esta entidad.
- *tcpRetransSegs*: Registra el número total de paquetes retransmitidos.

- *tcpConnTable*: Tabla para el grupo TCP. Esta contiene cinco columnas las cuales son representadas por los siguientes objetos:
 - *tcpConnState*: Describe el estado de la conexión TCP. Los siguientes estados son permitidos sobre un módulo TCP: 1 = closed, 2 = listen, 3 = synSent, 4 = synReceived, 5=established, 6 = finWait1, 7 = finWait2, 8 = closed Wait, 9 = lastAck, 10 = closing, 11 = timeWait, 12 = deleteTCB.
 - *tcpConnLocalAddress*: Contiene la dirección IP para la conexión.
 - *tcpConnLocalPort*: Contiene el número de puerto local de la conexión.
 - *tcpConnRemAddress*: Contiene una dirección IP remota para la conexión.
 - *tcpConnRemPort*: Contiene el número de puerto remoto para la conexión TCP.
- *tcpInErrs*: Registra el número total de paquetes recibidos en los cuales hubo un típico error, paquetes que revelan un chequeo de suma erróneo.
- *tcpOutRsts*: Contiene el número de paquetes TCP enviados los cuales tienen la bandera RST prendida.

Grupo UDP. Consiste de cuatro entradas escalares y una tabla (vea la figura D-7). El número pequeño de objetos administrados en este grupo refleja la naturaleza sin conexión de UDP. El grupo UDP es usado para desempeñar los siguientes servicios:

- *udpInDatagrams*: Registra el número total de datagramas UDP que han sido entregados a los usuarios por este módulo UDP.
- *udpNoPorts*: Registra el número de datagramas UDP que fueron recibidos en cualquier aplicación que no pudo ser encontrada en el puerto destino.
- *udpInErrors*: Registra el número total de datagramas UDP recibidos que no pudieron ser entregados a causa de problemas.
- *udpOutDatagrams*: Contienen el número total de datagramas UDP enviados desde la entidad.

La única tabla en este grupo es la tabla escuchante UDP. Ésta contiene dos columnas las cuales desempeñan los siguientes servicios:

- *udpLocalAddress*: Contiene la dirección local para el escuchante UDP.

- *udpLocalPort*: Contiene el número del puerto local para el escuchante UDP.

Udp (MIB - 2 7)

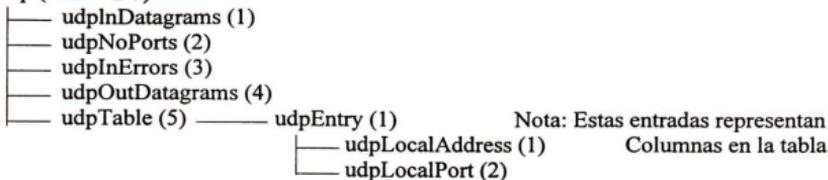


Figura D-7. Grupo UDP.

Grupo EGP. Utilizado para registrar y reportar información acerca de operaciones en el módulo EGP. (Vea la figura D-8). Consiste de cuatro valores escalares y un tabla de 15 columnas. El grupo EGP es usado para desempeñar los siguientes servicios:

- *egpInMsgs*: Registra el número total de mensajes EGP libres de error en el módulo.
- *egpInErrors*: Registra el número de mensajes recibidos los cuales contienen un error.
- *egpOutMsgs*: Registra el número total de mensajes generados por el módulo EGP.
- *egpNeighTable*: Esta es la tabla EGP, la tabla vecina EGP. Esta es usada para contener información (lógicamente suficiente) acerca de módulos EGP vecinos. Esta contiene los siguientes objetos:
 - *egpNeighState*: Describe el estado del vecino específico, los estados pueden ser: **idle, acquisition, down, up, cease**.
 - *EgpNeighAddr*: Contiene la dirección IP para un vecino específico.

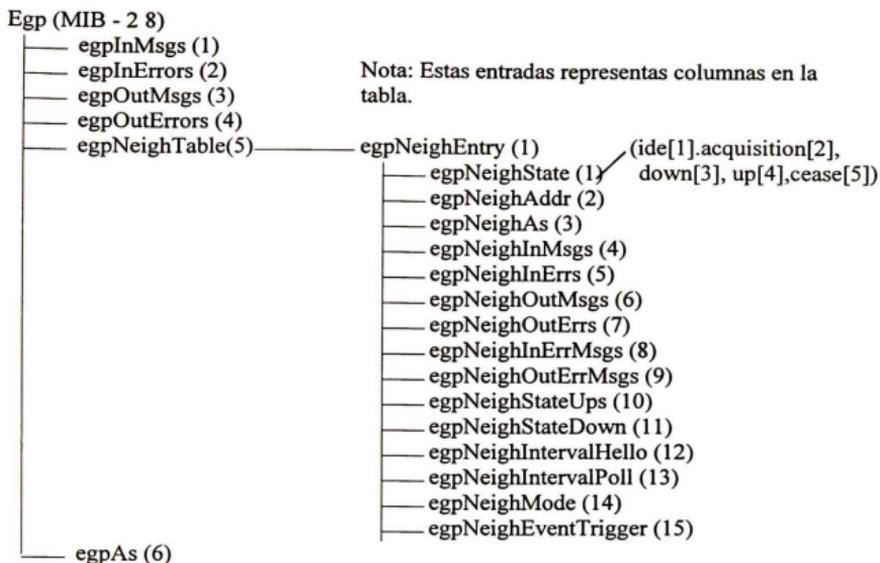


Figura D-8. Grupo EGP.

- *egpNeighAs*: Número del sistema autónomo para el vecino EGP.
- *egpNeighInMsgs*: Registro del número de mensajes recibidos sin problemas desde un vecino.
- *egpNeighInErrs*: Registra el número de mensajes recibidos desde los vecinos en los cuales hubo un error.
- *egpNeighOutMsgs*: Número de mensajes EGP generados por el módulo local EGP para un vecino.
- *egpNeighOutErrs*: Registra el número de mensajes que no fueron enviados a los vecinos a causa de problemas.
- *egpNeighInErrMsg*: Contiene el número de mensajes recibidos desde un vecino.

- *egpNeighOutErrMsgs*: Contiene el número de mensajes EGP que fueron definidos estando en error y se enviarán al vecino.
- *egpNeighStatesUps*: Registra el número de estados de transición EGP para un vecino.
- *egpNeighStateDown*: Registra el número de estados de transición para un vecino.
- *egpNeighInervalHello*: Contiene el límite en cuanto a mensajes enviados de hola.
- *egpNeighIntervalPoll*: Determina el intervalo entre mensajes poll EGP sucesivos.
- *egpNeighMode*: Describe el modo de preguntar del vecino. Éste es activo o pasivo.
- *egpNeighEventTrigger*: Utilizado para controlar operaciones iniciadas y detener eventos.
- *egpAs*: Contiene el valor del número del sistema autónomo para el módulo EGP.

Grupo SNMP. Proporciona información acerca de los objetos SNMP (vea la figura. D-9), principalmente estadísticas relacionadas con el tráfico y problemas o errores de condición. Todos estos objetos tiene sintaxis de tipo *Counter* con excepción de la última entrada la cual es un entero. (*SnmplibEnableAuthenTraps*).

- *snmpInPkts*: Número de paquetes recibidos de la capa inferior a SNMP.
- *snmpOutPkts*: Identifica el número de paquetes recibidos SNMP con dirección hacia la capa inferior.
- *snmpInBadVersions*: Número de PDUs recibidos con un versión errónea.
- *snmpInBadCommunityNames*: Número de PDUs recibidos con nombres de comunidad sin identificación o sin autorización.

snmp (MIB - 2 1 1)

- snmpInPkts (1)
- snmpOutPkts (2)
- snmpInBadVersions (3)
- snmpInBadCommunity (4)
- snmpInBadCommunityUses (5)
- snmpInASNParseErrs (6)
- snmpInBadTypes (7) -- no usados
- snmpInTooBigs (8)
- snmpInNoSuchName (9)
- snmpInBadValues (10)
- snmpInReadOnlys (11)
- snmpInGenErrs (12)
- snmpInTotalRegVars (13)
- snmpTotalSerVars (14)
- snmpInGetRequests (15)
- snmpInGetNexts (16)
- snmpInSetRequests (17)
- snmpInGetresponses (18)
- snmpInTraps (19)
- snmpOutTooBigs (20)
- snmpOutNoSuchNames (21)
- snmpOutBadValues (22)
- snmpUtReadOnlys (23) -- no usado
- snmpOutGenErrs (24)
- snmpOutGetRequests (25)
- snmpOutGetNexts (26)
- snmpOutSetRequests (27)
- snmpOutGetResponses (28)
- snmpOutTraps (29)
- snmpEnableAuthenTraps (30).

Figura D-9. Grupo SNMP.

- *snmpInBadCommunityUses*: Número de usos incorrectos de la comunidad.
- *snmpInASNParseErrs*: Número de PDUs que no pudieron ser analizadas gramaticalmente por objetos ASN.1.
- *snmpInBadTypes*: No usado.
- *snmpInTooBigs*: Número de PDUs recibidos con el campo status error tooBig.
- *snmpInNoSuchNames*: Número de PDUs recibidos con el campo status error del campo NoSuchName.

- *snmpInBadValues*: Número de PDUs recibidos con un valor erroneo en el campo del estado de error.
- *snmpInReadOnlys*: No usado.
- *snmpInGenErrs*: Número del PDUs recibidos con el campo genErr activado.
- *snmpInTotalRegVars*: Número de objetos MIB que han sido recobrados.
- *snmpInTotalSetVars*: Número de objetos MIB que han sido cambiados/alterados.
- *snmpInGetRequests*: \
- snmpInGetNexts*: \ Estos valores son usados para indicar el número de PDUs
- snmpInSetRequests*: / respectivos que fueron recibidos
- snmpInGetResponses*: /
- snmpInTraps*: /
- *snmpOutTooBig*: Número de PDUs enviados con el campo tooBig.
- *snmpOutNoSuchNames*: Número de PDUs enviados con el campo nosuchName.
- *snmpOutBadValues*: Número de PDUs enviados con el campo badValue.
- *snmpOutReadOnlys*: Número de PDUs enviados con el campo readOnly (no usado en tiempo presente).
- *snmpOutGenErrs*: Número de PDUs enviados con el campo genErr.
- *snmpEnableAuthenTraps*: Describe si las traps están habilitadas o deshabilitadas; pueden ser escritas o leídas.
- *snmpOutGetRequests*: \
- snmpOutGetNexts*: \ Estos valores son usados para indicar el número
- snmpOutSetRequests*: / de los respectivos PDUs que fueron enviados.
- snmpOutGetResponses*: /
- snmpOutTraps*: /

GLOSARIO

GLOSARIO

ABSTRACT SYNTAX: (SINTAXIS ABSTRACTA) Una descripción de un tipo de dato que es independiente de estructuras de maquinas orientadas y restricciones.

ABSTRACT SYNTAX NOTATION ONE: (NOTACIÓN DE SINTAXIS ABSTRACTA 1) El lenguaje OSI para la descripción de sintaxis abstracta.

ACCES MODE (SNMPv1): (MODO DE ACCESO SNMPv1) El nivel de autorización implementado para una comunidad SNMP.

ACCES POLICY (SNMPv2): (POLÍTICA DE ACCESO SNMPv2) Las operaciones permitidas cuando una parte pregunta a la otra para desempeñar una operación sobre los objetos en un contexto.

ADDRESS RESOLUTION: (DIRECCIÓN DE RESOLUCIÓN) Un significado por mapeo direcciones de capa de red en direcciones específicas medias

ADDRESS RESOLUTION PROTOCOL: El protocolo en el departamento de Internet de protocolos usados mapea direcciones IP en Ethernet y otras direcciones medias

AGENT: (AGENTE) Observe el agente de administración de red.

AMERICAN NATIONAL STANDARDS INSTITUTE: (INSTITUTO NACIONAL DE ESTÁNDARES AMERICANO) El cuerpo de estandarización nacional de estados unidos. ANSI es un miembro de ISO.

APPLICATION SERVICES: (SERVICIOS DE *APLICACIÓN*) Los servicios colectivos ofrecidos bajo la capa 4 del modelo OSI.

AUTHENTICATION: (AUTENTIFICACION) proceso por medio del cual un mensaje es asociado con una particular entidad originada.

AUTHENTICATION ENTITY(SNMPv1): (ENTIDAD DE AUTENTIFICACION SNMPv1) Esa porción de un agente SNMP responsable para la verificación de una entidad SNMP es miembro de la comunidad el cual por costumbre pertenece. Esta porción del agente es también responsable de la codificación y descodificación de mensajes SNMP acordados al algoritmo de autenticación de una comunidad dada.

BASIC ENCODING RULES: (REGLAS BASICAS DE CODIFICACIÓN) El lenguaje OSI para describir sintaxis de transferencia.

BROADCAST ADDRESS: Un medio específico o una dirección IP referida a todas las estaciones en un medio.

BROADCASTING: El acto de enviar a todas las direcciones.

CMIP OVER TCP: Un mapeo del paquete de información de la *administración* de red OSI a la administración de red basada en la Suite Internet de protocolos.

CMOT: Vea CMIP sobre TCP.

CO-MODE: Vea modo orientado a conexión.

COMMON MANAGEMENT INFORMATION PROTOCOL (CMIP): El protocolo OSI para administración de red.

COMMON MANAGEMENT INFORMATION SERVICE (CMIS): El servicio ofrecido por CMIP.

COMMON MANAGEMENT INFORMATION SERVICE ELEMENT (CMISE): El servicio elemental responsable para el intercambio de información administrada de red.

COMMUNITY: Una relación administrativa entre entidades SNMP.

COMMUNITY PROFILE: Esa porción de los objetos administrados en un agente que un miembro de la comunidad es alojado para manipular.

CONNECTION: Un enlace lógico entre dos o *más* usuarios de un servicio.

CONNECTION-LESS MODE: (MODO DE MENOR CONEXIÓN) Un servicio que tiene una fase sencilla envolviendo mecanismos de control tales como direccionamiento en adición para transferencia de datos.

CONNECTION-ORIENTED MODE:(MODO ORIENTADO A CONEXION) Un servicio que tiene tres distintas fases: Establecer, en el cual dos o *más* usuarios son unidos a una conexión; Transferencia de datos, en el cual los datos son intercambiados entre usuarios; y, Liberación, en la cual el enlace es finalizado.

CONS: Servicio de red orientado a conexión.

CONTEXT: (SNPv2) Una colección de recursos de objetos.

COTS: Servicio de transporte orientado a conexión.

DARPA: Agencia de proyectos de investigación avanzada para la defensa.

-
- DATAGARAM:** El mismo contiene unidad de datos transmitidos independientemente de otros datagramas.
- DEFAULT ROUTE:** Cuando enviamos un datagrama IP, entra en la tabla de *enrutamiento* la cual es usada si ninguna otra ruta es apropiada.
- DEFENSE ADVANCED RESEARCH PROJECTS AGENCY:** Una agencia del departamento de defensa de los estados unidos que realiza investigación de alto riesgo.
- DEVICE:** *Algún* elemento de red.
- DIRECT ROUTING:** El proceso de enviar un datagrama IP cuando el destino reside en la misma red IP(o subred IP) a enviar.
- DOMAIN:** Una entidad administrativa responsable de nombrar entidades.
- DOMAIN NAME SYSTEM:** El protocolo de *aplicación* ofrece nombramientos de servicio en la Suite Internet de protocolos.
- EGP:** Vea protocolo de compuerta exterior.
- END-SYSTEM:** Un dispositivo de red ejecutando funciones de todas las capas del modelo OSI. Los sistemas terminales son comunmente pensados como aplicaciones.
- END-TO-END SERVICES:** Servicios de terminal a terminal. Los servicios colectivamente ofrecidos por las tres capas inferiores del modelo OSI.
- ENTERPRICE MIB:** Un módulo MIB definido en la porción específica de empresa de espacio de administración Internet.
- EXTERIOR GATEWAY PROTOCOL:** Protocolo de rehabilitación usado por enrutadores en un nivel dos de Internet.
- EXPERIMENTAL MIB:** Un modulo MIB definido en la porción experimental de un espacio de administración Internet.
- EXTERNAL DATA REPRESENTATION:** Una sintaxis de transferencia definida por SUN Microsystems, Inc.
- FILE TRANSFER PROTOCOL:** (FTP) El protocolo de aplicación que ofrece servicios de archivos en sitios Internet.
- FRAGMENT:** Un datagrama IP que contiene solo una porción de los datos del usuario de un datagrama IP largo.

FRAGMENTATION: El proceso de romper un datagrama IP en partes pequeñas, tal que cada fragmento puede ser transmitido completo en un medio físico dado.

FULLY - QUALIFIED DOMAIN NAME: Un nombre de dominio que contiene la ruta completa de etiquetas a la raíz del árbol nombrado.

GATEWAY: Un enrutador, también, (uso impreciso) una entidad responsable para mapeos complejos, usualmente en la capa de aplicación.

HOST: Un sistema terminal.

HOST-IDENTIFIER: La porción de una dirección IP identificando un HOST en la red IP.

HOST - NUMBER: La porción de una dirección IP de subred que identifica un número de HOST en una subred.

ICMP: Protocolo de control de mensajes en Internet.

INSTANCE - IDENTIFIER: Un medio de identificar una estancia de un tipo de objeto en particular.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: Es la organización que produce mucho de los estándares mundiales. OSI es solo una de las muchas áreas de estandarización por ISO/IEC.

INTERNET: Una red en el sentido OSI; históricamente terminada como un conjunto concatenado de redes. La Internet, es la red internacional *más* grande que existe.

INTERNET ARCHITECTURE BOARD: El cuerpo técnico sobre el cual se desarrolla la Suite Internet de protocolos.

INTERNET ASSIGNED NUMBERS AUTHORITY: La entidad responsable para asignar números en la Suite Internet de protocolos.

INTERNET CONTROL MESSAGE PROTOCOL: Un reporte simple del protocolo para IP.

INTERNET PROTOCOL: El protocolo de red que ofrece un modo de conexión al servicio de red en la Suite Internet de protocolos.

INTERNET STANDARD MIB: RFC 1213.

INTERNET STANDARD NETWORK MANAGEMENT FRAMEWORK: RFCs 1155,1212,1213 y 1157.

IP ADDRESS: 32 bit, usados para representar un punto de ubicación en un Internet.

IP CONNECTED: El subconjunto de la comunidad Internet que cambia el tráfico basado en IP.

ISO/IEC: Vea Organización Internacional para la estandarización.

ISODE: Entorno de desarrollo de ISO.

LEAF OBJECT: Un tipo de objeto definido por el cual no hay objetos definidos con un nombre asignado subordinado en particular, tablas y filas no son objetos visores.

LEXICOGRAPHIC ORDERING: Una comparación de esquema.

LOCAL AREA NETWORK: Cualquiera de un numero de tecnologías a condición de alta velocidad, transferencia de estado bajo y siendo limitada en medida geográfica.

MANAGED NODE: Un dispositivo contenido, implementación de un agente administrador de red.

MANAGED OBJECT: Un objeto *administrado*.

MANAGEMENT INFORMATION BASE (MIB): Una colección de objetos que pueden ser accedados vía protocolo administrador de red. vea estructura de información de *administración*.

MANAGEMENT STATION: Estación que administra la red.

MANAGER: Uso impreciso una aplicación residente en una estación de *administración* de red..

MAXIMUN TRANSMISSION UNIT: La *más* larga cantidad de datos de usuario (ejemplo : la *más* larga medida de un datagrama IP) que puede ser enviada en un sólo cuadro(frame) dentro de un medio particular.

MEDIA ADDRESS: La dirección de una interfaz física.

MEDIA DEVICE: Un dispositivo de bajo nivel que no usa un protocolo en la capa de Internet como función primaria.

MIB: Vea base de información administrada.

MIB MODULE: Una colección de relaciones manejadas de objetos definidos.

MIB VIEW: Una colección de objetos manejados realizada por un agente el cual es visible para una aplicación de *administración*.

NAME SERVER: Una entidad cuales mapas nombran a estos atributos asociados.

NAMING AUTHORITY: Una entidad administrativa teniendo la autoridad para asignar nombres dentro de un dominio dado.

NETWORK: Una colección de subredes conectadas por sistemas intermedios y sistemas poblados por sistemas terminales, también (uso de Internet) una sola subred o un recuento de un juego de subredes en el sentido OSI.

NETWORK MANAGEMENT: La tecnología usada para manejar un Internet.

NETWORK MANAGEMENT AGENT: La implementación de un protocolo administrador de red el cual intercambia información de *administración* de red con la estación administradora de red.

NETWORK MANAGEMENT PROTOCOL: El protocolo usado para transportar información *administrada*.

NIST: Instituto Nacional de Estándares y Tecnología.

NMS: Estación de *administración* de red.

OBJECT INSTANCE: Una instancia particular de un tipo de objeto.

OBJECT TYPE: Una *definición* abstracta de un objeto manejado.

OPEN SYSTEM INTERCONNECTION: Un esfuerzo internacional para facilitar comunicaciones entre computadoras de diferente manufactura y tecnología.

PARTY(SNMPv2): Una entidad de comunicación.

PDU: Protocolo de unidad de datos.

PORT NUMBER: Identifica una entidad de aplicación para un servicio de transporte en un sitio Internet de protocolos.

PROTOCOL CONTROL INFORMATION: Conceptualmente la parte inicial de unidad de dato de protocolo usada para un protocolo de maquina para comunicar información o para ser observada.

PROTOCOL DATA UNIT: Un objeto dato intercambiado por protocolo de maquina contiene ambos protocolos control de información y datos de usuario.

PROTOCOL MACHINE: Una maquina de estado finito (FSM) que implementa un protocolo cuando hay una entrada particular (ejemplo solicitud de usuario o actividad de red).ocurre en un estado particular. El FSM potencialmente genera una salida particular (ejemplo indicación de usuario o actividad de red y posibles movimientos a otro estado).

PROTOTYPE: Uso de *administración* .El tipo objeto corresponde a un prototipo o instancia.

PROXY AGENT: Un agente el cual tiene acceso a información de *administración* la cual no tiene localmente y tiene que realizar una interacción no local en orden para satisfacer los requerimientos de *administración* los cuales refieren esa información.

PROXY RELATIONSHIP: Una configuración administrativa en la cual un agente de procuración es usado para tener un acceso remoto de información de *administración*.

REQUEST FOR COMMENTS: La serie de documentos describen el sitio Internet de protocolos y experimentos relacionados.

RETRANSMISSION: El proceso de repetición de envío a una unidad de datos mientras esperan para un reconocimiento

ROUTER: (ENRUTADOR) Un relevador de nivel 3.

SEGMENT: La unidad de intercambio en TCP.

SIMPLE GATEWAY MONITORING PROTOCOL: El predecesor del sencillo protocolo de *administración* de red.

SMTP: (PROTOCOLO DE TRANSFERENCIA SENCILLA DE CORREO) El protocolo de aplicación ofrece manejo de mensajes, servicio en sitios Internet de protocolos.

SIMPLE NETWORK MANAGEMENT PROTOCOL: El protocolo de aplicación ofrece *administración* de red servicios en sitios Internet de protocolos.

SNMP: Protocolo de *administración* simple de red.

SNMP SECURITY: Trabajo inicial el cual fue usado como la base para la seguridad en implementos SNMPv2).

SOCKET: Una función de una dirección IP y un numero de puerto.

STRUCTURE OF MANAGEMENT INFORMATION: (SMI) Las reglas usadas definen los objetos que pueden ser accedidos via protocolo administrador de red.vea base de información de *administración*.

SUBNET: (El *más* desafortunado uso de internet) una red física dentro de una red IP.

SUB-NET NUMBER: Esa porción de un host identificador IP el cual identifica una red física particular dentro de una red IP.

SUBNETWORK: Una simple conexión de red de usuarios, herramientas en simple medio de transmisión virtual.

SUBTREE FAMILY (SNMPv2): Una colección de objetos identificadores los cuales son definidos por un objeto de valor identificador y un bit de *máscara*.

SYSTEM MANAGEMENT: El nombre OSI para administrador de red

TCP: Protocolo de control de transmisión.

TRANSMISSION CONTROL PROTOCOL (TCP): El protocolo de transporte ofrece un servicio de transporte orientado a conexión en sitios Internet de protocolos.

USER DATAGRAM PROTOCOL (UDP): El protocolo de transporte ofrece una conexión modo menor, servicio de transporte en sitios internet de protocolos.

USER DATA: Conceptualmente, la parte de datos de un protocolo, usados para transportar y comunicar información entre los usuarios del protocolo.

VARIABLE BINDING: Uso de SNMP para fusionar un nombre de instancia y valor de asociación.

WIDE AREA NETWORK (WAN): Red *alguna* de dispositivos siempre que se transfieran a distancias muy grandes.

BIBLIOGRAFIA

BIBLIOGRAFIA

- [2-1] Rose, M.T. y K. McCloghrie. "Structure and Identification of Management Information for TCP/IP-based Internets." RFC 1155, Mayo 1990.
- [2-3] J.D. Case, M. Fedor, M.L. Schoffstall, and C. Davin. "Simple Network Management Protocol (SNMP)." RFC 1157, Mayo 1990.
- [2-4] International Organization for Standardization, Information Processing Systems : Open Systems Interconnection, Basic Reference Model-Part 4 : Management Framework, ISO 7498-4 : 1989.
- [2-7] Steedman, Douglas. "Abstract Syntax Notation One (ASN.1), the Tutorial and Reference. Isleworth, Middlesex, UK: Technology Appraisals, Ltd. ISBN 1-871802-06-7, 1990.
- [2-9] Rose, M. y K. McCloghrie. "Concise MIB Definitions." RFC 1212, Marzo 1991.
- [2-11] Perkins, Dave. "How to Read and Use an SNMP MIB." 3TECH, the 3com Technical Journal, Primavera 1991: 31-55.
- [C-1] Simposium Internacional de Computación, I.P.N., del 12 al 14 de Noviembre de 1997.
- [C-2] Semana de Conferencias en Telecomunicaciones, AMICE, del 19 al 23 de Mayo de 1997.
- [C-3] Simposium Internacional MAAPET 96, I.P.N., del 13 al 15 de Noviembre de 1996.
- [C-4] II Congreso Nacional del CIE, del 28 al 30 de Octubre de 1996.
- Gulbransen, David. Y Rawlings kenrick. "Cree sus Applets para Web con Java", Prentice Hall, 1996.
- Arnold, Ken. Y Gosling James. "The Java Programming Lenguaje", Addison-Wesley Publishing, 1996.
- Campione, Mary y Walrath, Kathy. "The Java Tutorial Object-Oriented Programming for the Internet", Addison-Wesley Publishing, 1996.
- Naughton, Patrick. "Manual de Java", McGraw-Hill Series, 1996
- Black, Uyless. "Network Management Standards", McGraw-Hill Series, 1992.

- Rose, Marshall T. "The Simple Book, An Introduction to Internet Management", Prentice Hall, 2nd edition, 1994.
- Townsend, Robert L. "SNMP Application Developer's Guide", VNR Communications Library, 1995.
- Miller, Mark A. "Managing Internetworks With SNMP", M&T Books, 1995.
- Perrow, Graeme S. "The Abstraction and Modelling of Management Agents", University of Western Ontario, London, Ontario, September 1994.
- Divakara K. Udupa. "Network Management Systems Essentials", McGraw-Hill series,
- Steedman, Douglas. "Abstract Syntax Notation One (ASN.1): The Tutorial and Reference", Technology Appraisals, Ltd., 1991.
- Bapat, Subodh. "Object-Oriented Networks : Models for Architecture, Operations, and Management", Prentice Hall, 1994.
- Feit, Sidnie. "SNMP : A Guide to Network Management", McGraw-Hill, 1994.
- Perkins, David T. "Understanding SNMP MIBS", September, 1993.
- Harnedy, Sean J. "Total SNMP : Exploring the Simple Network Management Protocol", CBM Books, 1994.
- G.S. Perrow, J. W. Hong, H.L. Lutfiyya, y M.A. Bauer, "The abstraction and modelling of management agents. Technical report", Department of Computer Science. University of Western Ontario, Toronto, Canada, Mayo 1995.
- J. Case, K. McCloghrie, M. Rose y S. Waldbusser. "Simple network management protocol versión 2 C specification", RFC 1901-1908, January 1996.
- Gray, Robert. Rus, Daniela y Kotz David "Transportable Information Agents", Department of Computer Science, Dartmouth College, January 1996.
- Wies, Rene. "Policies in Network and Systems Management -Formal Definition and Architecture-", Munich Network Management team, University of Munich, March 1994.
- Aguilar, Luis J. "PROGRAMACION ORIENTADA A OBJETOS", McGraw-Hill, 1996.
- Rumbaugh, James. Blaha, Michel. Premerlani, William. "Modelado y Diseño orientados a objetos", Prentice Hall, 1996.

Los abajo firmantes, integrantes de jurado para el examen de grado que sustentará el Ing. Carlos Alberto Guizar Gómez, declaramos que hemos revisado la tesis titulada:

“Una propuesta de agente y de administrador para la administración de redes”, consideramos que cumple con los requisitos para obtener el grado de Maestro en Ciencias, con especialidad en Ingeniería Eléctrica.

Atentamente

Dr. Adriano de Luca Pennacchia



Dr. Héctor Ruíz Barradas



Dr. Uriel Tirado Rios



CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL
INSTITUTO POLITECNICO NACIONAL

BIBLIOTECA DE INGENIERIA ELECTRICA
FECHA DE DEVOLUCION

El lector está obligado a devolver este libro
antes del vencimiento de préstamo señalado
por el último sello.

DEVOLUCION

