Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Laboratorio de Comunicación y Redes

AdminUCV NGN: Aplicación de Código Abierto para la Configuración y la Administración de Redes de Próxima Generación

Trabajo Especial de Grado
presentado ante la Ilustre
Universidad Central de Venezuela
por los Bachilleres:

Kathleen S. Jiménez D.
C.I.: 17.348.487
E-mail: kathleensioux@gmail.com

William E. López L.
C.I.: 17.115.557
E-mail: wellop83@gmail.com

para optar al título de Licenciado en Computación

Tutor: Prof. Eric Gamess

Caracas, Julio 2008

Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Laboratorio de Comunicación y Redes

ACTA DEL VEREDICTO

Quienes suscriben, Miembros del Jurado designados por el Consejo de Escuela de Computación, para examinar el Trabajo Especial de Grado, presentado por los Bachilleres Kathleen S. Jiménez D. C.I.: 17.348.487 y William E. López L. C.I.: 17.115.557, con el título "**AdminUCV NGN: Aplicación de Código Abierto para la Configuración y la Administración de Redes de Próxima Generación**", a los fines de cumplir con el requisito legal para optar al título de Licenciado en Computación, dejan constancia de lo siguiente:

Leído como fue dicho trabajo por cada uno de los Miembros del Jurado, se fijó el día 15 de julio de 2008, a las 3:00 PM, para que sus autores lo defiendan en forma pública, en Planta Alta III de la Escuela de Computación, mediante la exposición oral de su contenido, y luego de la cual respondieron satisfactoriamente a las preguntas que le fueron formuladas por el Jurado, todo ello conforme a lo dispuesto en la Ley de Universidades y demás normativas vigentes de la Universidad Central de Venezuela. Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobarlo.

En fe de lo cual se levanta la presente Acta, en Caracas a los 15 días del mes de julio del año dos mil ocho, dejándose también constancia de que actuó como Coordinador del Jurado el Profesor Tutor Eric Gamess.

Prof. Eric Gamess
(Tutor)

Prof. David Pérez
(Jurado Principal)

Prof. Ana Romero
(Jurado Principal)

AGRADECIMIENTOS

A Dios, por cada una de las oportunidades que me ha brindado a lo largo de mi vida, por darme la fortaleza para superar los obstáculos y permitirme alcanzar cada una de mis metas.

A mis padres, Jenny y Victor Hugo, por ofrecerme su apoyo incondicional ante cualquier situación, por cada palabra de aliento y consejo ante los obstáculos, por estar siempre conmigo, por confiar en mi y por ser mis guías y amigos incomparables. Los ADORO!

A mis tíos, abuelos, primos y demás familiares, por su apoyo, por tener siempre la disposición de ayudarme y alentarme a continuar en cada uno de los proyectos que me he planteado.

Al tutor, Prof. Eric Gamess, por habernos tomado en cuenta para la creación de este proyecto, por su profesionalismo, por su disposición y guía durante el desarrollo del presente Trabajo Especial de Grado.

A mi compañero de Tesis, William, por su gran apoyo, por su dedicación y entrega continua en este proyecto, por brindarme su confianza y por apostar siempre a que obtengamos excelentes resultados.

A mis amigos, por los momentos especiales compartidos, porque me han apoyado siempre y han confiado en mi.

A cada uno de los profesores de la Escuela de Computación, por contribuir en mi formación como profesional y ayudarme a alcanzar este objetivo.

A todas aquellas personas que de alguna manera han contribuido en la creación de este proyecto y que nos han apoyado.

A TODOS, MIL GRACIAS!!!
Kathleen S. Jiménez D.

AGRADECIMIENTOS

Quisiera agradecer a Dios por darme la paciencia y la fuerza necesaria para llevar a cabo mi Trabajo Especial de Grado, así como por todas las oportunidades que me ha dado en la vida y por haber puesto a mi lado a personas tan maravillosas.

A mis padres por apoyarme, especialmente a mi madre por aconsejarme y guiarme en cada una de mis decisiones y por estar a mi lado en todo momento.

A mi abuela Carmen Estela por ser mi segunda madre y darme tanto optimismo y amor.

A mi hermano por estar a mi lado y compartir juntos cada experiencia nueva en la vida.

A mis ti@s: Mireya, Rosa, Negra, Blanca, Pedro y Carlos y a mis prim@s: Mirelvys, Mery, Pedro, Mary, Luis, Karina, Yavanna, Simón, Dayana y Hannah por todo el cariño y el amor que me han dado durante toda la vida.

Al Licenciado Daniel Hernández, por sus sabios consejos y su apoyo incondicional.

A mi compañera de Tesis Kathleen, por su apoyo y por todo el esfuerzo realizado durante el desarrollo de este trabajo.

Al Profesor Eric Gamess, por todo su apoyo y por brindarnos la ayuda necesaria para solucionar los problemas, que se nos plantearon durante la realización de la investigación.

A la UCV y a los Profesores de la escuela de computación por todos los conocimientos que me transmitieron durante estos años.

A mis compañeros del Grupo Docente de Comunicación de Datos y en especial a la Profesora Karima Velásquez por su apoyo durante la carrera.

Finalmente quisiera agradecer a todos mis amigos y amigas por su compañía, consejos y ayuda en esta gran carrera. Y a todos aquellos que de alguna manera contribuyeron con este trabajo.

Gracias a todos.
William E. López L.

RESUMEN

TÍTULO:
AdminUCV NGN: Aplicación de Código Abierto para la Configuración y la Administración de Redes de Próxima Generación.

AUTORES:
Kathleen S. Jiménez D.
William E. López L.

TUTOR:
Prof. Eric Gamess

El presente Trabajo Especial de Grado consiste en el desarrollo de una aplicación denominada AdminUCV NGN para la administración y la configuración de redes con soporte para IPv4 e IPv6. AdminUCV NGN está basada en el protocolo SNMP y provee un conjunto de herramientas que permiten monitorizar y conocer el estado de la red, facilitando la labor del administrador.

El trabajo se inicia con un estudio teórico de los aspectos más relevantes que se deben tomar en cuenta para la creación de un sistema para administración de redes y un análisis de cada uno de los protocolos sobre los que se fundamenta. Durante la implementación de la aplicación, se utilizó una metodología iterativa basada en las nuevas tendencias del modelo de desarrollo ágil que plantea la creación de cada una de las herramientas de la aplicación en una iteración del proceso que abarca las fases de Análisis, Diseño, Implementación y Pruebas.

AdminUCV NGN brinda soporte para SNMP versión 1 y 2c mediante un manager SNMP que complementa las primitivas del protocolo con operaciones avanzadas tales como *Walk* (para recorrer un subárbol de la MIB) y *Table view* (para visualizar los datos de una tabla SNMP ordenados por columnas). Adicionalmente permite importar MIBs, recibir traps y generar gráficas basadas en variables SNMP.

AdminUCV NGN incorpora también un conjunto de utilidades administrativas como ping, tracert, un sniffer y un escáner de puertos TCP, donde cada una de ellas provee soporte para IPv4 e IPv6.

El producto final junto a la documentación de la investigación está publicada como una aplicación con licencia GNU GPL a través de SourceForge.net (un sitio Web que permite el hospedaje y distribución de software libre y sistemas de código abierto) en el siguiente enlace: http://adminucvngn.sourceforge.net.

Palabras Clave: Administración de redes, IPv4, IPv6, SNMP, MIB, GNU.

Tabla de Contenido

Índice de Figuras

Índice de Tablas

Introducción

La masificación de la comunicación, unida a la amplia gama de medios para el intercambio de información que existen en la actualidad, ha creado la necesidad de converger y agrupar las distintas formas de comunicación en una arquitectura común. De esta manera, ha surgido el concepto de Redes de Próxima Generación (NGN por sus siglas en inglés “Next Generation Network”) que propone la integración de múltiples servicios que hoy en día trabajan en distintas plataformas (como por ejemplo, redes de comunicación de datos, redes telefónicas conmutadas, redes celulares inalámbricas, etc.) en una misma red.

Con la finalidad de garantizar dicha convergencia, NGN plantea establecer una red IP basada en paquetes que ofrezca transporte con distintos niveles de calidad de servicio (QoS), lo cual requiere que los diferentes protocolos utilizados soporten los diversos servicios que se pretenden unificar.

Por esta y otras razones, surge el nuevo Protocolo de Internet en su versión 6 (IPv6), el cual en la actualidad se encuentra en constante estudio y evolución, con el fin de hacerlo suficientemente robusto y confiable para ser la base que fundamente esta arquitectura de red de próxima generación.

Una de las principales limitaciones que existen en la actualidad ante esta situación, es la carencia de aplicaciones con soporte para el protocolo IPv6 que permitan avanzar un poco más rápido en el estudio del mismo.

El objetivo del presente Trabajo Especial de Grado es la creación de una aplicación llamada AdminUCV NGN basada en módulos que brinden un conjunto de funcionalidades genéricas para la administración de redes utilizando el protocolo SNMP y con soporte para IPv4 e IPv6.

Con este fin, se ha estructurado el informe en 5 capítulos que explican los diferentes aspectos tomados en cuenta durante la creación de la aplicación. A continuación, se ofrece un breve resumen del contenido de cada uno de estos capítulos:

- **Capítulo 1 (El Problema):** Plantea los diferentes enfoques desde los que puede ser estudiado el problema junto al análisis de la solución; mostrando en detalle los objetivos y el alcance del presente Trabajo Especial de Grado.
- **Capítulo 2 (Marco Teórico):** Presenta las bases teóricas que fueron estudiadas y sobre las que se fundamenta la creación de AdminUCV NGN.

- **Capítulo 3 (Marco Metodológico):** Describe la metodología utilizada y otros aspectos relevantes a tomar en cuenta para la implementación de AdminUCV NGN.
- **Capítulo 4 (Marco Aplicativo):** Explica en detalle las diferentes etapas del proceso de implementación mediante la metodología utilizada.
- **Capítulo 5 (Conclusiones):** Presenta las conclusiones y recomendaciones obtenidas a partir del Trabajo Especial de Grado realizado.

1. El Problema

1.1 Planteamiento del problema

En los últimos años se ha incrementado el uso y la popularidad del nuevo Protocolo de Internet en su versión 6 (IPv6) y con ello, ha nacido la necesidad de contar con sistemas que brinden soporte para dicho protocolo. En el caso particular de la administración de redes, existen pocas aplicaciones que brinden funcionalidades básicas de administración con soporte para IPv6 y aún más escasas son las aplicaciones manager o NMSes de administración vía SNMP que trabajen sobre este protocolo.

Como parte del problema, se agrega el hecho de que las pocas aplicaciones para administración de redes que ofrecen soporte para IPv6, en su mayoría, son aplicaciones propietarias las cuales presentan limitaciones en cuanto a períodos de prueba, acceso al código fuente, precio, entre otros.

La administración de redes comprende una gran cantidad de aspectos para cumplir con sus objetivos y gran parte de las aplicaciones de administración sólo cubren algunos de estos aspectos, creando la necesidad de instalar más de una aplicación para atender las distintas tareas que debe realizar un administrador de red. Además, se debe destacar que muchas de estas aplicaciones no brindan interfaz gráfica de usuario lo cual dificulta el estudio y aprendizaje de las mismas.

1.2 Objetivos

1.2.1 Objetivo General

Desarrollar una aplicación de código abierto para administración de redes con soporte para IPv6, la cual ofrezca funcionalidades básicas de administración, así como un manager SNMP con capacidades para compilar MIBs y monitorización gráfica de la red.

1.2.2 Objetivos Específicos

- Utilizar el lenguaje de programación Java para crear un sistema de administración de redes que ofrezca interfaz gráfica de usuario.
- Conocer, estudiar y utilizar algunos de los paquetes de código abierto compatibles con el lenguaje de programación utilizado, que amplían sus funcionalidades para dar soporte a las distintas herramientas de administración de redes a implementar.
- Complementar las limitaciones de desarrollo en el lenguaje Java con las ventajas del desarrollo en el lenguaje C++.

- Desarrollar como parte de las herramientas de la aplicación, funcionalidades básicas para administración de redes, como: ping, tracert, sniffer y escáner de puertos TCP.
- Proporcionar funcionalidades para la administración de redes vía SNMP.
- Implementar una herramienta que permita la monitorización gráfica de la red vía SNMP.
- Garantizar que todas las funcionalidades provistas por la aplicación ofrezcan soporte para el protocolo IPv6.
- Publicar el resultado final como un producto de código abierto mediante una licencia GNU GPL.

1.3 Justificación

El trabajo a desarrollar surge como una solución ante la necesidad de contar con una aplicación integral para administración de redes que ofrezca soporte para el Protocolo de Internet en sus dos versiones (IPv4 e IPv6), y que proporcione tanto funcionalidades básicas de administración de redes como herramientas orientadas a la administración basadas en SNMP.

Además de las consideraciones a nivel funcional, la aplicación debe contemplar en todo momento los requerimientos de usabilidad e interfaz gráfica que debe proveer una aplicación desarrollada en la actualidad, para asegurar que se le brinda al usuario una aplicación que verdaderamente le ayude y facilite en las diversas tareas para las que fue creada.

Debido a la gran popularidad y aceptación que tiene hoy en día el software de código abierto y gracias a plataformas como SourceForge.net que brindan la capacidad para almacenar y distribuir este tipo de software, el resultado final de la aplicación será distribuido como un software de código abierto, que brinde a la comunidad la capacidad de utilizar, estudiar, analizar y modificar la solución creada, colaborando así con el proceso global de desarrollo de aplicaciones con soporte para IPV6.

Al ofrecer las bondades de toda aplicación de código abierto con licencia GNU GPL, se obtienen otros beneficios como el incremento de la robustez, aceleración en el proceso de desarrollo y mayor flexibilidad en futuras versiones de la aplicación.

1.4 Alcance

La aplicación creada para el Trabajo Especial de Grado tiene el siguiente alcance:

- Las herramientas provistas para la administración de redes mediante SNMP son: manager y generador de tablas SNMP, compilador de MIBs, receptor de traps y herramienta gráfica de monitorización de la red.
- Las herramientas o funcionalidades generales para administración de redes provistas por la aplicación, son: ping, tracert, sniffer y escáner de puertos.
- Todas las herramientas del sistema ofrecen soporte para el protocolo de Internet en sus 2 versiones (IPv4 e IPv6).
- Las funcionalidades de administración vía SNMP presentan soporte para este protocolo en sus versiones 1 y 2c.
- Las operaciones del manager SNMP permiten la consulta de una variable a la vez.
- La consulta de tablas SNMP se realiza indicando el OBJECT IDENTIFIER de la misma. Los resultados obtenidos son mostrados en una tabla donde cada columna está identificada por el OBJECT IDENTIFIER correspondiente.
- Se permite la recepción de traps por cualquier puerto UDP y a nivel de interfaz gráfica se muestran las últimas 5000 alertas recibidas.
- El sniffer captura los primeros 500.000 paquetes recibidos y ofrece soporte para los siguientes protocolos: ARP, IPv4, IPv6, ICMPv4, ICMPv6, TCP, UDP y SNMP.
- El escáner de puertos trabaja sobre el protocolo TCP mediante la especificación de un rango de puertos a escanear.

2. Marco Teórico

En el marco teórico se presentan las bases conceptuales sobre los múltiples aspectos que deben ser estudiados para la creación de una aplicación para la administración de redes tomando en cuenta las nuevas tecnologías y protocolos de redes.

2.1 Administración de Redes Informáticas

Las redes informáticas comprenden una amplia variedad de tecnologías ofrecidas por servicios y productos de múltiples fabricantes. Adicionalmente, el creciente número de usuarios de sistemas informáticos que se apoyan sobre las redes de comunicaciones actuales, así como las exigencias de los mismos, han dado origen al campo de la Administración de Redes para controlar, en términos de capacidad, utilización, configuración, desempeño y rendimiento, las redes de comunicaciones.

2.1.1 ¿Qué es la Administración de Redes?

En relación con la definición de administración de redes, Rodríguez, Sánchez y Mendillo, señalan lo siguiente:

> *"Es un término muy amplio que implica coordinar recursos para planificar, organizar, diseñar, operar, contabilizar, controlar, analizar, evaluar y expandir las redes de comunicaciones con el objetivo de obtener niveles de servicio óptimos, a un costo razonable y con la máxima eficiencia [01]."*
>
> *"Es la planificación, organización, operación, mantenimiento y control de los elementos que forman una red para garantizar un nivel de servicio de acuerdo a un costo [02]."*
>
> *"Conjunto de capacidades que permiten el intercambio y procesamiento de información de gestión a fin de ayudar a la administración de la red a realizar sus actividades con eficiencia [02]."*

A partir de las citas anteriores, se puede definir la administración de redes como el conjunto de actividades que se basan en el acoplamiento de diversas tecnologías para planificar, modelar, diseñar, configurar y desarrollar redes informáticas con la finalidad de obtener un óptimo desempeño a un costo razonable y con la máxima eficiencia; dando así la capacidad de

mantener, corregir, contabilizar, evaluar y expandir los recursos que la componen.

Para llevar a cabo estas actividades de forma eficiente, la administración de redes emplea softwares (Sistemas Informáticos para Administración de Redes) y hardwares (Equipos de red), que permiten evaluar, estudiar y monitorizar el estado de la red en cualquier momento, para brindar así, soluciones a cualquier anomalía que pueda presentarse y un servicio de calidad a los usuarios.

2.1.2 Importancia de la Administración de Redes

La importancia de la administración de redes radica principalmente en la capacidad de unificar coordinadamente distintas áreas (planificación, modelado, diseño, configuración, desarrollo y mantenimiento), de manera que se obtenga una red estructurada que permita la optimización de cada uno de los procesos involucrados en el funcionamiento de la misma.

Asimismo, permite el análisis y planificación para la ubicación, instalación y configuración de cada uno de sus componentes y finalmente, la monitorización de la red para conocer las propiedades y el estado de ésta en un período de tiempo deseado, con la finalidad de prevenir situaciones de error, generar estadísticas relacionadas con su comportamiento o reestructurar su diseño debido a la evolución o expansión de la red.

2.1.3 Sistema para Administración de Redes (NMS)

Un Sistema para Administración de Redes (NMS por sus siglas en inglés “Network Management System”) es un software que centraliza las distintas áreas que abarca la administración de redes en una sola herramienta. Es utilizado por el administrador de la red para controlar y monitorizar el comportamiento del dominio administrado, utilizando servicios para obtener información de la red y/o modificar un valor de alguno de los elementos de la misma, ya sea como resultado de una petición explícita o de manera automatizada.

Dentro de las funciones que puede desempeñar un NMS para lograr los objetivos mencionados anteriormente, se encuentran:

- Capturar y analizar el tráfico que pasa por una red, entender el comportamiento de la misma y de esta forma, poder realizar diagnósticos, localizar averías o posibles problemas. Adicionalmente, presentar la data capturada en un formato legible para que pueda ser interpretado por el administrador de la red.

- Recolectar y guardar información diversa de la red, almacenando, por ejemplo, un registro de sucesos.
- Ejecutar diversas tareas programadas, como un análisis periódico del estado de la red.
- Monitorizar y detectar alarmas en la red, así como correlacionar las distintas alarmas para ayudar a descubrir la causa raíz de la misma.
- Capturar información acerca de los problemas ocurridos en la red, ofrecer soluciones a los mismos y almacenar un reporte de dichas soluciones.
- Realizar operaciones de mantenimiento a la red.
- Proporcionar estadísticas relacionadas con los diversos estados de la red y generar patrones de tráfico, basándose en los datos recolectados.
- Proveer facilidades para ampliar o crear servicios en la red.
- Ofrecer un sistema de tarificación que permita al administrador de la red, identificar y contabilizar qué servicios de comunicaciones están siendo ofrecidos, a quién y en qué momento.

2.1.4 Elementos que Intervienen en la Administración de Redes

En base al enfoque presentado en [03], se realizó el análisis de los elementos presentes en una red que se desea administrar.

La red administrada está compuesta por varios elementos, entre los que se encuentran el conjunto de dispositivos interconectados que intercambian datos y los sistemas diseñados para la administración de la misma. Estos elementos deben comunicarse, y por tanto, la red de comunicación que los conecta se convierte en un elemento vital e importante.

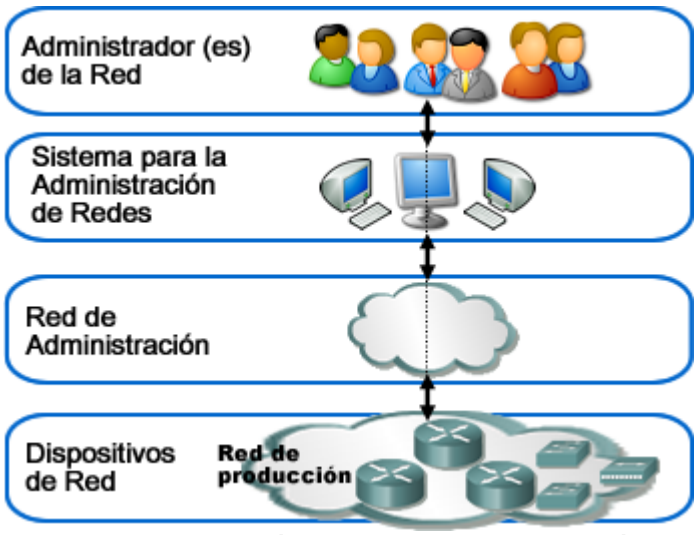


Figura 2.1: Componentes básicos para la administración de redes.

Finalmente, el administrador de la red, constituye el último elemento de esta cadena, ya que éste, es el que configura e inicializa los NMSes para su correcto funcionamiento y posteriormente utiliza la información arrojada por los mismos para el ejercicio de sus funciones. En la Figura 2.1 (tomada de [03]) se muestra un diagrama de los elementos o componentes que conforman una red, los cuales se explican detalladamente a continuación.

- **Dispositivos de Red**

Representan el componente principal de este esquema; son los dispositivos que deben ser administrados, y forman parte del dominio de la red. Éstos, aparte de desempeñar la función para la que fueron creados, deben proveer soporte para comunicarse mediante protocolos de administración, así como también, ofrecer capacidades especiales que les permitan realizar funciones relacionadas a su administración.

- **Agente de Administración:**

Es un software que está presente en todo dispositivo de red que pertenezca al dominio administrado. Su función es la de proveer una interfaz, a través de la cual los NMSes se puedan comunicar con el dispositivo y así, realizar el intercambio de datos con propósitos administrativos.

Los sistemas para la administración de redes (NMS) y el dispositivo de red, comúnmente son referenciados por sus roles, “manager” y “agente”, respectivamente. El intercambio de información entre éstos, es asimétrico, ya que los mensajes se pueden generar debido a peticiones realizadas por el manager hacia el agente o por eventos que se originan dentro del dispositivo y que activan mensajes (traps) desde el agente hacia el manager.

Tal como lo indican en [03], un agente de administración se compone de tres partes conceptuales: La interfaz de administración, una base de información para la administración y la lógica central del agente de administración.

- Interfaz de Administración: este componente de administración es el encargado de la comunicación entre el manager y el dispositivo de red. Para esto, debe soportar el protocolo de administración que define las reglas para el diálogo.

  Existen diversas razones por las que un mismo agente puede proveer varias interfaces de administración, lo cual implica que el dispositivo

podría tener la capacidad de soportar múltiples protocolos para su administración, permitiendo además, responder peticiones provenientes de distintos managers.

- Base de Información para la Administración (MIB, Management Information Base): es un almacén de datos lógico, en donde se guarda información del dispositivo administrado. La información de administración almacenada en la MIB es una abstracción de dicho dispositivo, en la cual se omiten detalles y solamente se exponen datos relativos a la administración.

  Como se mencionó anteriormente, en un mismo dispositivo de red puede haber más de un agente de administración, que puede manejar sus propias MIBs, ya que es posible que éste se enfoque en un aspecto particular de la administración y por lo tanto, tenga que manejar una abstracción distinta del dispositivo subyacente.

- Lógica Central del Agente de Administración: es el motor del agente de administración. Su función principal es la de tomar la data recibida por la interfaz de administración, hacer la traducción de la petición en la MIB y posteriormente, convertir esta información en una operación interna del dispositivo, la cual puede ser leída o asignada (por ejemplo, la solicitud para asignar un valor se puede traducir en una operación de hardware, que escriba cierta data en el dispositivo para modificar su comportamiento).

  De igual manera, al momento de enviar información desde el agente hacia el manager, la lógica central del agente de administración se encarga de tomar valores en el dispositivo y mediante el uso de la MIB crea un mensaje, el cual es traducido al protocolo de administración en la interfaz de administración antes de ser enviada al manager.

- **Sistema para la Administración de Redes**

Tal como se mencionó anteriormente, un NMS es un software que unifica una serie de funcionalidades para administrar una red. En esta subsección, se hace énfasis en el papel que dicho software cumple dentro de una red que se desea administrar.

La principal característica de un NMS es la de brindar una interfaz clara y comprensible a los administradores y traducir las solicitudes de éstos en peticiones que son usadas por los protocolos, para comunicarse con los distintos agentes presentes en el dominio administrado; siendo esta última

característica el rol que cumplen los NMSes dentro del dominio de red. Dicho rol es conocido como manager.

El rol de manager no es exclusivo de los NMSes, un dispositivo que actúe como Proxy de otro(s), puede ser percibido como un manager desde el punto de vista del dispositivo de red final al cual va dirigido la petición. De la misma manera, el rol de manager en algunos casos no está concentrado en un único nodo de la red, ya que el software que cumple esta función, puede ser una aplicación distribuida.

- **Red de Administración**

Los NMSes (manager) y los dispositivos de red (agentes) intercambian datos con propósitos administrativos por medio de una red, la cual es conocida como la red de administración, que se diferencia de la red de producción, en donde transitan los datos de usuario.

Físicamente, las redes de administración y producción pueden ser la misma; lo importante es que el agente sea capaz de diferenciar la data correspondiente a la administración, de otro tipo de tráfico, ya sea mediante el uso de un puerto físico distinto para la administración, el uso de un puerto lógico dentro de un protocolo de capa 4 o alguna otra distinción que facilite la separación del tipo de data recibida.

- **Administrador de la Red**

El elemento final en esta cadena corresponde al administrador de la red, y su función principal es garantizar que la misma funcione adecuadamente.

Tal como lo indican en [03] el administrador tiene diversas tareas, entre las cuales se encuentran:

- Establecimiento de procesos y políticas operacionales: documentación de procedimientos operacionales.
- Recolección de reportes: consiste en configurar los NMSes para que almacenen sucesos en un registro (log) y posteriormente, revisar y analizar dichos registros.
- Documentación de la red: llevar control de la topología y configuración inicial de la red y registrar todos los cambios hechos en la misma, tales como actualización de software y modificaciones en la red.
- Respaldo confiable y políticas de restauración: configurar un respaldo periódico y apropiado para cada tipo de data soportada y establecer políticas para la recuperación de la misma.

- Configuración de parámetros de seguridad: incluye la creación de políticas de seguridad apropiadas y parámetros de acceso dentro de la red, así como verificar periódicamente que las mismas se están cumpliendo.

Adicionalmente, un administrador de red será el encargado de llevar el control de todos los NMSes, para instalarlos y configurarlos, operarlos y así poder utilizarlos, con la finalidad de realizar actividades que van desde el mantenimiento de los sistemas informáticos y líneas de cableado, diagnóstico de fallas, análisis de desempeño y planificación de actualizaciones, hasta tomar acciones preventivas para evitar errores y planificar posibles expansiones de la red o modificaciones de topología.

2.1.5 Áreas Funcionales de la Administración de Redes

La mayoría de las arquitecturas de red utilizan una estructura básica común. Por esta razón, varias organizaciones a nivel mundial han creado estándares que sirven como modelos de referencia, que facilitan el diseño y la implementación de sistemas para la administración de redes.

La primera de estas organizaciones es la ISO (International Organization for Standardization), la cual introdujo un modelo para interconexión de sistemas abiertos (conocido como el modelo OSI, Open System Interconnection) constituido por siete capas, donde cada una representa los niveles que componen una red. Dentro de este modelo se ha hecho una clasificación de las funciones concernientes a la administración de redes, dividiéndola en cinco áreas funcionales conocidas bajo el acrónimo inglés FCAPS (Failure, Configuration, Account, Performance, Security). Cada una de estas áreas se encarga de un conjunto de las actividades establecidas dentro de la administración de redes.

- **Administración de Fallas (F):** La administración de fallas se relaciona con el conjunto de mecanismos que permiten la detección, el aislamiento y la corrección de las operaciones anormales de una red o de un sistema de comunicaciones [05].

- **Administración de Configuración (C):** La administración de configuración abarca las operaciones realizadas para inicializar y modificar las propiedades de configuración de los equipos que componen la red.

  La administración de configuración implica establecer parámetros y valores umbral (threshold), definir filtros, asignar nombres a los objetos administrados, proveer la documentación de los cambios de configuración y cambiar dichas configuraciones cuando sea necesario.

- **Administración de Contabilidad (A):** La administración de contabilidad contempla todas las actividades que permiten la identificación de los recursos utilizados y cuantificación de los costos (cobro de servicios), elaboración de facturas y seguimiento en los pagos, así como todas las funciones que permitan la obtención de ingresos, por medio de los servicios ofrecidos [02].

  La administración de contabilidad no sólo abarca aspectos de tarificación, sino además, incluye el mantenimiento de un historial del uso de los servicios ofrecidos, para generar reportes que reflejen la utilización de dichos servicios, de forma individual o grupal de los distintos usuarios de la red.

- **Administración de Desempeño (P):** La administración de desempeño es necesaria para optimizar la calidad de servicio (QoS). El desempeño de una red se encuentra directamente relacionado con una serie de parámetros de rendimiento que se representan mediante un sistema de métrica específico.

  La administración de desempeño, por tanto, tendrá a su cargo la evaluación del comportamiento de la red, basándose en los valores de los parámetros recolectados [02].

- **Administración de Seguridad (S):** La administración de seguridad se encuentra relacionada con todos los aspectos que puedan afectar la seguridad de la red y su objetivo principal es brindar soporte al sistema de administración de redes en cuanto a las políticas de seguridad. En esta área se deben distinguir tres tópicos fundamentales que generalizan las funciones de la administración: seguridad física, seguridad de acceso y seguridad de datos.

2.1.6 Protocolos de Administración de Redes

Una vez visto los elementos que conforman una red administrada, el estudio se enfoca en los protocolos para la administración de redes. Éstos se crean con el fin de establecer reglas de comunicación para la interacción entre los managers y agentes.

El patrón común de interacción entre managers y agentes, se rige por un modelo asíncrono de petición/respuesta (request/response), que es independiente del protocolo de administración de red utilizado. En la mayoría de los casos, el manager realiza una petición (con el fin de obtener información de administración, cambiar propiedades de configuración o realizar actividades de mantenimiento), y posteriormente el agente envía una

respuesta con la información solicitada. Sin embargo, existe otro tipo de escenario, en el que el agente envía una notificación (alerta) al manager acerca de la ocurrencia de un evento particular.

Existen múltiples protocolos para administración de redes, creados como soporte de los sistemas de administración integrados. Éstos abarcan tres tendencias distintas: administración TMN (Telecommunications Management Network), administración OSI (CMIP-CMISE) y administración SNMP (Simple Network Management Protocol), a continuación se describen los más relevantes.

2.1.7 Administración OSI (CMIP-CMISE)

A pesar que el modelo de referencia OSI es poco empleado, muchos entes gubernamentales y diversas compañías principalmente de telecomunicaciones, han optado por su uso, creando así la necesidad de NMSes basados en OSI. En esta subsección se analizan los componentes básicos presentes en la administración OSI.

Tal como se especifica en [04], a pesar que la administración OSI fue propuesta por la ISO, los estándares resultantes fueron creados en cooperación con la ITU-T. El primer documento significativo que describe la estructura general es el estándar ISO/IEC 7498-4 (OSI Management Framework), publicado en el año 1989, debido a su poca aceptación como punto de partida, el estándar ISO/IEC 10040 (OSI Systems Management Overview) fue creado en el año 1992, y juntos, estos estándares proveen la base para la administración OSI.

La administración de sistemas descrita en el estándar ISO/IEC 10040 distingue entre los aspectos informacionales, organizacionales, funcionales y comunicacionales, los cuales son descritos a continuación.

- **Aspectos Informacionales**

Los aspectos informacionales manejan temas relacionados con los recursos que van a ser administrados. Estos recursos son vistos como objetos administrados (MO, Managed Object), que son una abstracción del recurso que está sujeto a administración. Esta abstracción representa las propiedades del recurso desde un punto de vista relevante para la administración OSI. Tal como se muestra en la Figura 2.2 (tomada de [04]) un MO está formado por los siguientes componentes:

  - Atributos: propiedades o características del objeto y se expresan mediante valores que pueden ser modificados.

- Operaciones: acciones que se pueden ejecutar sobre el MO.
- Notificaciones: mensajes enviados por el MO sin previa solicitud.
- Comportamiento: es lo exhibido por el objeto, producto de relaciones entre las características y la semántica que pueden llevar las mismas.

Figura 2.2: Objeto administrado.

- **Aspectos Organizacionales**

La administración de sistemas OSI está organizada de una manera centralizada, en donde un único manager controla múltiples agentes. El manager puede realizar múltiples operaciones sobre los agentes, los cuales responden con notificaciones a su manager [04].

- **Aspectos Funcionales**

Junto con la creación de los estándares para administración OSI, la ISO inicia el desarrollo de estándares para las áreas funcionales. Luego de un tiempo notaron que la mayoría de los protocolos funcionales usan un conjunto similar de funciones elementales de la administración.

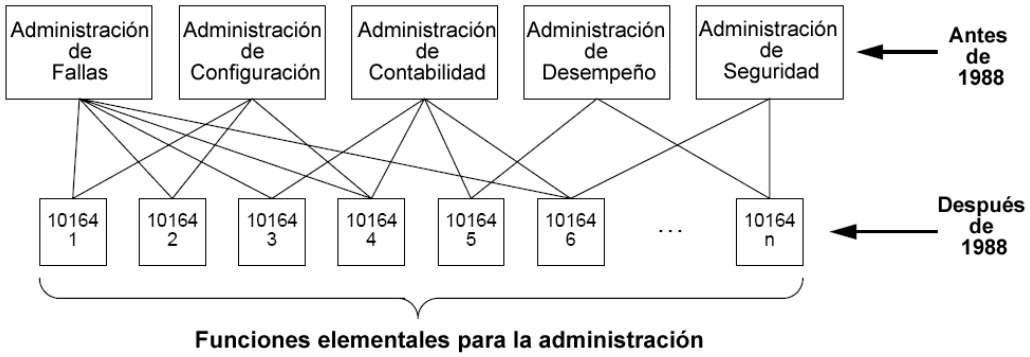


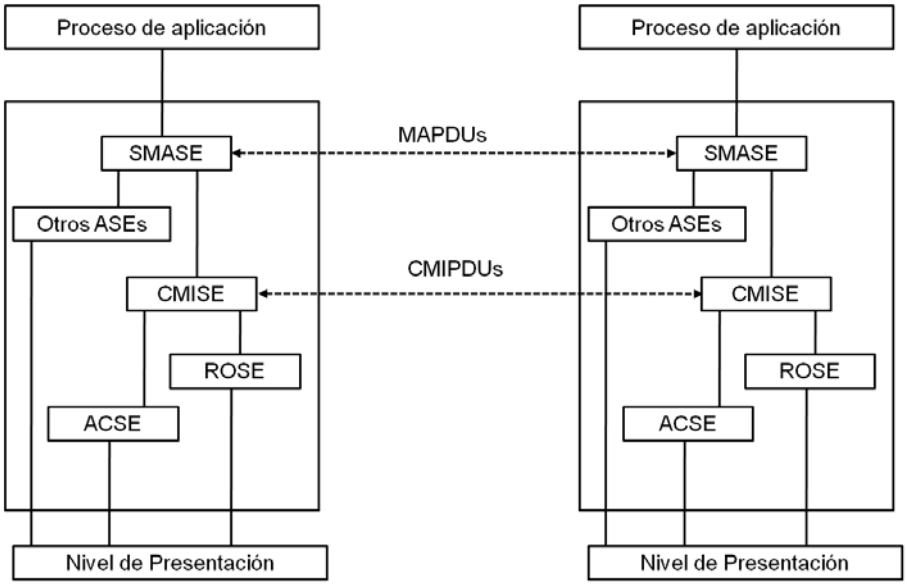
Figura 2.3: Áreas funcionales y funciones elementales para la administración.

Luego de diciembre de 1988 (tal como lo reseñan en [04]), se decide detener la progresión de los protocolos de las áreas funcionales y se concentra en la definición de las funciones elementales para la administración. En la Figura 2.3 (tomada de [04]) se muestra cómo se realizaron algunas divisiones de

áreas funcionales en funciones elementales comunes, las cuales están especificadas dentro del conjunto de estándares de la ISO/IEC 10164.

- **Aspectos Comunicacionales**

A continuación se realiza una breve descripción del marco de trabajo OSI, detallando así los componentes necesarios para que la comunicación entre manager y agente se efectúe. Como se aprecia en la Figura 2.4 (tomada de [07]), los protocolos y componentes para la administración OSI están ubicados en la capa de aplicación y se construyen sobre servicios confiables orientados a conexión [06][07].



MAPDU: management-application protocol data unit.
CMIPDU: common management-information protocol data unit.

Figura 2.4: Elementos principales del marco de trabajo OSI.

Las funciones de la capa de aplicación son usadas por procesos de aplicación, los cuales sirven como interfaz entre el usuario y el protocolo de administración. Estos procesos se comunican con un SMAE (System Management Application Entity), quien a su vez usa un SMASE (System Management Application Service Element) para formar la PDU de datos a intercambiar. La creación de estas PDUs por parte del SMASE se hace mediante la invocación de servicios ofrecidos por los distintos ASE (Application Service Element) [07][08].

En particular se enfoca en el ASE conocido como CMISE (Common Management Information Service Element), el cual tiene como propósito la transferencia de información de administración. Con ese fin, CMISE proporciona servicios básicos para reportar eventos y manipular datos de

administración. Los servicios que CMISE crea pueden ser simétricos o asimétricos y se mapean mediante el protocolo de administración de red CMIP (Common Management Information Protocol) en una operación CMIP remota [08].

Por lo tanto CMIP es un protocolo de administración de red implementado sobre OSI, que provee un modo para que la información de control y de administración pueda ser intercambiada. CMIP convierte las primitivas de servicio en las PDUs correspondientes que utilizan los servicios ACSE y ROSE, los cuales se describen a continuación:

- ACSE (Association Control Service Element): se utiliza para establecer y liberar asociaciones entre entidades de aplicación. El establecimiento lo puede realizar el manager o un agente [07].
- ROSE (Remote Operation Service Element): es el equivalente OSI a una llamada de un procedimiento remoto. ROSE permite la invocación de una operación en un sistema remoto. CMIP usa los servicios orientados a conexión proporcionados por ROSE para todas las peticiones, respuestas y respuestas de error [07].

2.2 SNMP (Simple Network Management Protocol)

SNMP es el protocolo para administración de redes más popular. Está diseñado para trabajar en la capa de aplicación y utiliza los servicios de transporte UDP, por medio de los puertos 161 para intercambio de datos y el puerto 162 para alertas. Este protocolo está compuesto por un conjunto de funciones simples, las cuales proveen capacidades básicas de administración, tales como obtener y modificar el estado de los dispositivos administrados. Se basa en la interacción manager/agente descrita anteriormente y mantiene el modelo asíncrono para la comunicación entre éstos.

Desde sus primeras versiones hasta la actualidad, ha tenido gran aceptación debido a su sencillez. SNMP está pensado para administrar todo tipo de dispositivos que soporten el protocolo. Al igual que CMIP, SNMP trata a los recursos administrados como objetos administrados (MOs). La estructura para definir los MOs y su comportamiento, están descritos en la RFC 1155.

Otras RFCs que complementan el marco de trabajo de SNMP, son la RFC 1157, que trata temas específicos de la definición del protocolo SNMP y la RFC 1213, que define la base para la información de administración de redes TCP/IP (MIB-II).

2.2.1 SMI (Structure Management Information) y MIBs

La Estructura de Información de Administración (SMI), especifica una manera para definir los objetos administrados y su comportamiento. Por su parte, la Base para la Información de Administración (MIB), es la definición (utilizando la sintaxis SMI) de una base de datos conceptual en donde se mantiene información de todos los MOs presentes en el agente. En conjunto, esta lista de MOs define la información que puede ser accedida desde un NMS para obtener el estado general del agente [09].

La SMI utiliza el lenguaje ASN.1 (Abstract Syntax Notation 1) para la definición de los objetos. Dicho lenguaje representa una notación formal que elimina toda posibilidad de ambigüedad entre el significado y la representación del objeto [03].

En este punto, es importante destacar la manera en que SMI define los objetos administrados. Este proceso se inicia con una fase de identificación de los objetos, mediante el uso de OBJECT IDENTIFIER. Un OBJECT IDENTIFIER es una secuencia de enteros que recorren un árbol global. Dicho árbol está compuesto por una raíz (no etiquetada) que a su vez está conectada a un conjunto de nodos etiquetados, los cuales pueden tener hijos. Esta fase puede repetirse hasta llegar a cualquier nivel de profundidad deseado. Cada etiqueta es un par formado por una breve descripción textual del nodo y un entero.

La RFC 1157 describe en detalle los nodos *directory*(1), *mgmt*(2), *experimental*(3) y *private*(4) [10], ya que su función es la de proveer un marco de trabajo para la administración sobre Internet (siendo ésta la base que fundamenta SNMP). La Figura 2.5 muestra de manera gráfica los primeros nodos que componen el árbol global de SMI.

La segunda fase corresponde con la definición de tipos y sintaxis. SMI utiliza un subconjunto de los elementos definidos por ASN.1, dividiéndolo en tres grupos (tipos primitivos, construidos y predefinidos). Dentro de los tipos primitivos se encuentran: *INTEGER*, *OCTET STRING*, *OBJECT IDENTIFIER* y *NULL*. Los tipos construidos son el *SEQUENCE* y *SEQUENCE OF*. Finalmente, se tienen tipos predefinidos que abarcan: *NetworkAddress*, *IpAddress*, *Counter*, *Gauge*, *TimeTicks* y *Opaque* [10]. La fase final se encuentra relacionada con las funciones de codificación; la cual, consiste en instanciar un objeto bajo las normas definidas por BER (Basic Encoding Rules) de ASN.1 [10].

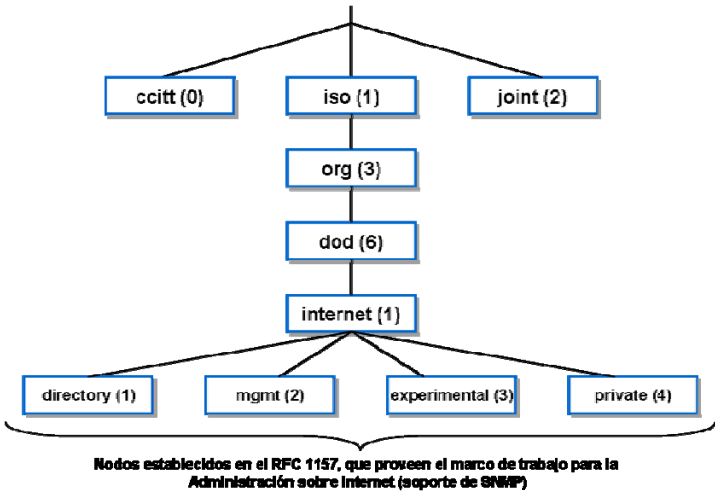


Figura 2.5: Árbol de objetos de SMI.

2.2.2 Operaciones en SNMP

En SNMP, el manager y el agente se comunican a través de mensajes. Un mensaje SNMP consiste en un identificador de versión, un nombre de comunidad y una PDU (Protocol Data Unit). Los tipos de PDU definen un conjunto de cinco primitivas sobre las cuales se basa toda la administración SNMP. Estas primitivas son: GetRequest, GetNextRequest, GetResponse, SetRequest y Trap.

Cada instancia de un tipo de objeto definido en la MIB es identificado unívocamente en SNMP como un "variable name" (variable binding). En general, el nombre de una variable SNMP es un par nombre/valor compuesto por un OBJECT IDENTIFIER (OID) de la forma *x.y*, donde *x* es el nombre de un objeto definido en la MIB e *y* es una porción del OID que identifica la instancia deseada [10].

- **GetRequest:** Esta operación es utilizada por el manager para recuperar uno o más valores del agente. La petición (*GetRequest*) tiene como parámetro una lista de variable bindings que especifican los objetos solicitados. En este caso, el valor del objeto será NULL, ya que al hacer una solicitud, se está interesado en el valor del mismo y por tanto, no se conoce [03].

- **GetNextRequest:** La operación *GetNextRequest* es utilizada para recuperar información de administración del agente. A diferencia de la operación descrita anteriormente (*GetRequest*), se especifica el objeto previo al deseado, por tanto, esta operación retornará el valor del OID que sigue en orden lexicográfico al objeto especificado. Es importante

destacar, que esta operación generalmente es utilizada para consultar valores de tablas.

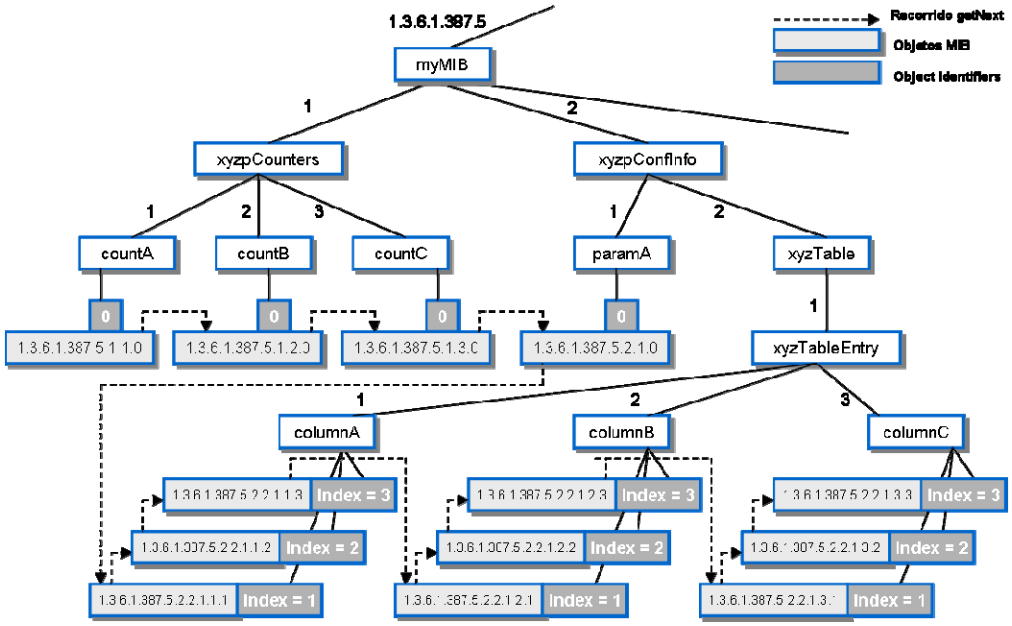


Figura 2.6: Recorrido de objetos con la operación GetNextRequest.

En la Figura 2.6 (tomada de [03]) se muestra un ejemplo en la que se tiene una MIB que abrevia el OID 1.3.6.1.387.5 con la palabra *"myMIB".* Este nodo, tiene dos hijos que se corresponden con *"xyzpCounters*" y "*xyzpConfInfo*". El primero de ellos tiene a su vez, tres nodos hijo que representan objetos de tipo Counter (contadores) y el segundo, tiene como uno de sus nodos hijo, una tabla con información de configuración. Dicha figura, presenta un ejemplo claro de cómo sería el recorrido de objetos y tablas utilizando la operación *GetNextRequest*.

- **SetRequest:** La operación *SetRequest* es utilizada para asignar o modificar un valor a un objeto de la MIB que reside en el agente. Esta operación puede ser empleada de tres formas distintas, que son: para modificar algún valor de configuración de un objeto en el agente; o también para crear o eliminar entidades lógicas en la MIB.

- **GetResponse:** Esta operación es utilizada por el agente para responder a las peticiones realizadas por el manager (*GetRequest*, *GetNextRequest* y *SetRequest*). Utiliza el mismo formato que las operaciones anteriores, con la diferencia de que devuelve el valor del objeto solicitado.

- **Trap:** Los Traps son operaciones utilizadas por el agente de forma asíncrona (sin haberse realizado una solicitud previa), para indicar al

manager que ha ocurrido un evento inesperado. Se han definido siete valores posibles para los diferentes eventos que pueden ocurrir en el agente; los cuales se mencionan en la Tabla 2.1.

Valor	Trap	Descripción
0	coldStart	La entidad ha sido reiniciada, indicando que la configuración del agente pudo haber sido alterada.
1	warmStart	La entidad ha sido reiniciada, pero ni la configuración del agente ni la implementación de la entidad del protocolo ha sido alterada.
2	linkDown	La comunicación con un enlace ha fallado.
3	linkUp	La comunicación con un enlace se ha restablecido.
4	AuthenticationFailure	El agente ha detectado un fallo en la autenticación por parte del manager (comunidad incorrecta).
5	egpNeighborLoss	Un vecino EGP está caído.
6	enterpriseSpecific	Trap no genérico (Específico de una empresa de administración).

Tabla 2.1: Tipos de Traps.

2.2.3 Mensajes SNMP y Estructura del Mensaje

Tal como se mencionó previamente, el manager y el agente se comunican a través de mensajes. Un mensaje SNMP está formado por tres partes [03]:

- Version: número de versión SNMP.
- Community: la comunidad es un string que debe coincidir con el que se ha configurado en el agente, ya que éste funciona como contraseña en el proceso de autenticación de SNMP.
- PDU SNMP (Protocol Data Unit SNMP): la cual consiste en la codificación de la operación SNMP que se desea realizar.

Las operaciones *GetRequest*, *GetNextRequest*, *GetResponse* y *SetRequest* comparten un formato común de PDU, el cual es mostrado en la Figura 2.7. Dicho formato está formado por los siguientes campos [11]:

- PDU Type: indica el tipo de operación que se está realizando. El valor del tipo de PDU varía según se menciona a continuación: 0 para *GetRequest*, 1 para *GetNextRequest*, 2 para *GetResponse* y 3 para *SetRequest*.
- RequestID: es un campo de tipo INTEGER, el cual es asignado aleatoriamente y relaciona la petición del manager con la respuesta del agente.
- Error Status: es un campo de tipo INTEGER, que indica si la operación ha finalizado exitosamente o si ocurrió algún error, teniendo además, cinco condiciones de error predefinidos que se mencionan en la Tabla 2.2.

- Error Index: Identifica el índice del error que haya ocurrido. En caso que no haya errores, este campo se inicializa en 0 en la respuesta (GetResponse).
- Variable Bindings: presenta una lista de variable bindings, las cuales no son más que una lista de pares nombre/valor que representa el(los) objeto(s) que está(n) siendo solicitado(s).

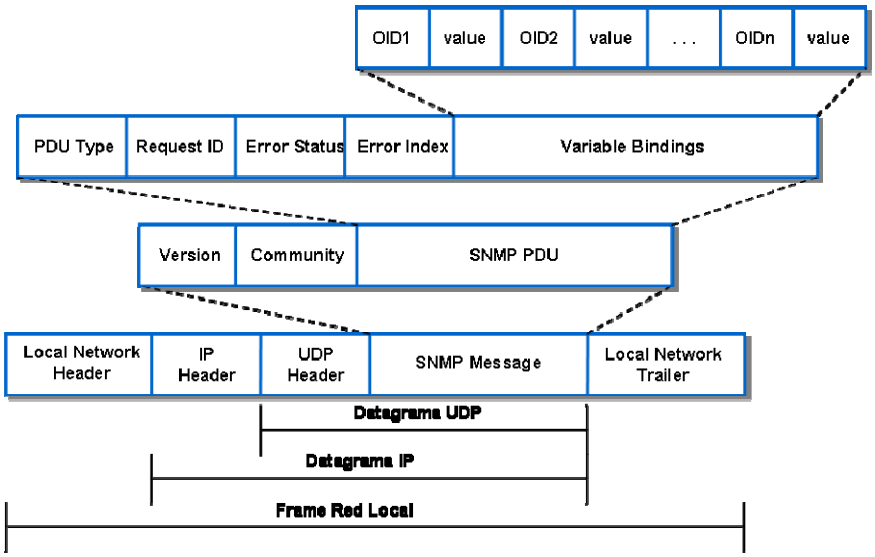


Figura 2.7: PDU para GetRequest, GetNextRequest, GetResponse y SetRequest.

Error Status	Error	Descripción
0	noError	Operación finalizada exitosamente.
1	TooBig	El tamaño de la PDU del GetResponse excede el tamaño predefinido.
2	NoSuchName	El objeto solicitado no hace match con algún nombre de variable en la MIB.
3	badValue	La operación SetRequest, contiene un valor erróneo para el tipo, longitud y valor de la variable especificada.
4	readOnly	El objeto no puede ser alterado.
5	genErr	Otro tipo de error no definido ha ocurrido.

Tabla 2.2: Tipos de errores en la PDU.

Los Traps presentan un formato de PDU distinto al de las operaciones mencionadas anteriormente. Los campos que conforman dicha PDU son:

- PDU Type: indica el tipo de PDU, el cual es 4 para un trap.
- Enterprise: campo de tipo OBJECT IDENTIFIER, que representa la empresa de administración que emitió el trap. Dicho objeto se encuentra definido en el árbol global.
- Agent Address: contiene la dirección IP del agente.

- Generic Trap Type: provee información acerca del tipo de evento generado. Los posibles valores para este campo se encuentran en la Tabla 2.1, donde se indican los tipos de traps.
- Specific Trap Type: provee información del trap o evento ocurrido para fabricantes privados, o para ampliar la información del trap genérico (Generic Trap Type).
- Timestamp: contiene un valor que representa la cantidad de tiempo transcurrido desde la última vez que fue reiniciado el agente hasta la ocurrencia del trap.
- Variable Bindings: presenta una lista de variable bindings, las cuales son una lista de pares nombre/valor que representa el(los) objeto(s) que está(n) siendo solicitado(s).

En la Figura 2.8 se puede apreciar el formato de la PDU del Trap.

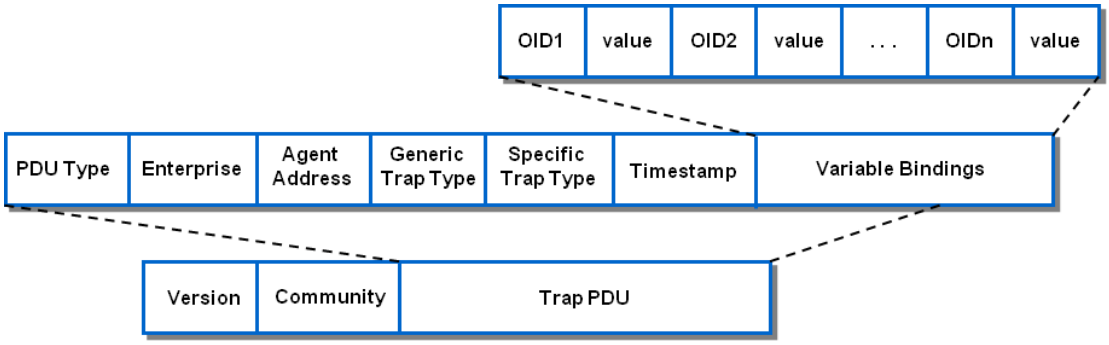


Figura 2.8: PDU Trap.

2.2.4 SNMPv2

La primera versión del protocolo SNMP (ahora conocida como SNMPv1) fue creada con la intención de ser una solución temporal y sencilla mientras se desarrollaba la versión para TCP/IP de CMIP, la cual es conocida como CMOT (CMIP over TCP/IP). Sin embargo, debido a la popularidad que tomó SNMP y la poca aceptación de la administración OSI, en 1992 se decide realizar revisiones y mejoras al protocolo SNMPv1, para ampliar el uso del mismo.

Las debilidades presentes en SNMPv1 que se tomaron en consideración al momento de iniciar los trabajos para mejorar el protocolo son las siguientes:

- Esquema de seguridad vulnerable: la única seguridad que se maneja es el nombre de la comunidad, el cual viaja en texto plano dentro de la PDU. Lo que hace imposible asegurar la autenticidad del mensaje, ya que es fácilmente forjable.

- Administración centralizada: SNMPv1 no contempla la comunicación entre varias estaciones de administración, limitando así la administración a un único nodo centralizado. El nodo centralizado puede sobrecargarse o necesitar que el tráfico de datos administrativo atraviese diversos segmentos de red para alcanzar a todos los dispositivos administrados.
- Obtención de data secuencial: SNMPv1 no provee mecanismos para obtención de datos en bloque, lo que obliga a realizar peticiones repetitivas para la obtención de tablas, generando así una carga extra de tráfico administrativo.
- Manejo limitado de errores: no se especifica la información que se debe incluir en el mensaje trap, ni en qué circunstancias deben ser enviados los mismos. Dejando toda la implementación a decisión de los fabricantes.
- Transporte limitado: SNMPv1, sólo está pensado para usar servicios ofrecidos por UDP para transporte, los cuales son encapsulados en paquetes IP.

Los primeros resultados para crear mejoras al protocolo SNMPv1 fueron publicados en 1992 en un conjunto de 12 documentos (RFC 1411 a la RFC 1452). Esta primera versión que ahora se conoce como Party-based SNMPv2, no fue aceptada debido a la complejidad referente a las mejoras en la seguridad, por lo que se hicieron modificaciones para simplificar el protocolo desde el punto de vista de seguridad y modelo administrativo [11].

Los resultados actuales son conocidos como SNMPv2c (SNMPv2 community-string-based) y están reflejados en las RFCs 3416 y 3418. Éstas usan la estructura de datos creada en la RFC 2578 conocida como SMIv2. Las mejoras incluyen nuevos tipos de datos, nuevos macros, nuevas operaciones que facilitan la transferencia de bloques de datos, códigos de error más específicos y soporte de múltiples protocolos para el transporte.

- **SMIv2 (Structure Management Information version 2)**

Para comenzar el estudio de SNMPv2c (el cual se referencia a lo largo del documento como SNMPv2) se debe analizar la nueva estructura para el manejo de la información. Al árbol de objetos SMI se le agregan dos nuevas ramas dentro del subárbol Internet, añadiendo también otros cambios, tales como la definición de nuevos tipos de datos y la creación de definiciones de módulos.

Al nuevo nodo snmpv2(6), perteneciente al subárbol de *internet*(1). Se le agregan tres nodos hijo: *snmpDomains*(1), *snmpProxys*(2), *snmpModules*(3). Dentro del último de estos tres nodos (*snmpModules*) se tienen a su vez

varios nodos, donde uno de ellos es el subárbol *snmpMIB*(1), sobre el cual se definen varios de los objetos de la MIB utilizada en SNMPv2.

La modificación realizada a la SMI, la cual es conocida como SMIv2, dio pie a modificaciones de la MIB. Tales modificaciones incluyen: introducción de nuevos objetos en el grupo *system*, modificación del grupo *snmp* y la creación de objetos dentro del nuevo nodo *snmpMIB*(1). Éstos proveen nuevas capacidades como las notificaciones, generación de traps bien definidos y la creación de un grupo *set* que le permite a los managers coordinar operaciones [11].

- **Definición de Objetos en SNMPv2**

Los objetos definidos en SNMPv2, son mejoras a los usados en SNMPv1 mediante la creación de un nuevo módulo OBJECT-IDENTITY, la creación de tipos de datos no usados en SNMPv1 y la revisión del módulo OBJECT-TYPE. El módulo OBJECT-TYPE tiene 3 cláusulas que ayuda en la definición de la información acerca de la asignación de un OBJECT IDENTIFIER. Las cláusulas son: STATUS, DESCRIPTION y REFERENCE [11][12].

SNMPv2 mantiene algunos de los tipos de datos usados en SNMPv1 e incluye el soporte para nuevos tipos de datos ASN.1 no usados previamente. Los tipos de datos usados en SNMPv2 son: *Integer32*, INTEGER, OCTET STRING, OBJECT IDENTIFIER, BITS, *IpAddress*, *Counter32*, *Gauge32*, *TimeTicks*, *Opaque*, *Counter64*, *Unsigned32*.

Aparte de los nuevos tipos de datos se crean macros como: MODULE-IDENTITY que es usado para proveer información sobre el contacto y el historial de revisiones de los módulos informacionales. El macro NOTIFICATION-TYPE define la información para la transmisión de data no solicitada. Es utilizada por los traps y los reportes de SNMPv2.

- **Operaciones en SNMPv2**

SNMPv2 mantiene el soporte para las operaciones GetRequest, GetNextRequest, Response, SetRequest y además, introduce nuevas operaciones que son descritas a continuación.

- **GetBulkRequest**: Esta operación permite obtener bloques de datos, dando la capacidad a los managers de conseguir largas secciones de una tabla con una simple consulta. Funciona similar a la aplicación repetitiva de un GetNextRequest por parte del agente antes de retornar la respuesta. Para un manejo más específico se proveen 2 parámetros adicionales: nonrepeaters y max-repetitions. El primero indica el número

de variables a las cuales no se les realizarán consultas sucesivas y el segundo indica el número de sucesores que deben llevar cada una de las variables a las cuales si se le realizarán consultas sucesivas [03].

- **InformRequest:** Esta operación es un mecanismo de información que provee SNMPv2, el cual permite el envío confirmado de información. Por lo tanto, el receptor de una solicitud de información confirmará la recepción de la misma. Este mecanismo puede ser usado para el envío de traps confirmados y es muy útil para la comunicación entre dos managers.

- **SNMPv2-Trap:** En SNMPv1 la sintaxis del PDU para el trap es distinta a las otras PDUs. SNMPv2 soluciona esto utilizando la misma PDU que el resto de los mensajes (exceptuando el GetBulkRequest que usa una PDU distinta) para los traps, dejando así la PDU del trap SNMPv1 como obsoleta [11].

- **Report:** La operación Report fue definida en la versión de borrador de SNMPv2 y aparece en la RFC 3416 como parte de las operaciones soportadas, pero nunca se ha implementado.

- **Formato PDUs en SNMPv2**

En SNMPv2 se mantiene el formato del mensaje similar a SNMPv1 con la diferencia que los campos de la PDU son redefinidos para agregar un nuevo código de errores, permitiendo así reutilizar esta misma PDU tanto para solicitudes, respuestas y traps.

El formato de la PDU genérica usada para los mensajes SNMPv2 es similar a la usada por SNMPv1. Los campos PDU Type, Request ID, Error Status, Error Index y Variable Bindings, tienen una descripción similar a la vista en la sección anterior, con la diferencia que se define una lista de Error Status más amplia a la utilizada en SNMPv1. En la Figura 2.9 se puede apreciar la similitud con la PDU descrita previamente, así como la nueva lista de errores y el soporte para las nuevas PDUs.

- **Ventajas de SNMPv2**

Según lo mencionado en esta subsección, SNMPv2 consta de ventajas como: proveer mecanismos para intercambio de datos entre dos managers, PDUs similares para todos los mensajes soportados, compatibilidad con los PDUs de SNMPv1, operaciones para obtener datos en bloques y manejo de errores más específico. Adicionalmente, se hacen mejoras en los siguientes aspectos: soporte de operaciones atómicas en donde se aceptan resultados parciales y soporte para diversos mecanismos de transporte.

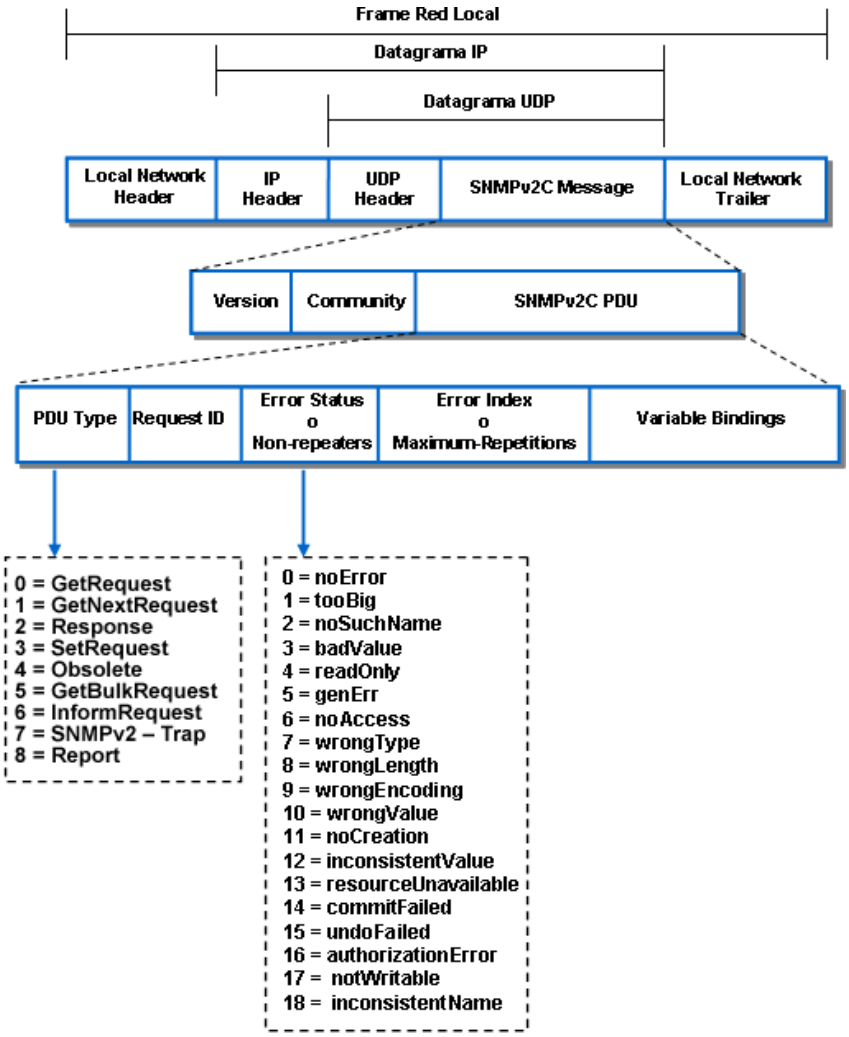


Figura 2.9: PDU para SNMPv2c.

2.3 IPv6 (Internet Protocol version 6)

IPv6, también conocido como Protocolo de Internet de Nueva Generación o IPng, es un nuevo protocolo perteneciente a la capa de red, diseñado para reemplazar al actual Protocolo de Internet (IPv4). Dicho protocolo, surge a partir de grupos de esfuerzo paralelo que buscaban solucionar las limitaciones relacionadas con el espacio de direcciones disponible en IPv4 y adicionalmente, proveer nuevas funcionalidades [13].

2.3.1 Características de IPv6

IPv6 cuenta con un conjunto de características que pueden ser divididas en categorías. Es importante destacar que un subconjunto de éstas constituyen mejoras a características ya existentes en el protocolo IPv4, mientras que

otro subconjunto de ellas representan propiedades completamente nuevas que le añaden funcionalidades a IPv6 en comparación con IPv4.

- **Espacio de direcciones mejorado:** La longitud de la dirección en IPv6 ha sido incrementada de 32 bits a 128 bits, lo cual garantiza el soporte para el continuo crecimiento de usuarios en Internet. Este nuevo formato facilita la estructuración jerárquica del espacio de direcciones, las cuales se asignan en distintos niveles de agregación, simplificando así el enrutamiento global.

- **Simplificación del formato de la cabecera:** La longitud de la cabecera en IPv6 se ha fijado en 40 bytes, de los cuales 16 bytes corresponden a cada una de las direcciones IP (origen y destino). Algunos campos de la cabecera IPv4 han sido eliminados o se han convertido en campos opcionales (mediante el uso de cabeceras de extensión).

- **Soporte mejorado para extensiones y opciones:** En IPv4, las opciones son integradas a la cabecera permitiendo un máximo de 40 bytes para éstas. En IPv6, las opciones son manejadas mediante cabeceras de extensión. Las cabeceras de extensión son opcionales, por lo que sólo son agregadas si son necesarias y no están limitadas en cuanto a cantidad, siempre y cuando no excedan el límite de tamaño de un paquete IPv6. Esto permite que las opciones que sean agregadas en un futuro puedan ser integradas fácilmente.

- **Autoconfiguración:** Esta nueva característica de IPv6 permite que un dispositivo configure automáticamente una dirección para comunicarse dentro del enlace. Adicionalmente, mediante la obtención de uno o más prefijos que anuncian los routers, el dispositivo puede configurar una o más direcciones IPv6 globales.

- **Soporte mejorado para Calidad de Servicio:** Se han añadido nuevas capacidades para permitir etiquetar los paquetes pertenecientes a un flujo particular.

- **Capacidad de autenticación y protección de datos:** En IPv6 la seguridad es provista mediante 2 cabeceras de extensión: AH (Authentication Header) y ESP (Encapsulating Security Payload). La cabecera AH ofrece integridad y autenticación, mientras que ESP proporciona integridad, confidencialidad, autenticación y servicio anti-reenvío.

2.3.2 Formato de la Cabecera IPv6

La longitud de la cabecera IPv6 es de 40 bytes, los cuales se han dividido en un total de 8 campos que son descritos a continuación:

- Version (4 bits): representa la versión del Protocolo de Internet. El valor de este campo siempre debe ser 6.
- Traffic Class (8 bits): este campo reemplaza el campo TOS de IPv4. Es utilizado para datos que requieran un tratamiento especial. Los nodos y routers pueden usar este campo para identificar y distinguir entre diferentes clases o prioridades en los paquetes IPv6 [13].
- Flow Label (20 bits): puede ser usado por la fuente para etiquetar una secuencia de paquetes, que requieren tratamiento especial por parte de los routers IPv6 [14].
- Payload Length (16 bits): es representado por un entero sin signo, el cual indica la longitud en bytes de la carga útil del paquete. Las cabeceras de extensión son consideradas parte de la carga útil y por lo tanto son incluidas en esta longitud.
- Next Header (8 bits): identifica el tipo de cabecera que se encuentra inmediatamente después de la cabecera IPv6 [14].
- Hop Limit (8 bits): este valor es decrementado en 1 por cada nodo que reenvía el paquete. El paquete es descartado si dicho valor es cero. Por lo tanto, en el emisor representa el número máximo de reenvíos para el paquete antes de alcanzar el destino [14].
- Source Address (128 bits): representa la dirección IPv6 del dispositivo que originó el paquete.
- Destination Address (128 bits): representa la dirección IPv6 del destinatario del paquete (posiblemente no del último destinatario si la cabecera de extensión Routing Header está presente) [14].

En la Figura 2.10 se muestra el formato de la cabecera IPv6.

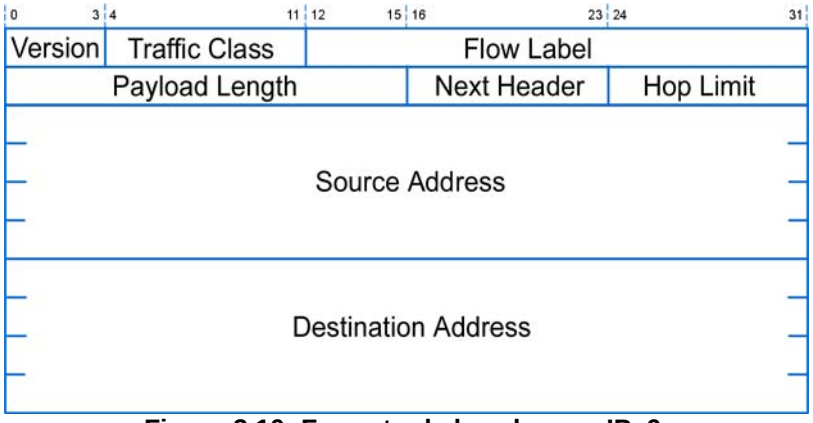


Figura 2.10: Formato de la cabecera IPv6.

2.3.3 Cabeceras de Extensión

Como se mencionó anteriormente para mejorar la extensibilidad de IPv4, donde sólo se podía extender la cabecera de 20 bytes a un máximo de 60 bytes, se definieron las cabeceras de extensión, las cuales aparecen bajo demanda. Las cabeceras de extensión no son analizadas en los nodos intermedios presentes en la ruta (a excepción que aparezca la cabecera de extensión Hop-by-Hop Options).

Las cabeceras de extensión aparecen entre la cabecera IPv6 y la cabecera del protocolo de capa superior. Existe un pequeño número de cabeceras de extensión. El paquete IPv6 puede llevar cero, una o varias cabeceras de extensión y cada una es identificada por el valor del campo Next Header presente en la cabecera previa [14]. En la Figura 2.11 se muestra un ejemplo de cómo se encadenan las cabeceras de extensión, donde siempre la cabecera previa tiene el valor de la próxima en el campo Next Header.

Figura 2.11: Cabeceras de extensión encadenadas.

Si durante el procesamiento de una cabecera de extensión el valor del Next Header no se reconoce se debe descartar el paquete y enviar un mensaje ICMP de error de tipo Parameter Problem, con código 1 (Unrecognized Next Header type encountered). La RFC 2460 hace referencia a 6 tipos de cabeceras de extensión, de las cuales detalla 4. Cuando se usa más de una cabecera de extensión, la RFC 2460 recomienda el siguiente orden: cabecera IPv6, Hop-by-Hop Options header, Destination Options header, Routing header, Fragment header, Authentication header, Encapsulating Security Payload header, Destination Options header, cabecera de capa superior [14].

En la lista previa aparece dos veces la cabecera de extensión Destination Options, la primera vez representa las opciones que serán procesadas por el primer nodo y los subsiguientes nodos presentes en los destinos listados en la cabecera de extensión Routing header. La segunda vez que aparece Destination Options, indica las opciones que solamente se procesarán en el destino final del paquete [14].

A continuación se realiza una breve descripción de las cuatro cabeceras de extensión detalladas en la RFC 2460:

- **Hop-by-Hop Options Header:** Transporta información que debe ser examinada por cada nodo presente en la ruta del paquete. Si esta cabecera está presente debe aparecer inmediatamente después de la cabecera IPv6, la cual debe tener en el campo Next Header un valor de cero [13]. Esta cabecera de extensión contiene diversos campos entre los cuales destacan el campo Options, donde pueden aparecer una o más opciones.

  Cada opción tiene 3 campos, el primero indica qué se debe hacer en caso de que no se reconozca la opción e indica si la información de la opción se puede modificar al enrutar. Entre las opciones más comunes se encuentran, Type Jumbogram, que permite el envío de paquetes con longitudes entre 65.536 y 4.294.967.295 bytes y la opción Router Alert que indica al router que el paquete contiene información importante que debe ser procesada cuando se reenvíe el paquete. Esta última opción principalmente es usada en MLD (Multicast Listener Discovery) y RSVP (Resource Reservation Protocol) [13].

- **Routing Header:** Esta cabecera de extensión es usada para dar una lista de uno o más nodos intermedios que deben ser visitados en la ruta del paquete hacia el destino. Entre los campos de esta cabecera destacan el campo de 1 byte Routing Type, del cual sólo se describe en la RFC 2460 el caso donde vale 0 y el campo de 1 byte Segments Left que indica la cantidad de nodos por visitar [14].

- **Fragment Header:** Para determinar el MTU (unidad máxima de transferencia) el emisor usa un mecanismo conocido como Path MTU Discovery, el cual permite conocer el tamaño máximo permitido para un paquete a través de la ruta. De este modo, en caso que sea necesario los datos son fragmentados en el emisor. La cabecera Fragment Header contiene diversos campos que le permiten al receptor reconstruir los datos fragmentados [13].

- **Destination Options Header:** Esta cabecera transporta información adicional que solamente es examinada por el nodo destino. Como se explicó anteriormente la misma puede aparecer 2 veces. Esta cabecera de extensión tiene un campo Options que es usado de la misma manera que el campo Options de Hop-by-Hop Options header [13].

2.3.4 Direccionamiento en IPv6

La RFC 4291 define la arquitectura para el direccionamiento de la versión 6 del Protocolo de Internet, la cual se analiza en la presente subsección. Se

incluyen en este análisis, los formatos básicos de los diversos tipos de direcciones (unicast, anycast, multicast), así como su estructura [15].

La longitud de la dirección IPv6 es de 128 bits o 16 bytes. Para representarla se dividen los 128 bits en 8 bloques de 16 bits cada uno. Cada bloque de 16 bits se escribe en hexadecimal y se separan por dos puntos (“:”) [13]. Es importante destacar que las direcciones están asignadas a interfaces y no a nodos. Un ejemplo de una dirección IPv6 sería:

2001:0DB8:0000:0000:0000:0000:0E1E:53EF

Para simplificar la representación de las direcciones, se permite eliminar los ceros no significativos dentro de cada bloque. Es decir, los ceros a la izquierda dentro de cada bloque. Por ejemplo, la dirección escrita previamente quedaría así: 2001:DB8:0:0:0:0:E1E:53EF.

Adicionalmente, se permite la compresión, sustituyendo uno o más bloques de ceros seguidos por “::”. Esta compresión de bloques de ceros puede aparecer una sola vez. Por ejemplo, la dirección escrita previamente quedaría así: 2001:DB8::E1E:53EF.

- **Representación de los prefijos**

Un prefijo son los bits más significativos dentro de la dirección IP. Éstos son usados para identificar la subred. La notación del prefijo se obtiene adhiriendo a la dirección IPv6 la longitud del prefijo escrito como un número decimal. Entre la dirección IPv6 y la longitud del prefijo se agrega un slash (/). El formato es el siguiente:

Dirección IPv6/longitud del prefijo

La longitud del prefijo especifica cuantos bits de la dirección son usados como prefijo. Por ejemplo, si se requiere tener un prefijo de 64 bits la notación sería: 2001:DB8::/64.

- **Direcciones Unicast**

Tal como lo indican en [15], son direcciones que son asignadas a una interfaz. Un paquete enviado a una dirección unicast es entregado a la interfaz con esa dirección. Existen diversos tipos de direcciones unicast que se explican más adelante.

- **Interface Identifiers:** Es el conjunto de 64 bits ubicados al final de una dirección unicast IPv6. Éstos son utilizados para identificar una interfaz dentro de un enlace y deben ser únicos dentro de la subred. Para todas las direcciones unicast excepto aquellas que inician con el valor binario

000, el Interface ID debe ser de 64 bits de longitud y debe ser construido con el formato EUI-64 modificado [15].

El formato EUI-64 modificado se obtiene a partir de la dirección MAC de 48 bits, a la cual se le agregan los 16 bits 0xFFFE entre el tercer y el cuarto byte. Luego se invierte el bit "u" (7 bit más significativo) de 0 a 1. Por ejemplo, el Interface ID con el formato EUI-64 modificado de la dirección MAC 00:02:B3:1E:83:29 es 0202:B3FF:FE1E:8329 [13].

- **Unspecified Address:** La dirección IPv6 0:0:0:0:0:0:0:0 ó :: es conocida como Unspecified Address. Esta dirección indica la ausencia de dirección y nunca debe ser asignada a ningún nodo [15].

- **Loopback Address:** La dirección IPv6 0:0:0:0:0:0:0:1 ó ::1 es conocida como Loopback Address. Puede ser usada para que un nodo se envíe un paquete IPv6 a sí mismo. No se le debe asignar a ninguna interfaz física. Nunca se deben enviar paquetes con esta dirección ni deben ser reenviados por un router [16].

- **Global Unicast Address:** Son direcciones Unicast alcanzables y enrutables de manera global. El formato de una dirección Unicast Global es mostrado en la Figura 2.12, donde el *global routing prefix* es el valor asignado a la empresa, *subnet ID* es un identificador asignado a un enlace dentro del la empresa y el *interface ID* es calculado como se definió previamente.

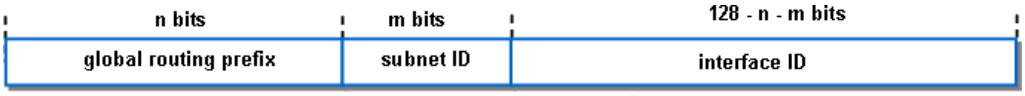


Figura 2.12: Estructura de la dirección Global Unicast.

Como se mencionó anteriormente todas las direcciones Global Unicast que no comiencen con el valor binario 000 deben tener un interface ID con una longitud de 64 bits (n + m = 64). Las direcciones Global Unicast que inicien con el valor binario 000 no tienen limitaciones en tamaño o estructura [15].

- **Direcciones IPv4 embebidas en direcciones IPv6:** Existen 2 tipos de direcciones IPv6 que transportan direcciones IPv4 dentro de sus 32 bits menos significativos. La primera de estas direcciones es conocida como IPv4-Compatible y consiste en los 96 bits más significativos en cero. La segunda de estas direcciones se conoce como IPv4-mapped. Consiste en los 80 bits más significativos en cero y los siguientes 16 bits en uno. En ambos casos, los 32 bits menos significativos de la dirección IPv6 contienen los 32 bits de la dirección IPv4 [15].

- **Link-Local IPv6 Unicast Address:** Esta dirección es usada para la comunicación dentro del mismo enlace y los routers no deben reenviarlas. En la Figura 2.13 se puede apreciar el formato de la misma.

Figura 2.13: Estructura de la dirección Link-Local.

- **Site-Local IPv6 Unicast Address:** Fue diseñada para ser usada dentro de una empresa sin necesidad de un global routing prefix. Tal como lo indica la RFC 3879, estas direcciones están obsoletas [15]. En la Figura 2.14 se ve el formato de la misma.

Figura 2.14: Estructura de la dirección Site-Local.

- **Direcciones Anycast**

Una dirección anycast es asignada a múltiples interfaces (generalmente, múltiples nodos). Cuando un paquete es enviado a una dirección anycast, es entregado sólo a una de las interfaces, siendo escogida usualmente la interfaz más cercana según el protocolo de enrutamiento [15].

Las direcciones anycast están ubicadas dentro del mismo espacio de direcciones unicast y usan los formatos definidos para las direcciones unicast. Cuando una misma dirección unicast es asignada a más de una interfaz se convierte en anycast, creando así múltiples entradas en las tablas de enrutamiento. Cada interfaz participante debe ser configurada explícitamente para tener conocimiento de que la dirección asignada es anycast [13][15].

- **Subnet-Router Anycast Address:** Esta dirección anycast está definida en la RFC 4291 y es requerida. Es creada a partir del prefijo de una interfaz. Tal como se muestra en la Figura 2.15, existe un campo *subnet prefix* que tiene el valor del prefijo que identifica el enlace, el resto de los bits de la dirección son colocados en cero [16].

Figura 2.15: Estructura de la dirección Anycast Subnet-Router.

Todas las interfaces de un router conectadas a una subred tienen asignada la dirección Anycast Subnet-Router para dicha subred.

- **Direcciones Multicast**

Una dirección multicast identifica un conjunto de interfaces IPv6. Una interfaz puede pertenecer a más de un grupo multicast. Un paquete enviado a una dirección multicast es procesado por todos los miembros del grupo multicast [13]. En la Figura 2.16 se puede observar el formato de una dirección multicast IPv6.

Figura 2.16: Estructura de la dirección multicast IPv6.

El byte más significativo identifica la dirección multicast con el valor 0xFF. De los siguientes 4 bits más significativos, el bit O está reservado, el bit R cuando vale 1 indica que un punto *Rendezvous* está embebido en la dirección multicast (un punto *Rendezvous* es un punto de distribución para un flujo multicast específico). Un valor de cero en el bit T indica que la dirección multicast es una dirección bien conocida asignada por la IANA y un valor de uno en el bit T indica que es una dirección temporal. Finalmente, si el bit P vale 1 indica que esta dirección multicast incluye información del prefijo. El campo scope es usado para indicar el alcance de la dirección multicast. Los valores posibles de este campo son mostrados en la Tabla 2.3.

Valor	Alcance	Descripción
0, 3, F	Reserved	No se permite su uso.
1	Interface-local scope	Extiende su alcance a la interfaz de un nodo.
2	Link-local scope	Su alcance es solamente dentro del enlace.
4	Admin-local scope	Es el alcance más corto que debe ser administrativamente configurado.
5	Site-local scope	Alcance dentro de una misma área geográfica o sitio.
8	Organization-local scope	Este tipo intenta que el alcance sea a múltiples sitios que pertenecen a la misma empresa
6, 7, 9, A, B, C, D	Unassigned	Están libres para futura definiciones
E	Global scope	Alcance global.

Tabla 2.3: Valores del campo Scope.

- **Direcciones Multicast bien conocidas:** En la Tabla 2.4 se muestra un resumen de algunas de las direcciones multicast bien conocidas definidas en la RFC 2375.

Dirección	Descripción
FF01:0:0:0:0:0:0:1	Todos los nodos en el interfaz.
FF01:0:0:0:0:0:0:2	Todos los routers en la interfaz.
FF02:0:0:0:0:0:0:1	Todos los nodos en el enlace.
FF02:0:0:0:0:0:0:2	Todos los routers en el enlace.
FF02:0:0:0:0:0:1:2	Todos los agentes DHCP en el enlace.
FF05:0:0:0:0:0:0:2	Todos los routers en el sitio.
FF05:0:0:0:0:0:1:3	Todos los servidores DHCP en el sitio

Tabla 2.4: Resumen de las direcciones multicast bien conocidas.

- **Solicited-Node Multicast Address:** Esta es una dirección a la que todo nodo se debe unir por cada dirección unicast o anycast que tiene asignada. La dirección se forma a partir de los 24 bits menos significativos de la dirección unicast o anycast y se le agregan al prefijo FF02:0:0:0:0:1:FF00::/104 [15].

  Por ejemplo, un host con la MAC 00:02:B3:1E:83:29, puede derivar la dirección link local FE80::202:B3FF:FE1E:8329. La dirección multicast solicited-node correspondiente sería FF02::1:FF1E:8329 [13].

2.3.5 ICMPv6 (Internet Control Message Protocol versión 6)

A diferencia de su antecesor ICMPv4, el cual era utilizado básicamente para envío de información y algunos mensajes de error, ICMPv6 incluye nuevas funcionalidades que lo hacen más completo y robusto

Todos los mensajes ICMPv6 son precedidos por la cabecera IPv6 y cero o más cabeceras de extensión. La cabecera ICMPv6 es identificada por el valor 58 en el campo Next Header de la cabecera que la antecede [17].

- **Formato General de los Mensajes en ICMPv6**

En ICMPv6, todos los mensajes presentan el formato general mostrado en la Figura 2.17.

Figura 2.17: Formato general mensajes ICMPv6.

- Type (8 bits): este campo indica el tipo del mensaje, el cual determina el formato del resto del mensaje [13].
- Code (8 bits): es utilizado dependiendo del tipo de mensaje. Cuando es utilizado, indica un nivel adicional de granularidad (subtipo).

- Checksum (16 bits): permite detectar corrupción en los datos de la cabecera ICMPv6.
- Message Body (tamaño variable): el contenido varía dependiendo del tipo del mensaje. El tamaño total del paquete ICMPv6 no debe exceder al MTU mínimo de IPv6 que es 1280 bytes [13].

- **Clases de Mensajes ICMPv6**

Los mensajes en ICMPv6 son agrupados en 2 clases: mensajes de error y mensajes de información.

- **Mensajes de Error**

Son identificados con el valor cero en el bit más significativo del campo type. Por tanto, su rango de identificación va desde 0 hasta 127. Los mensajes de error definidos en la RFC 4443 son Destination Unreachable (Type 1), Packet Too Big (Type 2), Time Exceeded (Type 3), Parameter Problem (Type 4).

- **Mensaje Destination Unreachable:** Este mensaje es generado cuando un paquete no puede ser entregado. Este mensaje es identificado con el valor 1 en el campo Type. Valor de los campos para el mensaje Destination Unreachable:

  - Type (8 bits): dicho valor es 1.
  - Code (8 bits): especifica la razón por la que no se pudo entregar el mensaje. Los posibles valores de este campo son mostrados en la Tabla 2.5 según la especificación hecha en [17].
  - Unused (32 bits): reservado. Dicho campo es inicializado en cero por el emisor e ignorado por el receptor [17].
  - Data (tamaño variable): información requerida sin exceder el tamaño mínimo del MTU de IPv6.

Code	Descripción
0	No route to destination
1	Communication with destination administratively prohibited
2	Beyond scope of source address
3	Address unreachable
4	Port unreachable
5	Source address failed ingress/egress policy
6	Reject route to destination

Tabla 2.5: Valores del campo Code del mensaje Destination Unreachable.

- **Mensaje Packet Too Big:** Este tipo de mensaje es generado cuando un router no puede enviar un paquete debido a que el tamaño del mismo es más grande que el MTU de salida del enlace.

- **Mensaje Time Exceeded:** Un mensaje de tiempo excedido es generado siempre que un router recibe un paquete con un Hop Limit en 0, o si el router decrementa dicho valor a cero. En este caso, el paquete es descartado y se genera un mensaje de error de tipo Time Exceeded con código 0. Adicionalmente, este tipo de mensaje es utilizado para reportar que se ha excedido el tiempo de reensamblaje de un paquete, en cuyo caso el mensaje de error se genera con código de error 1 [17].

- **Mensaje Parameter Problem:** Cuando existe algún problema con un campo de la cabecera IPv6 o las cabeceras de extensión que impide el procesamiento del paquete; se descarta el paquete y se genera un mensaje de tipo 4, indicando problemas con parámetros:

  - Type: posee el valor 4.
  - Code: la Tabla 2.6 muestra los posibles valores para este campo.
  - Pointer: indica el byte donde fue encontrado el error en el paquete.

Code	Descripción
0	Erroneous header field encountered
1	Unrecognized Next Header type encountered
2	Unrecognized IPv6 option encountered

Tabla 2.6: Posibles valores campo Code del mensaje Parameter Problem.

- **Mensajes de Información**

Para los mensajes de información, el bit más significativo es colocado en 1. Por tanto, su valor varía entre 128 y 255. En la Tabla 2.7 se observa un resumen de los valores posibles para el campo type que corresponden con mensajes de información.

Type	Descripción
128	Echo Request
129	Echo Reply
130	Multicast Listener Query
131	Multicast Listener Report
132	Multicast Listener Done
133	Router Solicitation
134	Router Advertisement
135	Neighbor Solicitation
136	Neighbor Advertisement
137	Redirect Message

Tabla 2.7: Resumen de los mensajes de información ICMPv6.

La RFC 4443 describe los mensajes Echo Request y Echo Reply, los cuales son utilizados para determinar conectividad entre un par de hosts. Estos mensajes de información son descritos a continuación.

- **Echo Request:** En la Figura 2.18 se puede observar el formato de este mensaje. El tipo del mensaje es 128 y el código no es utilizado por lo que es colocado en 0. Los campos Identifier y Sequence Number, son utilizados para la correspondencia entre las peticiones y respuestas realizadas.

- **Echo Reply:** Se envía como respuesta a un mensaje de Echo Request. El tipo del mensaje es identificado por el valor decimal 129 y el campo código no es utilizado. Los campos Identifier y Sequence Number, poseen el mismo valor que el mensaje Echo Request recibido para indicar a qué mensaje (petición) está respondiendo.

Figura 2.18: Formato de mensajes Echo Request y Echo Reply.

2.4 Software Libre vs Código Abierto

Los términos Software Libre (“Free Software”) y Código Abierto (“Open Source”) tienden a generar confusión o malas interpretaciones entre la mayoría de las personas.

La Free Software Foundation define Software Libre, como todo aquel programa que ha sido creado para que cualquier usuario que lo adquiera pueda ejecutar, copiar, distribuir, estudiar, modificar y mejorar dicho software [18]. Es importante destacar que “libre” no implica gratis, y que todo software libre garantiza a los usuarios 4 libertades fundamentales:

- Usar el programa con cualquier propósito.
- Estudiar cómo funciona el software y adaptarlo a sus necesidades.
- Distribuir copias para ayudar a otros.
- Mejorar el programa y hacer públicas dichas mejoras.

El término Open Source surge a partir de 1998 como un nuevo movimiento de software conocido como la Open Source Initiative, la cual define Código Abierto como todo programa que es desarrollado y distribuido libremente bajo la filosofía de que al compartir el código, el programa resultante tiende a ser de mayor calidad, lo cual se contrapone a la visión filosófica de Software Libre que plantea como poco ético el no compartir el software, ya que va en contra de las leyes naturales.

La Open Source Initiative propone 10 principios que debe cumplir un código para poder llamarse "Open Source", las cuales son equivalentes a las 4 libertades o principios definidos por el Software Libre [19]; éstas son:

- Redistribución libre del programa.
- Se debe incluir el código fuente.
- Se debe permitir la publicación de modificaciones del software.
- Integridad del código fuente del autor, por lo que las licencias pueden requerir que las modificaciones sean redistribuidas sólo como parches.
- No se puede discriminar a personas o grupos, nadie puede dejarse fuera.
- Sin discriminación de áreas de iniciativa, por lo que los usuarios comerciales no pueden ser excluidos.
- Distribución de la licencia; deben aplicarse los mismos derechos a todo el que reciba el programa.
- La licencia no debe ser específica de un producto.
- La licencia no debe restringir otro software, no puede obligar a que algún otro software que sea distribuido por el mismo medio sea también de código abierto.
- La licencia debe ser tecnológicamente neutral.

2.4.1 Copyleft

Copyleft es el tipo de protección jurídica que otorgan un grupo de licencias para garantizar el derecho de cualquier usuario a utilizar, modificar y redistribuir un programa o sus derivados, siempre que se mantengan las mismas condiciones de utilización y difusión [20].

El copyleft protege la redistribución del software garantizando que cualquiera que distribuya un software con o sin modificaciones otorgue las mismas libertades que le fueron concedidas al obtener el programa.

2.4.2 Licencia Pública GNU (GPL)

GPL (General Public License) es un tipo de licencia copyleft, que establece un conjunto específico de términos de distribución que protegen un software para garantizar la libertad de compartirlo y modificarlo, asegurando que sea software libre para todos sus usuarios. Es importante destacar, que la licencia GNU GPL es aceptada tanto por el movimiento Free Software como por el movimiento Open Source.

3. Marco Metodológico

Para el logro exitoso de los objetivos planteados en el capítulo 1, es necesario definir un esquema o metodología de trabajo que permita el desarrollo rápido y eficiente de cada uno de los requerimientos de la aplicación. A continuación, se presenta la especificación de la metodología utilizada y otros detalles importantes que fueron tomados en cuenta para el desarrollo e implementación de la aplicación.

3.1 Adaptación de la Metodología de Desarrollo

La mayoría de las metodologías de desarrollo que existen plantean un esquema de trabajo que se divide en 4 fases: Análisis, Diseño, Codificación y Pruebas. Un conjunto de los modelos tradicionales proponen que dicho esquema se ejecute de forma lineal o secuencial para la implementación de la aplicación.

En la realidad, la adaptación de estos esquemas secuenciales a la práctica es poco frecuente, ya que durante el desarrollo de una aplicación, tienden a aparecer nuevos requerimientos durante la ejecución de cualquiera de las fases, ya sea porque el cliente tiene una nueva necesidad o porque al obtener resultados de la aplicación se decida agregar o incluir nuevos criterios, lo cual conlleva a la replanificación de las actividades, ejecutando nuevamente las 4 fases del método de desarrollo.

Por esta razón, se ha decidido trabajar con un método iterativo basado en el modelo de desarrollo ágil, el cual propone dividir el desarrollo de la aplicación por iteraciones y en cada iteración realizar las 4 fases del marco de trabajo tradicional. Es importante destacar, que una iteración (bajo dicha metodología) puede ser vista de 2 formas: (1) como un período de tiempo (que varía entre 1 y 4 semanas) para el desarrollo de un grupo de requerimientos definidos durante la fase de Análisis de dicha iteración, o (2) como la implementación de un módulo de la aplicación [21].

La adaptación del esquema de trabajo escogido se basa en el modelo de desarrollo ágil, dividiendo los requerimientos de la aplicación por módulos, y desarrollando un módulo por iteración. Por tanto, en cada iteración se realizan las 4 fases del método tradicional. A continuación se describen los aspectos que se tomaron en cuenta en cada una de las fases de desarrollo.

3.1.1 Análisis y Planificación

Durante la fase de análisis se definen los requerimientos para cada iteración (módulo). Es importante destacar, que por ser un proceso iterativo toda fase de análisis (excepto la primera iteración) es antecedida por una fase de pruebas del módulo que se estaba desarrollando.

Dichas pruebas pueden implicar el desarrollo de nuevos requerimientos al no obtener el resultado o comportamiento esperado, los cuales son analizados en una nueva iteración. En caso de que la fase de pruebas arroje los resultados esperados, se procede a la ejecución de una nueva iteración que incluirá el análisis de requerimientos del nuevo módulo a implementar.

Esta fase es documentada mediante la creación y especificación de diagramas de casos de uso, los cuales muestran por nivel de abstracción todas las funcionalidades (requerimientos) que debe incluir cada módulo.

3.1.2 Diseño

Durante la fase de diseño de cada iteración se crea la estructura lógica del módulo a desarrollar. Para ello, se definen las clases que van a interactuar, así como los métodos que deben ser incluidos dentro de cada una de las clases.

Esta fase debe ser documentada mediante el uso de diagramas de clases, los cuales pueden mostrar refinaciones de clases creadas en una iteración anterior o el diseño de nuevas clases del módulo a desarrollar.

3.1.3 Codificación

Una vez analizados los requerimientos y diseñada la solución para cubrir dichos requerimientos, se procede a codificar la misma. Es decir, se implementan las clases diseñadas y se desarrollan cada uno de los métodos definidos para las clases. Durante esta fase se documentan los aspectos de codificación más importantes por módulo.

3.1.4 Pruebas

Toda iteración culmina con la verificación de la solución creada para asegurar que se cumplan con los requerimientos planteados al inicio y determinar si a partir del desarrollo de la solución se desprenden nuevos requerimientos.

Al finalizar cada iteración se realizan pruebas de funcionamiento, para validar que todos los datos arrojados por el módulo sean correctos, es decir, se verifica que cada módulo tenga el comportamiento esperado.

Luego, durante las pruebas de integración se analiza si la interfaz gráfica de usuario cubre todos los requerimientos necesarios para la recolección de datos y muestra de resultados y se llevan a cabo pruebas generales que garanticen que la aplicación funcione en conjunto y que cada uno de los módulos probados de forma independiente continúen funcionando de manera adecuada luego de la integración.

Los resultados de cada una de las pruebas descritas y los posibles nuevos requerimientos alimentan a la fase de análisis de la nueva iteración, la cual toma esta información unida a los requerimientos de otros módulos para determinar los requerimientos a cubrir en la próxima iteración.

3.2 Tecnologías a Utilizar

Tal y como se planteó en el capítulo 1, para el desarrollo de la aplicación se utilizará Java como plataforma de desarrollo y un conjunto de paquetes de código abierto compatibles con dicho lenguaje de programación que amplían las funcionalidades del mismo. Los paquetes o librerías a utilizar son: SNMP4j, Mibble, JFreeChart y Jpcap. Es importante destacar, que debido a que existen algunas tareas que resultan de muy bajo nivel para Java, también fue utilizado C++ como lenguaje complementario para la resolución de este tipo de problemas.

A continuación se describen brevemente cada una de las tecnologías mencionadas:

- Java: tecnología desarrollada por Sun Microsystems que ofrece la plataforma necesaria para la creación de interfaces gráficas (mediante el paquete Swing) y todo el entorno de la aplicación. Para la creación del sistema será utilizada la versión 1.6.0_03 del JRE (Java Runtime Environment).
- SNMP4j: diseñado para ser utilizado bajo entorno Java, que provee las funcionalidades básicas para las operaciones de SNMP.
- Mibble: permite la lectura y compilación de MIBs.
- JFreeChart: librería Java que facilita la creación de gráficos dentro de las aplicaciones.
- Jpcap: paquete Java que permite la captura y envío de paquetes por la red, ofreciendo soporte para los protocolos Ethernet, ARP, IPv4, IPv6, ICMPv4, TCP y UDP.

3.3 Prototipo General de Interfaz

Con el fin de obtener lineamientos estándar de interfaz gráfica entre los diferentes módulos de la aplicación, se diseñó un prototipo de interfaz general que establece dichos lineamientos y que se seguirán a lo largo del desarrollo de la aplicación para procurar la usabilidad y mantener la consistencia.

En la Figura 3.1 se muestra el prototipo de interfaz definido para la ventana principal, la cual será un frame con medidas 690 píxeles de alto por 900 píxeles de ancho. En la barra de título, se mostrará el logo de la aplicación AdminUCV NGN.

Se ha decidido utilizar una estructura en pestañas para el acceso a las distintas herramientas de la aplicación, con el objetivo de que el usuario pueda navegar fácilmente entre las diferentes funcionalidades de AdminUCV NGN y ofrecer mayor flexibilidad cuando varios módulos sean ejecutados simultáneamente. Adicionalmente, la Figura 3.1 muestra el prototipo de interfaz para el “Menú” (ubicado en la primera pestaña) de la aplicación, estableciendo que las diferentes opciones o herramientas sean mostradas en dos paneles con botones, donde cada botón permite el acceso a una herramienta o módulo, siendo ésta además, una segunda opción para el acceso a las funcionalidades de la aplicación.

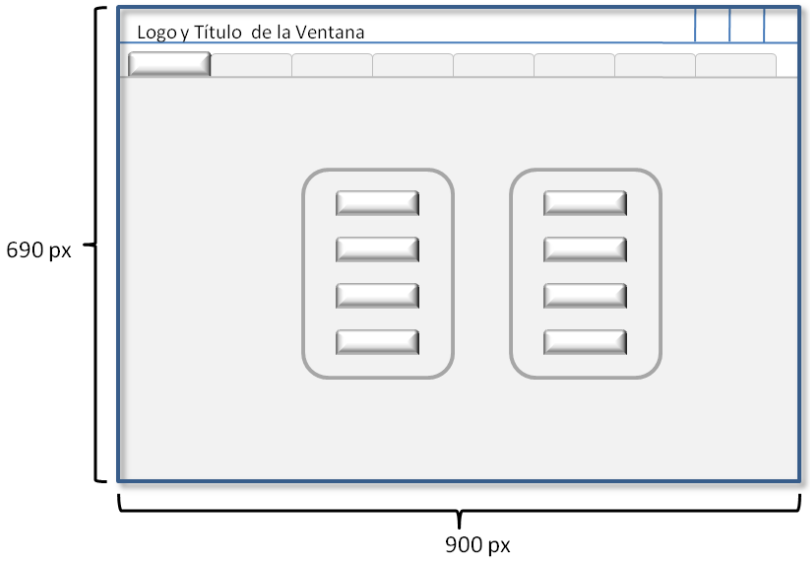


Figura 3.1: Prototipo de interfaz ventana principal.

Posteriormente, se diseñó el prototipo de interfaz para el panel general que se usará en cada una de las funcionalidades de la aplicación. Dicho prototipo es mostrado en la Figura 3.2, que propone que las medidas en píxeles del

panel sean 600 de alto por 885 de ancho. Adicionalmente, cuando la interfaz de un módulo amerite el uso de una barra de herramientas, la misma se definirá en una nueva sección con el mismo ancho que la anterior y con una altura de 40 píxeles.

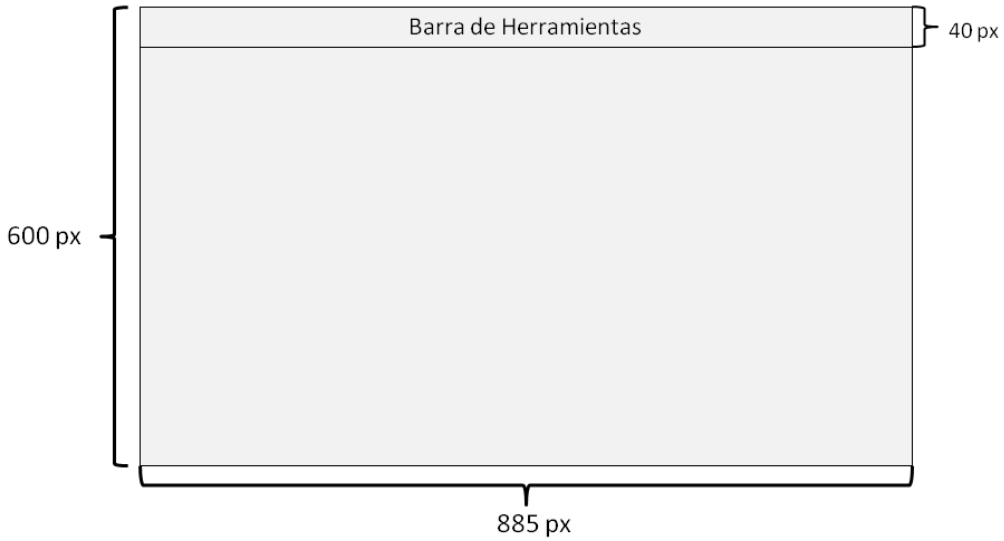


Figura 3.2: Prototipo de interfaz panel general.

Dentro de cada barra de herramientas, se colocarán las diferentes opciones que proporcione el módulo que la implemente, representadas a través del uso de botones. Cada botón tendrá 75 píxeles de ancho por 35 píxeles de alto, dando una separación de 5 píxeles entre dos opciones (botones) de la barra de herramientas. La estructura descrita anteriormente es mostrada en la Figura 3.3.

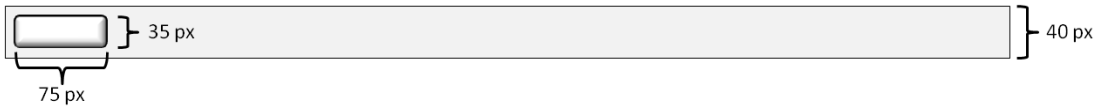


Figura 3.3: Prototipo de interfaz barra de herramientas.

4. Marco Aplicativo

En el capítulo 3 se describió la metodología a utilizar para la implementación de la aplicación, la cual consiste en una metodología iterativa adaptada al modelo de desarrollo ágil, donde una iteración refleja el proceso de creación de un módulo de la aplicación.

Con el fin de simplificar y facilitar la comprensión de los resultados obtenidos durante cada iteración, los mismos son documentados agrupándolos por fases (Análisis, Diseño, Codificación y Pruebas) y especificando en cada una de ellas el detalle de todas las iteraciones. A continuación, se describe el proceso práctico que se siguió a lo largo del desarrollo de la aplicación.

4.1 Fase de Análisis

Antes de iniciar el desarrollo de la aplicación, se lleva a cabo la fase de análisis donde se determina de manera global y luego, por cada uno de los módulos en cada iteración, los principales requerimientos de la aplicación, los cuales deben cubrir los objetivos definidos en el capítulo 1 para dar solución al problema. A partir de este punto, se crea el diagrama de casos de uso, que refleja las principales funcionalidades de la aplicación y cómo debe interactuar la misma con el usuario para el logro de cada uno de los objetivos.

A continuación, se muestra cómo se estructuró la lista de requerimientos de la aplicación:

- Definir una interfaz gráfica de usuario basada en el prototipo general de interfaz planteado en el Marco Metodológico que se fundamenta en principios de usabilidad.
- Contar con un manager SNMP.
- Dar soporte SNMP para las versiones 1 y 2c del protocolo.
- Permitir compilar MIBs.
- Mostrar resultados de consultas SNMP avanzadas, como la construcción de tablas y monitorización de elementos.
- Brindar un módulo para la recepción de traps.
- Incorporar utilidades generales para administración de redes tales como ping, tracert, escáner de puertos y sniffer.
- Dar soporte para el protocolo IPv4 e IPv6 en cada uno de los módulos definidos en la aplicación.

Luego de definir la lista de requerimientos, se genera la documentación apropiada que incluye la creación de los diagramas de casos de uso y su

especificación. Éstos son mostrados por niveles de acuerdo al grado de abstracción aplicado para cada funcionalidad del sistema.
En la Figura 4.1 se puede observar el nivel 0 del diagrama de casos de uso, el cual refleja el sistema a crear y adicionalmente muestra la interacción con un actor que es llamado “Usuario”. Un Usuario será todo aquel que interactúe con el sistema, es decir que utilice la aplicación, el cual puede ser desde un usuario inexperto hasta un administrador de redes.

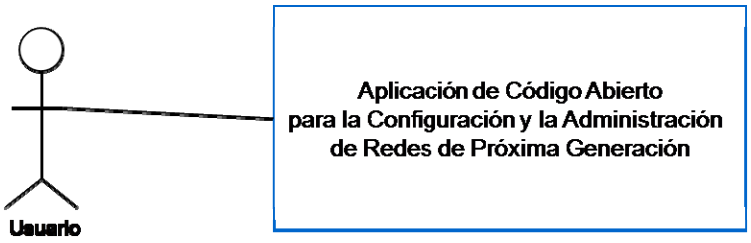


Figura 4.1: Diagrama de casos de uso nivel 0.

4.1.1 Iteración 1: General

En un siguiente nivel de abstracción o nivel 1, se definen las principales funcionalidades que debe cumplir la aplicación, obteniendo 11 casos de uso o módulos principales.

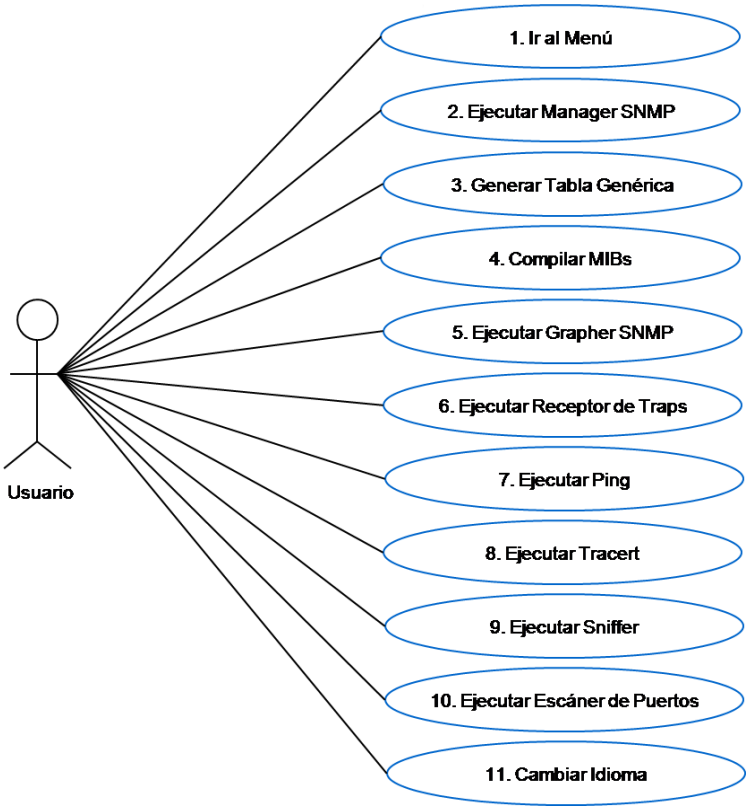


Figura 4.2: Diagrama de casos de uso nivel 1.

En la Figura 4.2 puede observarse el diagrama perteneciente al nivel 1 de abstracción. A continuación se presenta la especificación de cada caso de uso descrito en el nivel 1 (dicha especificación puede verse en el orden en que se definieron los casos de uso en las tablas: Tabla 4.1, Tabla 4.2, Tabla 4.3, Tabla 4.4, Tabla 4.5, Tabla 4.6, Tabla 4.7, Tabla 4.8, Tabla 4.9, Tabla 4.10 y Tabla 4.11).

Caso de uso	1. Ir al Menú
Actores	Usuario
Descripción	Presenta el menú de la aplicación, el cual muestra todas las opciones para acceder a los diferentes módulos.
Flujo básico	• Acceder a la aplicación • Escoger alguna de las opciones del menú

Tabla 4.1: Especificación caso de uso (1) Ir al Menú.

Caso de uso	2. Ejecutar Manager SNMP
Actores	Usuario
Descripción	Permite acceder al Manager para realizar las diferentes consultas SNMP.
Flujo básico	• Acceder al Manager. • Especificar los parámetros de la consulta. • Seleccionar la operación SNMP a realizar.
Puntos de exclusión	• Completar OID con .0. • Ejecutar Operación SNMP. • Generar Tabla Genérica. • Compilar MIBs.

Tabla 4.2: Especificación caso de uso (2) Ejecutar Manager SNMP.

Caso de uso	3. Generar Tabla Genérica
Actores	Usuario
Descripción	Permite realizar consultas avanzadas SNMP y mostrar los datos en una tabla ordenados por columnas.
Flujo básico	• Acceder al módulo Tabla Genérica. • Configurar los parámetros. • Mostrar los resultados en una tabla.
Puntos de exclusión	• Configurar Parámetros de Consulta.

Tabla 4.3: Especificación caso de uso (3) Generar Tabla Genérica.

Caso de uso	4. Compilar MIBs
Actores	Usuario
Descripción	Permite cargar, eliminar y recargar MIBs.
Flujo básico	• Acceder al módulo y realizar las diferentes operaciones permitidas dentro del mismo.
Puntos de exclusión	• Cargar MIB. • Eliminar MIB. • Recargar MIB.

Tabla 4.4: Especificación caso de uso (4) Compilar MIBs.

Caso de uso	5. Ejecutar Grapher SNMP
Actores	Usuario
Descripción	Permite monitorizar de manera gráfica variables vía SNMP.
Flujo básico	• Acceder al Grapher SNMP. • Configurar parámetros. • Seleccionar elementos a graficar (interfaces o variables). • Establecer intervalos de consulta • Comenzar la monitorización.
Puntos de exclusión	• Configurar Parámetros. • Abrir Configuración. • Guardar Configuración. • Cambiar intervalo de consulta. • Comenzar a graficar.

Tabla 4.5: Especificación caso de uso (5) Ejecutar Grapher SNMP.

Caso de uso	6. Ejecutar Receptor de Traps
Actores	Usuario
Descripción	Permite la recepción de traps en cualquier puerto UDP disponible en la máquina que se ejecuta.
Flujo básico	• Acceder al módulo Receptor de Traps. • Iniciar la recepción de traps.
Puntos de exclusión	• Configurar Filtros

Tabla 4.6: Especificación caso de uso (6) Ejecutar Receptor de Traps.

Caso de uso	7. Ejecutar Ping
Actores	Usuario
Descripción	Implementación gráfica de la herramienta Ping
Flujo básico	• Acceder al Ping. • Escoger la versión del protocolo IP. • Iniciar el envío de peticiones eco ICMP.
Puntos de inclusión	• Mostrar Estadísticas de Ping
Puntos de exclusión	• Especificar Opciones de Ping

Tabla 4.7: Especificación caso de uso (7) Ejecutar Ping.

Caso de uso	8. Ejecutar Tracert
Actores	Usuario
Descripción	Permite conocer la ruta de saltos (nodos) por los cuales transita un paquete antes de llegar a su destino.
Flujo básico	• Acceder al Tracert. • Escoger la versión del protocolo IP. • Iniciar el envío para determinar la ruta al host destino.
Puntos de exclusión	• Seleccionar Opciones Tracert

Tabla 4.8: Especificación caso de uso (8) Ejecutar Tracert.

Caso de uso	9. Ejecutar Sniffer
Actores	Usuario
Descripción	Permite capturar el tráfico de la red para posterior análisis de los paquetes.
Flujo básico	• Acceder al Sniffer. • Configurar los parámetros de captura.

	• Iniciar la captura de paquetes.
Puntos de exclusión	• Configurar Parámetros de Captura. • Abrir Captura. • Guardar Captura. • Mostrar Estadísticas de Captura.

Tabla 4.9: Especificación caso de uso (9) Ejecutar Sniffer.

Caso de uso	10. Ejecutar Escáner de Puertos
Actores	Usuario
Descripción	Permite analizar los puertos TCP activos en un host o rango de hosts.
Flujo básico	• Acceder al módulo Escáner de Puertos. • Definir el rango de puertos TCP a escanear. • Iniciar el escaneo del rango de puertos seleccionado.
Puntos de inclusión	• Definir Rango de Puertos a Escanear

Tabla 4.10: Especificación caso de uso (10) Ejecutar Escáner de Puertos.

Caso de uso	11. Cambiar Idioma
Actores	Usuario
Descripción	Permite cambiar el idioma de la aplicación.
Flujo básico	• Escoger el idioma de preferencia.
Puntos de exclusión	• Actualizar archivo de configuración.

Tabla 4.11: Especificación caso de uso (11) Cambiar Idioma.

En las iteraciones siguientes se muestra el análisis realizado en un segundo nivel de abstracción, correspondiente con cada módulo de la aplicación.

4.1.2 Iteración 2: Manager SNMP

En la Figura 4.3 se puede observar el detalle del caso de uso Manager SNMP en un segundo nivel de abstracción. Posteriormente, se muestra la especificación correspondiente en la Tabla 4.12 y Tabla 4.13.

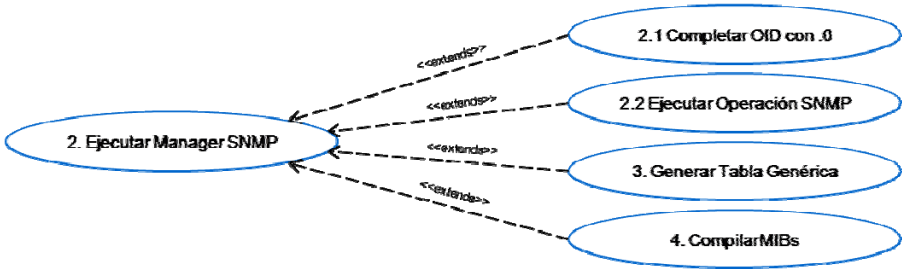


Figura 4.3: Diagrama de casos de uso nivel 2, Manager SNMP.

Caso de uso	2.1 Completar OID con .0
Actores	Usuario
Descripción	Agrega la extensión .0 a un OID que fue previamente seleccionado del árbol de la MIB.
Flujo básico	• Completa el OID con .0 para realizar la consulta

Tabla 4.12: Especificación caso de uso (2.1) Completar OID con .0.

Caso de uso	2.2 Ejecutar Operación SNMP
Actores	Usuario
Descripción	Permite ejecutar una operación SNMP definida en la versión 1 o 2c del protocolo.
Flujo básico	• Seleccionar la operación SNMP a realizar. • Configurar los parámetros de acuerdo a la operación. • Ejecutar la operación.

Tabla 4.13: Especificación caso de uso (2.2) Ejecutar Operación SNMP.

4.1.3 Iteración 3: Ping

El nivel 2 del módulo Ping está compuesto por 2 casos de uso, los cuales se muestran en la Figura 4.4.

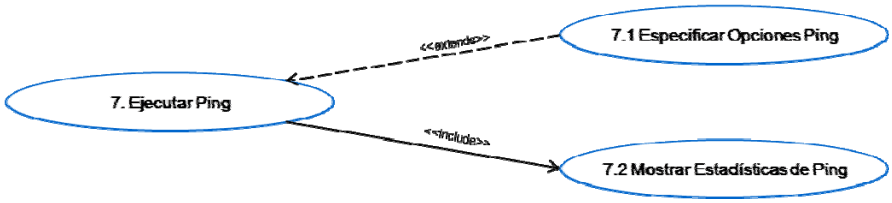


Figura 4.4: Diagrama de casos de uso nivel 2, Ping.

A continuación, en la Tabla 4.14 y Tabla 4.15 se muestra la especificación de los casos de uso (7.1) Especificar Opciones de Ping y (7.2) Mostrar Estadísticas de Ping.

Caso de uso	7.1 Especificar Opciones Ping
Actores	Usuario
Descripción	Permite configurar las diferentes opciones del módulo Ping relativas al paquete de petición eco que se va a enviar.
Flujo básico	• Configurar los valores de las opciones que sean de interés para el usuario.

Tabla 4.14: Especificación caso de uso (7.1) Especificar Opciones Ping.

Caso de uso	7.2 Mostrar Estadísticas de Ping
Actores	Usuario
Descripción	Muestra las estadísticas relativas al Ping, compuestas por cantidad de paquetes enviados, recibidos y perdidos y tiempos de ida y vuelta.
Flujo básico	• Al terminar la ejecución del Ping se muestra la estadística correspondiente con las peticiones realizadas.

Tabla 4.15: Especificación caso de uso (7.2) Mostrar Estadísticas de Ping.

4.1.4 Iteración 4: Grapher SNMP

En la Figura 4.5 se muestra el diagrama de casos de uso para el Grapher SNMP, en un segundo nivel de abstracción de acuerdo a las funcionalidades que debe ofrecer el mismo para mostrar resultados gráficos producto de la monitorización de elementos escogidos por el usuario.

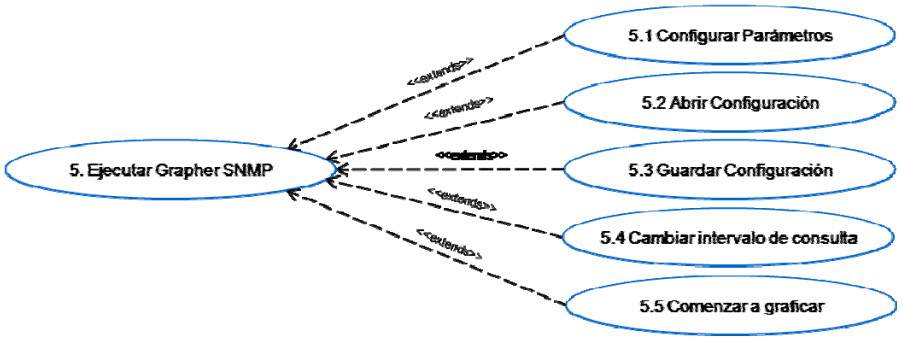


Figura 4.5: Diagrama de casos de uso nivel 2, Grapher SNMP.

La especificación de los casos de uso mostrados en la Figura 4.5 es descrita en la Tabla 4.16, Tabla 4.17, Tabla 4.18, Tabla 4.19 y Tabla 4.20.

Caso de uso	5.1 Configurar Parámetros
Actores	Usuario
Descripción	Muestra al usuario una nueva ventana para configurar los parámetros de captura.
Flujo básico	• Introducir los datos del agente que se quiere monitorizar. • Seleccionar el tipo de elementos a graficar (Interfaces o variables SNMP). • Escoger el subconjunto de elementos a graficar.

Tabla 4.16: Especificación caso de uso (5.1) Configurar Parámetros.

Caso de uso	5.2 Abrir Configuración
Actores	Usuario
Descripción	Permite al usuario cargar una configuración que se ha almacenado previamente, mostrar los gráficos correspondientes a la captura e iniciar la monitorización con la misma configuración.
Flujo básico	• Seleccionar el archivo de configuración que se desea abrir.

Tabla 4.17: Especificación caso de uso (5.2) Abrir Configuración.

Caso de uso	5.3 Guardar Configuración
Actores	Usuario
Descripción	Una vez que se detiene la monitorización, se permite al usuario guardar los valores de configuración y las gráficas obtenidas.
Flujo básico	• Seleccionar la ruta de destino donde se almacenará el archivo.

Tabla 4.18: Especificación caso de uso (5.3) Guardar Configuración.

Caso de uso	5.4 Cambiar intervalo de consulta
Actores	Usuario
Descripción	Permite personalizar el intervalo de tiempo con el que se realizarán las consultas SNMP y por ende cada cuánto se mostrará un valor en la gráfica.
Flujo básico	• Modificar el valor del campo Intervalo dentro del rango permitido.

Tabla 4.19: Especificación caso de uso (5.4) Cambiar intervalo de consulta.

Caso de uso	5.5 Comenzar a graficar
Actores	Usuario
Descripción	Una vez que se han configurado todos los valores para la monitorización, se puede iniciar la captura para comenzar a generar la gráfica correspondiente.
Flujo básico	• Iniciar la monitorización de las variables seleccionadas.

Tabla 4.20: Especificación caso de uso (5.5) Comenzar a graficar.

4.1.5 Iteración 5: MIB Parser

En la iteración 5, se realizó el análisis para el módulo correspondiente con el MIB Parser o compilador de MIBs. El detalle del nivel 2 del diagrama de casos de uso es mostrado en la Figura 4.6.

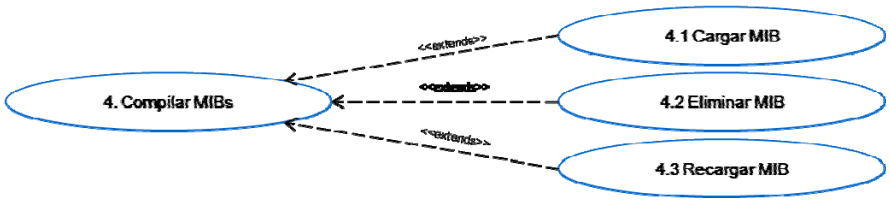


Figura 4.6: Diagrama de casos de uso nivel 2, MIB Parser.

La especificación de los casos de uso mostrada en la Figura 4.6 es descrita en la Tabla 4.21, Tabla 4.22 y Tabla 4.23.

Caso de uso	4.1 Cargar MIB
Actores	Usuario
Descripción	Permite cargar una nueva MIB.
Flujo básico	• Escoger el archivo que corresponde con la MIB que se desea cargar.

Tabla 4.21: Especificación caso de uso (4.1) Cargar MIB.

Caso de uso	4.2 Eliminar MIB
Actores	Usuario
Descripción	Permite al usuario eliminar una MIB que ha sido previamente cargada.
Flujo básico	• Seleccionar la MIB que se desea eliminar de la estructura que las contiene.

Tabla 4.22: Especificación caso de uso (4.2) Eliminar MIB.

Caso de uso	4.3 Recargar MIB
Actores	Usuario
Descripción	Carga en la estructura jerárquica, todas las MIBs que estén en la carpeta de configuración.
Flujo básico	• Cargar dentro del árbol las MIBs que se encuentren dentro de la carpeta de configuración.

Tabla 4.23: Especificación caso de uso (4.3) Recargar MIB.

4.1.6 Iteración 6: Idioma

En un segundo nivel de abstracción, en el módulo de idioma se especifica un caso de uso, mostrado en la Figura 4.7 y descrito en la Tabla 4.24.

Es importante señalar que el módulo de idioma surge como un requerimiento adicional luego de haber comenzado la creación de la aplicación, por lo que los módulos que ya habían sido elaborados (iteraciones de la 1 a la 5) fueron adaptados en este punto al nuevo requerimiento.

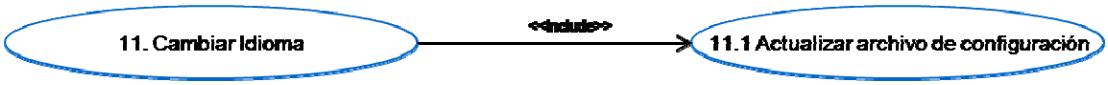


Figura 4.7: Diagrama de casos de uso nivel 2, Idioma.

Caso de uso	11.1 Actualizar archivo de configuración
Actores	Usuario
Descripción	Actualiza el archivo de configuración de idioma de la aplicación, con el seleccionado por el usuario.
Flujo básico	• Se actualiza el archivo de configuración de idioma de la aplicación.

Tabla 4.24: Especificación caso de uso (11.1) Actualizar archivo de configuración.

4.1.7 Iteración 7: Tabla Genérica

La Figura 4.8 muestra el detalle del diagrama de casos de uso en un nivel 2 para el módulo Tabla Genérica, la Tabla 4.25 describe la especificación del caso de uso (3.1) Configurar Parámetros de Consulta.

Figura 4.8: Diagrama de casos de uso nivel 2, Tabla Genérica.

Caso de uso	3.1 Configurar Parámetros de Consulta
Actores	Usuario
Descripción	Se muestra una nueva ventana donde se indican los parámetros de configuración.
Flujo básico	• Especificar los parámetros de configuración.

Tabla 4.25: Especificación caso de uso (3.1) Configurar Parámetros de Consulta.

4.1.8 Iteración 8: Sniffer

Las principales funcionalidades del Sniffer se muestran en la Figura 4.9 que representa el diagrama de casos de uso de nivel 2 de dicho módulo.

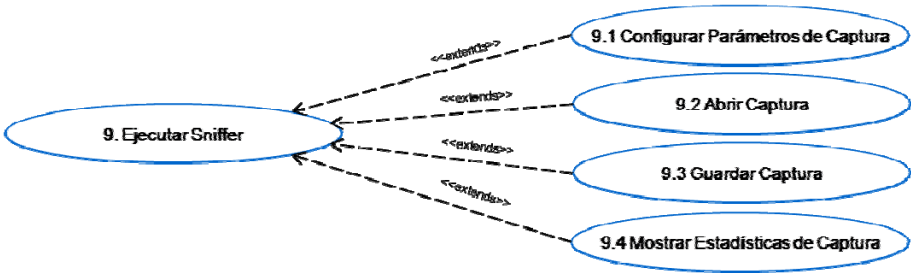


Figura 4.9: Diagrama de casos de uso nivel 2, Sniffer.

La especificación de cada uno de los casos de uso mostrados en la Figura 4.9 es descrita en la Tabla 4.26, Tabla 4.27, Tabla 4.28 y Tabla 4.29.

Caso de uso	9.1 Configurar Parámetros de Captura
Actores	Usuario
Descripción	Permite al usuario indicar los parámetros de captura, seleccionar la interfaz por la que se desea capturar, establecer filtros de precaptura, etc.
Flujo básico	• Especificar los diferentes parámetros de captura. • Iniciar la captura.

Tabla 4.26: Especificación caso de uso (9.1) Configurar Parámetros de Captura.

Caso de uso	9.2 Abrir Captura
Actores	Usuario
Descripción	Carga capturas que han sido previamente almacenadas y permite ver el detalle de las mismas.
Flujo básico	• Seleccionar el archivo de captura que se desea cargar.

Tabla 4.27: Especificación caso de uso (9.2) Abrir Captura.

Caso de uso	9.3 Guardar Captura
Actores	Usuario
Descripción	Luego que se detiene la captura, se permite al usuario almacenarla para su posterior utilización. Los archivos son almacenados con la extensión .cap.
Flujo básico	• Seleccionar la ruta de destino donde se va a almacenar la captura.

Tabla 4.28: Especificación caso de uso (9.3) Guardar Captura.

Caso de uso	9.4 Mostrar Estadísticas de Captura
Actores	Usuario
Descripción	Muestra al usuario las estadísticas correspondientes con la cantidad de paquetes capturados agrupados por protocolo.
Flujo básico	• Generar la estadística de la captura.

Tabla 4.29: Especificación caso de uso (9.4) Mostrar Estadística de Captura.

4.1.9 Iteración 9: Tracert

La iteración 9 corresponde con el módulo Tracert, el cual en un nivel 2 de abstracción se desglosa en un nuevo caso de uso, correspondiente con el caso de uso (8.1) Especificar Opciones Tracert. Dicho diagrama es mostrado en la Figura 4.10 y la especificación del mismo en la Tabla 4.31.

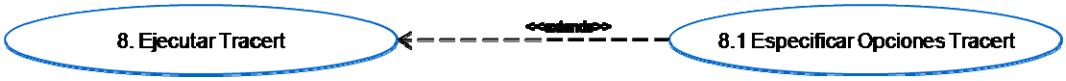


Figura 4.10: Diagrama de casos de uso nivel 2, Tracert.

Caso de uso	8.1 Especificar Opciones Tracert
Actores	Usuario
Descripción	Permite configurar las diferentes opciones del módulo Tracert relativas a la traza que se desea realizar.
Flujo básico	• Configurar los valores de las opciones que sean de interés para el usuario.

Tabla 4.30: Especificación caso de uso (8.1) Especificar Opciones Tracert.

4.1.10 Iteración 10: Receptor de Traps

Al igual que la iteración anterior, el Receptor de Traps en un nivel 2 de abstracción se desglosa en un nuevo caso de uso correspondiente con (6.1) Configurar Filtros.

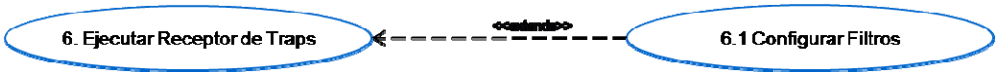


Figura 4.11: Diagrama de casos de uso nivel 2, Receptor de Traps.

La Figura 4.11 representa el diagrama de casos de uso en un segundo nivel de abstracción y la Tabla 4.31 la especificación del mismo.

Caso de uso	6.1 Configurar Filtros
Actores	Usuario
Descripción	Permite al usuario filtrar la(s) dirección(es) IP (IPv4 y/o IPv6) de la(s) cual(es) se desea recibir traps o alertas.
Flujo básico	• Agregar la(s) dirección(es) IP que se desea(n) establecer como filtro.

Tabla 4.31: Especificación caso de uso (6.1) Configurar Filtros.

4.1.11 Iteración 11: Escáner de Puertos

La última iteración se inicia con el estudio de los requerimientos para el escáner de puertos. La Figura 4.12 muestra el diagrama de casos de uso correspondiente con esta iteración. Adicionalmente, en la Tabla 4.32 se puede observar la descripción de la especificación del mismo.

Figura 4.12: Diagrama de casos de uso nivel 2, Escáner de Puertos.

Caso de uso	10.1 Definir Rango de Puertos a Escanear
Actores	Usuario
Descripción	Permite al usuario definir el rango de puertos que se desea escanear.
Flujo básico	• Especificar el rango a escanear.

Tabla 4.32: Especificación caso de uso (10.1) Definir Rango de Puertos a Escanear.

4.2 Fase de Diseño

Al finalizar la fase de análisis de cada iteración, donde se definen los requerimientos y se crean los diagramas de casos de uso de cada módulo, (los cuales reflejan las funcionalidades a implementar) se procede a diseñar las soluciones que mejor se adapten a las necesidades planteadas. Para esto, se define la estructura de los elementos a interactuar en la aplicación.

Posterior al diseño de la estructura, se definen las clases y con ello los diagramas de clases. Al igual que en la fase anterior y por simplicidad, los diagramas de clases son mostrados por iteración y divididos en distintos niveles de abstracción para detallar todas las clases que intervienen en cada módulo y cómo se relacionan.

La documentación mostrada en esta fase corresponde con los diagramas de clases obtenidos durante cada iteración luego de diversas refinaciones (los cuales reflejan en su mayoría la última refinación realizada a cada diagrama). Dentro de la documentación de las primeras 5 iteraciones se notará el uso del módulo para el cambio de idioma, a pesar de que dicho módulo surgió como un nuevo requerimiento diseñado en la iteración 6. Esto se debe, a que a partir de esta iteración todos los módulos existentes fueron adaptados a la nueva funcionalidad.

A continuación se muestra el significado de los símbolos utilizados dentro de los diagramas de clases para especificar el tipo de estructura con la que se está trabajando y para definir la visibilidad de los atributos y operaciones de cada clase:

- Indica que la estructura tratada es un paquete.
- Indica que la estructura definida es una clase.
- **+** Definición de un atributo o un método público dentro de la clase.
- **#** Definición de un atributo o un método protegido dentro de la clase.
- **-** Definición de un atributo o un método privado dentro de la clase.

4.2.1 Iteración 1: General

En esta primera iteración se desarrolla la estructura general de la aplicación y la base para cada uno de los módulos que son implementados en las diferentes iteraciones. El proceso se inicia definiendo la clase principal, su estructura y creando los paquetes para agrupar las futuras clases de la aplicación.

Los paquetes se diseñaron de manera tal que agrupen funcionalidades comunes y que cuenten con distintos niveles de anidación. Entre los paquetes principales se encuentran los siguientes:

- estructuras: diseñado para almacenar clases que representen elementos dentro de la aplicación.
- idioma: tiene como finalidad alojar a las clases relacionadas con el cambio de idioma.
- operaciones: este paquete almacenará todas las clases que realicen procedimientos no específicos a un módulo sino generales a toda la aplicación.
- principal: creado con la finalidad de agrupar las clases relacionadas con el entorno principal de la aplicación.
- snmp: es un paquete que agrupa a otros paquetes que implementan módulos relacionados con funcionalidades SNMP. Dentro del paquete snmp se encuentran los paquetes: manager, grapher, traps, mibparser y tabla.
- tools: agrupa otros paquetes asociados con las herramientas básicas para la administración de redes. Dentro del paquete tools se encuentran los paquetes: ping, sniffer, tracert y escaner.

Adicionalmente, se crean las clases principales para cada uno de los módulos que son desarrollados posteriormente. Entre las clases principales definidas en este nivel se encuentran: PanelPrincipal, ManagerPanel, GrapherPanel, TrapsPanel, TablaPanel, PingPanel, TracertPanel, SnifferPanel y EscanerPanel, que representan las clases principales de cada uno de los módulos que pueden ser accedidos desde el menú principal a través de las opciones: Manager SNMP, Grapher SNMP, Receptor de Traps, Tabla Genérica, Ping, Tracert, Sniffer y Escáner de Puertos, respectivamente.

En la Figura 4.13 se muestra el diagrama de clases inicial, donde se detalla la estructura propuesta para la clase principal de la aplicación llamada AdminUcvNGN y se puede apreciar también la estructura de paquetes diseñada como base para los futuros módulos.

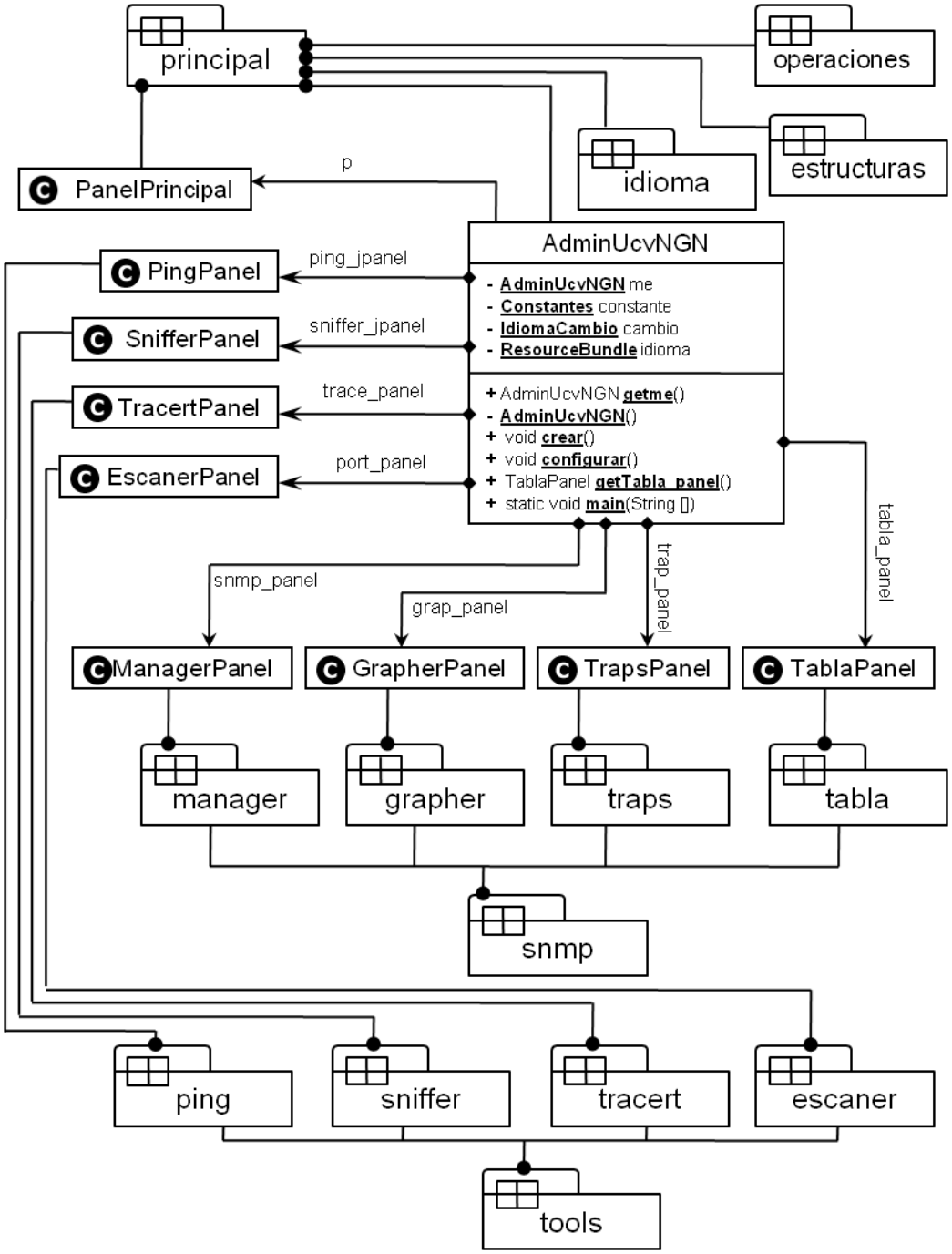


Figura 4.13: Diagrama de clases principal.

4.2.2 Iteración 2: Manager SNMP

En la presente iteración se describen las diferentes clases que conforman el módulo del Manager SNMP y cómo se relacionan. Para facilitar la compresión del diagrama de clases, fue dividido en dos niveles según el grado de abstracción. El primer nivel refleja las clases utilitarias manejadas

por el manager y en un segundo nivel se muestra la clase que compone al manager y la interrelación de ésta con las clases utilitarias.

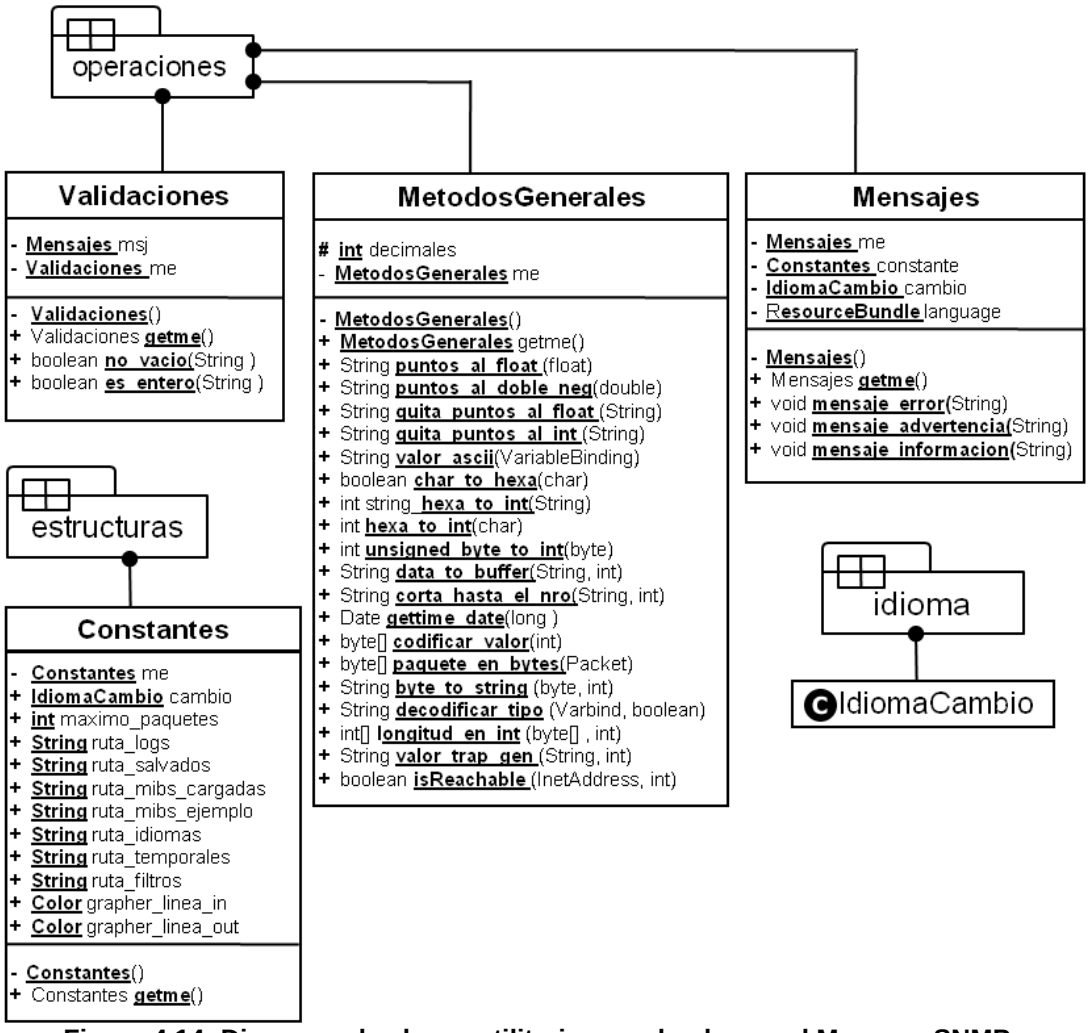


Figura 4.14: Diagrama de clases utilitarias empleadas en el Manager SNMP.

La Figura 4.14 muestra el detalle de la estructura de cada una de las clases utilitarias que son empleadas por el manager y se especifica el paquete al cual pertenecen. A continuación, se presenta una breve descripción de cada una de estas clases:

- estructuras.Constantes: define un conjunto de variables que son usadas en los diferentes módulos del sistema y mantienen valor constante a lo largo de la aplicación.
- operaciones.Validaciones: agrupa un conjunto de métodos para validaciones comunes dentro de la aplicación.

- operaciones.MetodosGenerales: define una serie de métodos usados para funciones generales como manejo de datos, codificación y decodificación de datos, conversión de tipos de datos, etc.
- operaciones.Mensajes: especifica funciones para personalizar los tres tipos de mensajes mostrados en el sistema, los cuales se clasifican en: mensajes de error, mensajes de advertencia y mensajes de información.
- idioma.IdiomaCambio: representa una clase intermedia para consultar el valor actual del idioma configurado para la aplicación. La especificación de esta clase será detallada en la iteración 6.

En la Figura 4.15 se pueden observar las clases utilitarias descritas anteriormente, cómo se compone el paquete manager y cómo se interrelacionan sus clases con las clases utilitarias.

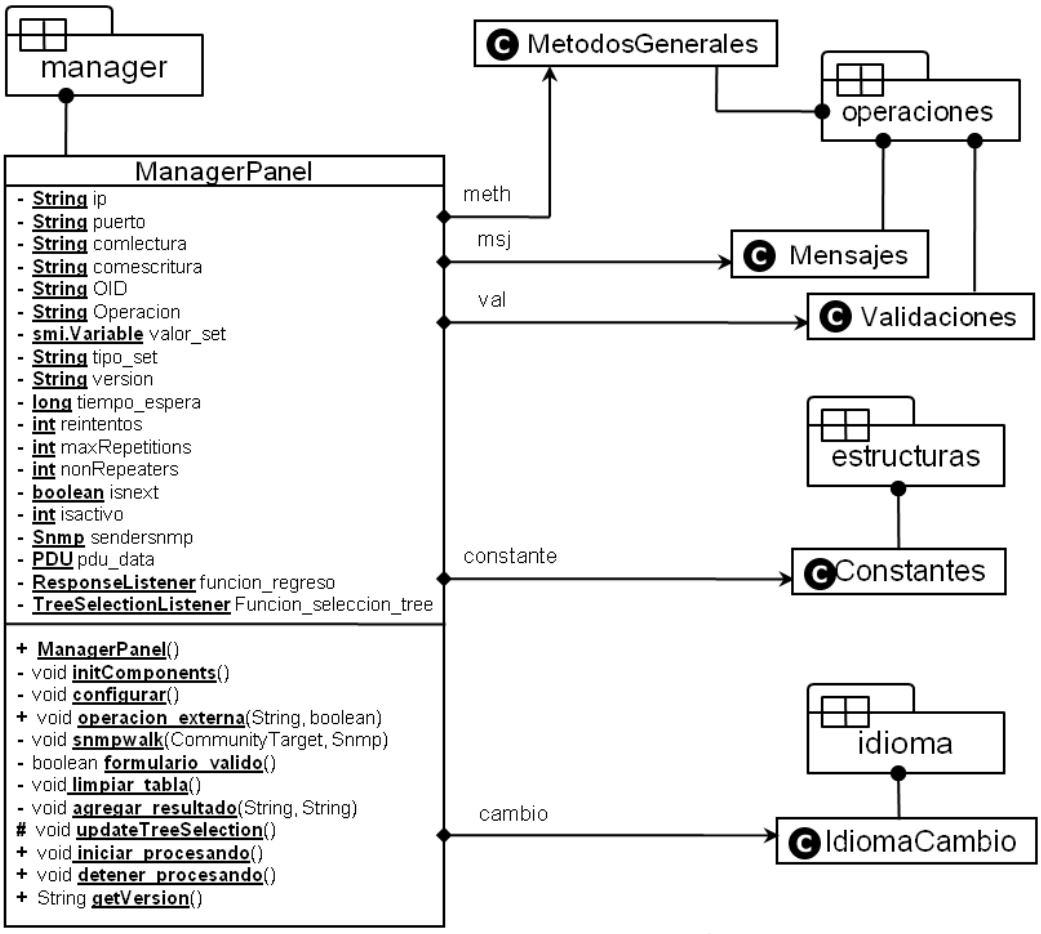


Figura 4.15: Diagrama de clases y relaciones del módulo Manager SNMP.

A continuación se describe la clase que compone al paquete manager:

- snmp.manager.ManagerPanel: define la interfaz gráfica para el Manager SNMP y aloja los métodos que se encargan de manejar todos los eventos activados dentro de la clase, como por ejemplo, validación de los parámetros introducidos para la consulta, la ejecución de las diferentes operaciones SNMP, etc.

4.2.3 Iteración 3: Ping

Al igual que en la iteración anterior, el diagrama de clases del Ping se dividió en dos niveles de acuerdo al grado de abstracción aplicado sobre el mismo. El nivel 1 representa las clases utilitarias que emplea el paquete ping para la realización de funciones generales y en un segundo nivel de abstracción se muestran el conjunto de clases que componen el paquete y la relación entre ellas y las clases utilitarias.

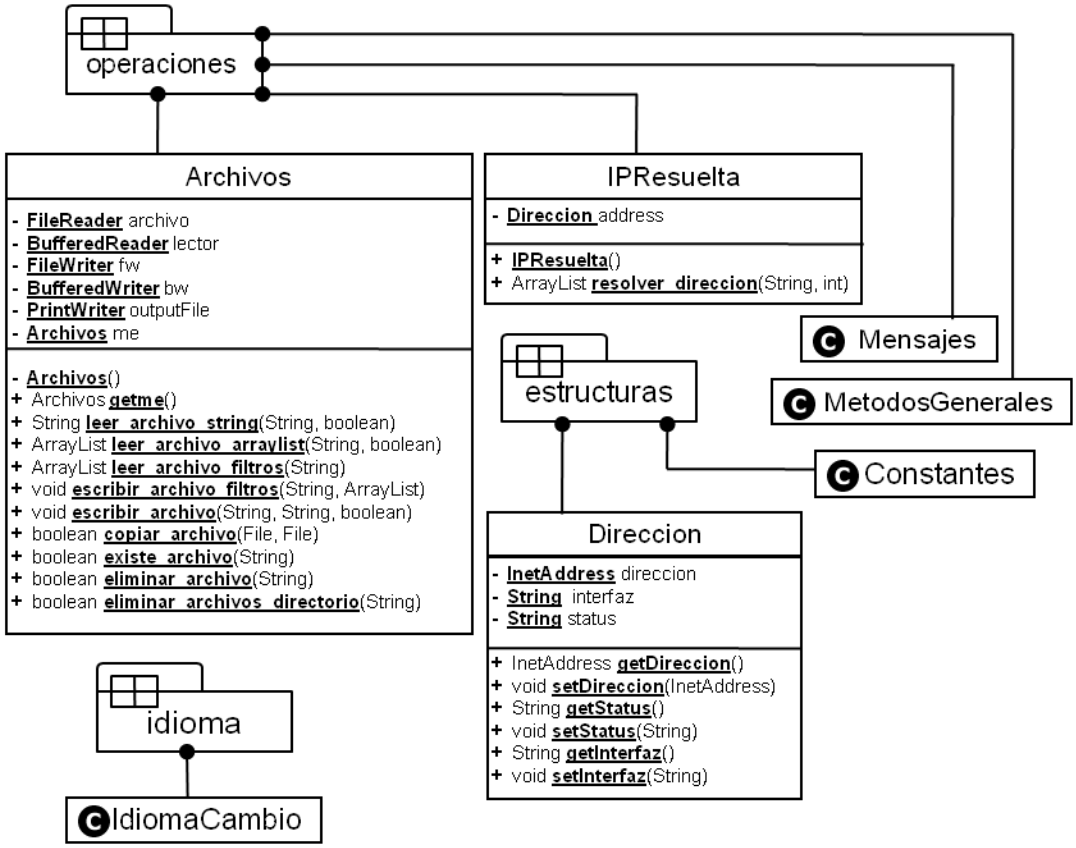


Figura 4.16: Diagrama de clases utilitarias empleadas en el módulo Ping.

La Figura 4.16 refleja las clases que componen el primer nivel de abstracción. De manera similar en la Figura 4.17 se puede detallar las clases que componen al paquete ping y su relación con las diferentes clases utilitarias.

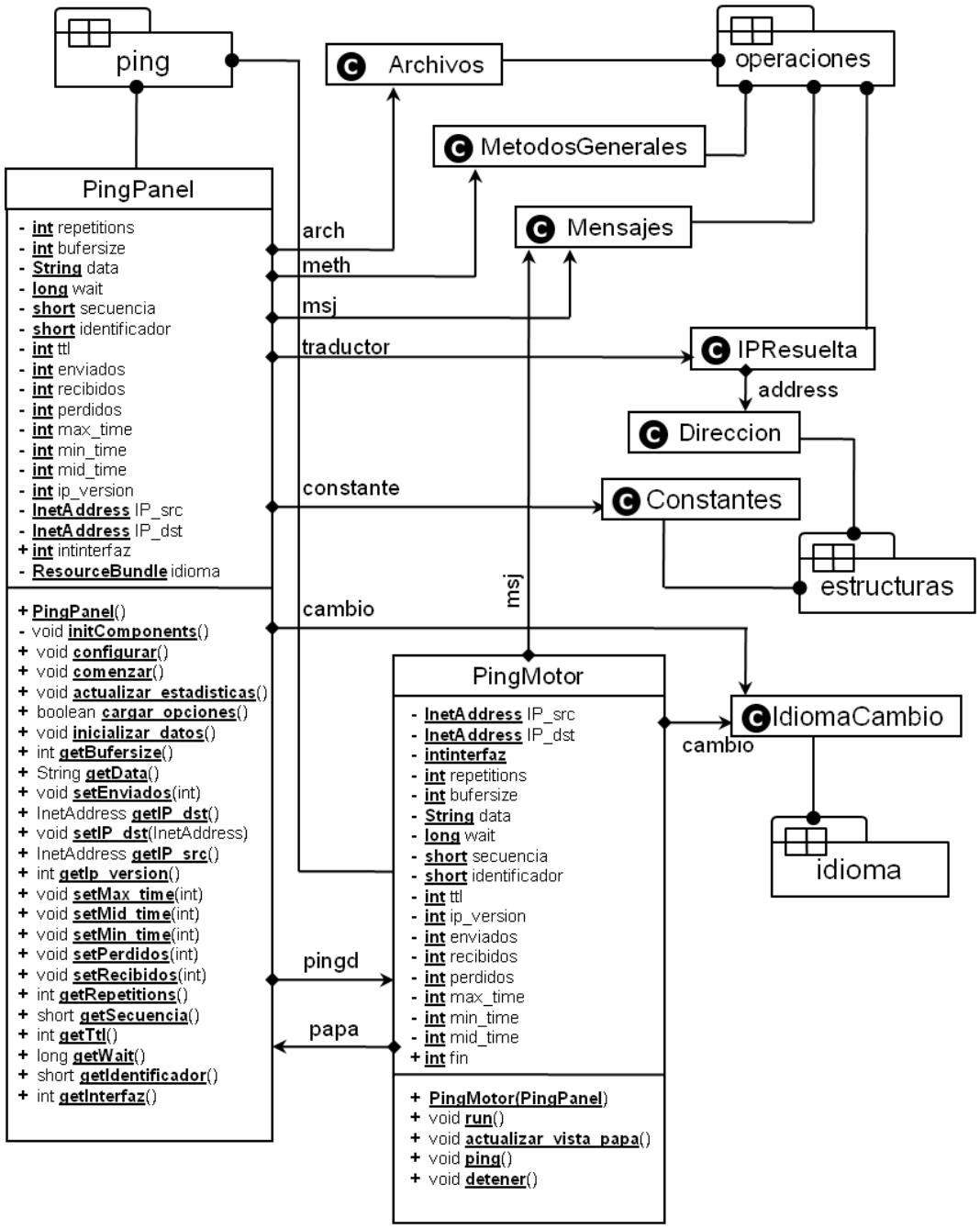


Figura 4.17: Diagrama de clases y relaciones del módulo Ping.

A continuación se presenta la descripción de las clases exhibidas en la Figura 4.16 y Figura 4.17:

- estructuras.Direccion: define una estructura que permite almacenar la dirección de un host junto con el status e interfaz (si la tiene) de la misma.

- operaciones.Archivos: presenta una serie de métodos, para el manejo de archivos de texto. Permite crear, modificar y eliminar archivos de texto. Adicionalmente, provee funciones que permiten leer y escribir en archivos así como copiar el contenido de un archivo hacia otro.
- operaciones.IPResuelta: representa una clase intermediaria entre el archivo ejecutable creado en el lenguaje C++ y Java para resolver las direcciones Fuente y Destino.
- tools.ping.PingPanel: define la interfaz gráfica de usuario del módulo, recoge los valores de las diferentes opciones del ping y llama a la clase PingMotor para que realice el envío de los mensajes ICMP (ICMPv4 o ICMPv6) de acuerdo a la versión del protocolo IP que se esté utilizando.
- tools.ping.PingMotor: ejecuta un hilo para realizar el envío de mensajes de solicitudes de eco al host indicado, tomando en cuenta la versión del Protocolo IP.

4.2.4 Iteración 4: Grapher SNMP

Debido a que en esta iteración se tiene como finalidad implementar un módulo que monitorice un dispositivo de red y realice gráficas de los resultados obtenidos, se decide separar la lógica del mismo en tres componentes: interfaz gráfica de usuario, monitorización del dispositivo y gráfica de resultados. Todas las clases de los componentes están ubicadas dentro del paquete grapher, desde el cual se emplea el uso de diversas clases utilitarias.

Por tanto, los diagramas de clase para esta iteración se presentan en tres niveles: un primer nivel donde se detallan las clases utilitarias a emplear, un segundo nivel donde se definen las clases que implementan los componentes de monitorización y el componente para graficar y un tercer nivel donde se aprecia el diagrama de clases que detalla la definición del componente para la interfaz gráfica de usuario, así como las relaciones entre las clases de los tres componentes.

En la Figura 4.18 se muestra el diagrama de clase para las clases utilitarias empleadas en este módulo. Entre las clases utilitarias se encuentran: Constantes, Archivos, MetodosGenerales, Mensajes, IdiomaCambio y Validaciones que fueron definidas previamente y la clase JSeleccionarColor.

- estructuras.JSeleccionarColor: permite definir una ventana para la selección del color de las líneas del gráfico. Dicha clase utiliza a la clase PanelPrevista para cumplir con sus objetivos.

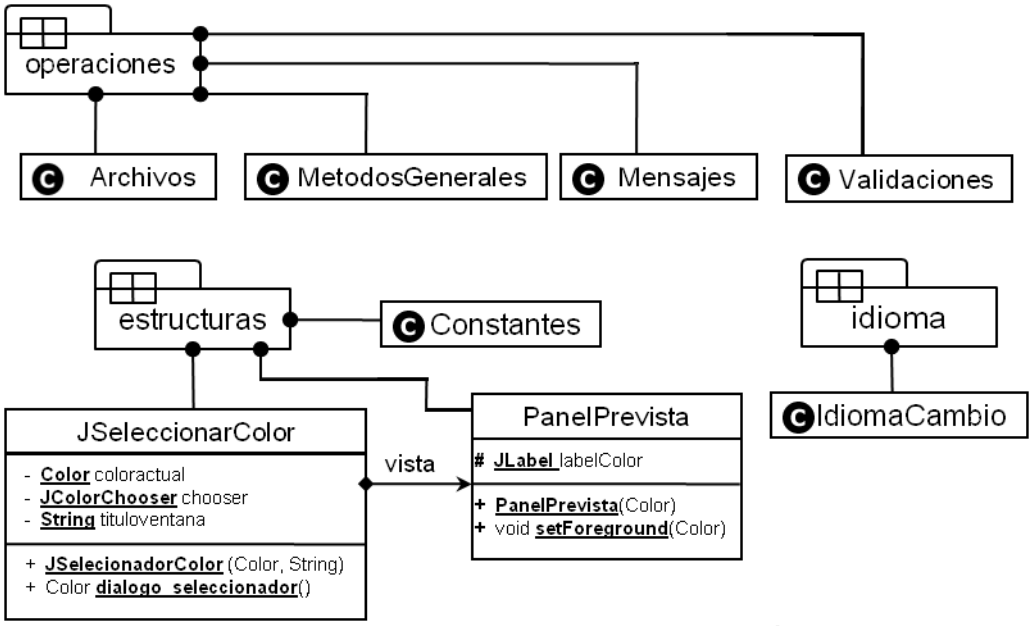


Figura 4.18: Diagrama de clases utilitarias empleadas en el módulo Grapher SNMP.

El siguiente componente de este módulo es el encargado de monitorizar periódicamente el dispositivo, el cual está diseñado mediante la creación de las clases GrapherMotor y demonio_graficador.

- snmp.grapher.GrapherMotor: se encarga de almacenar toda la información de la conexión SNMP y de servir como interfaz para invocar a la clase demonio_graficador.
- snmp.grapher.demonio_graficador: se ejecuta de manera asíncrona y tiene como objetivo interrogar al dispositivo periódicamente.

En la Figura 4.19 se puede apreciar el diagrama donde se detalla la definición de estas dos clases.

De la misma forma se diseña el componente para la creación de los gráficos, que está constituido por la clase GrapherPantallaPanel y GrapherTacometroPanel.

- snmp.grapher.GrapherPantallaPanel: es la encargada de mostrar el resultado gráfico de los valores obtenidos.
- snmp.grapher.GrapherTacometroPanel: muestra el detalle del último valor graficado.

En la Figura 4.20 se detalla el diagrama de clases donde se muestra la definición de las clases GrapherPantallaPanel y GrapherTacometroPanel diseñadas para implementar el módulo de creación de gráficos.

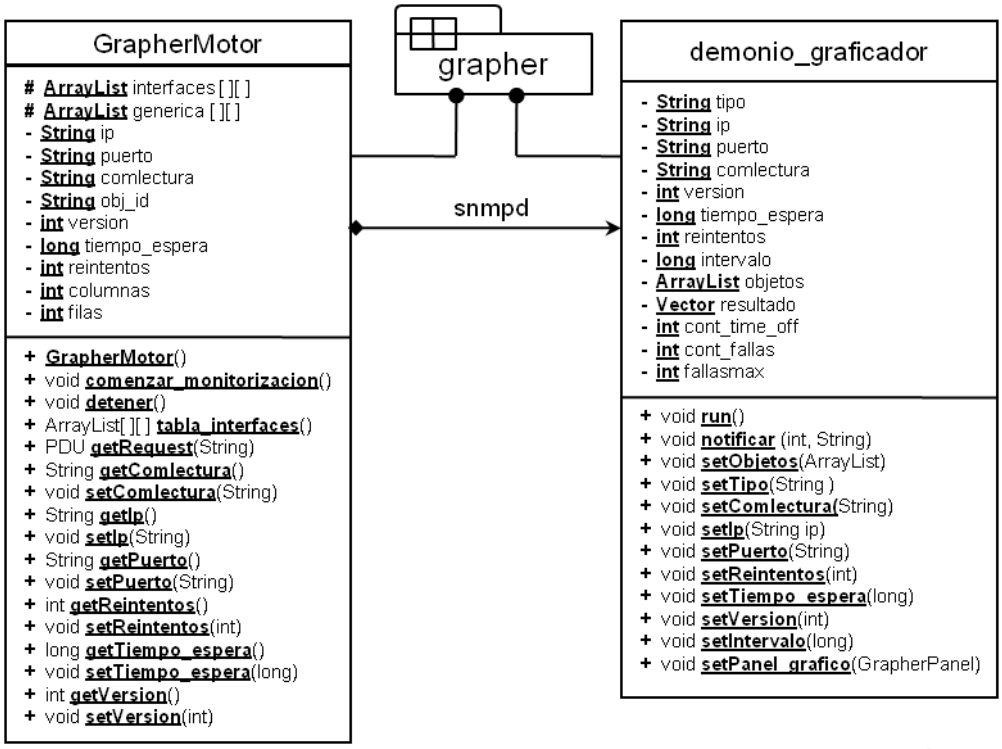


Figura 4.19: Diagrama de clases para el componente de monitorización.

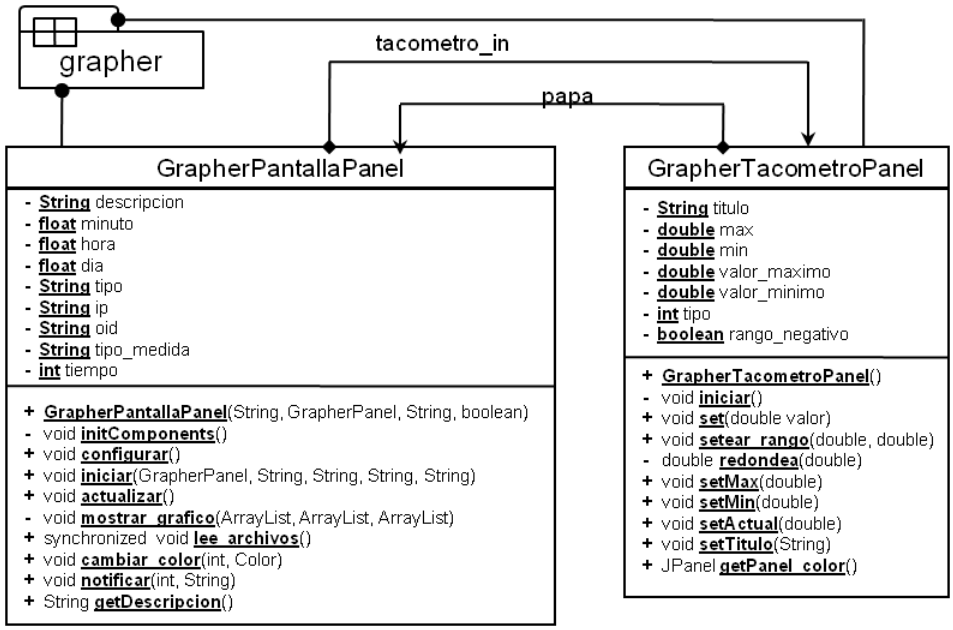


Figura 4.20: Diagrama de clases para el componente de creación de gráficos.

En la Figura 4.21 se pueden observar las relaciones entre las clases que componen el módulo grapher y adicionalmente se puede detallar el diseño de las clases para el componente de interfaz con el usuario, la cual está formada por las siguientes clases:

- snmp.grapher.GrapherPanel: representa la clase principal del módulo, la cual presenta al usuario funcionalidades básicas para interactuar con la herramienta y desde donde se puede acceder a la ventana de configuración definida en la clase GrapherConfiguraFrame.
- snmp.grapher.GrapherConfiguraFrame: permite especificar los valores para la conexión SNMP, así como definir el conjunto de valores a graficar.

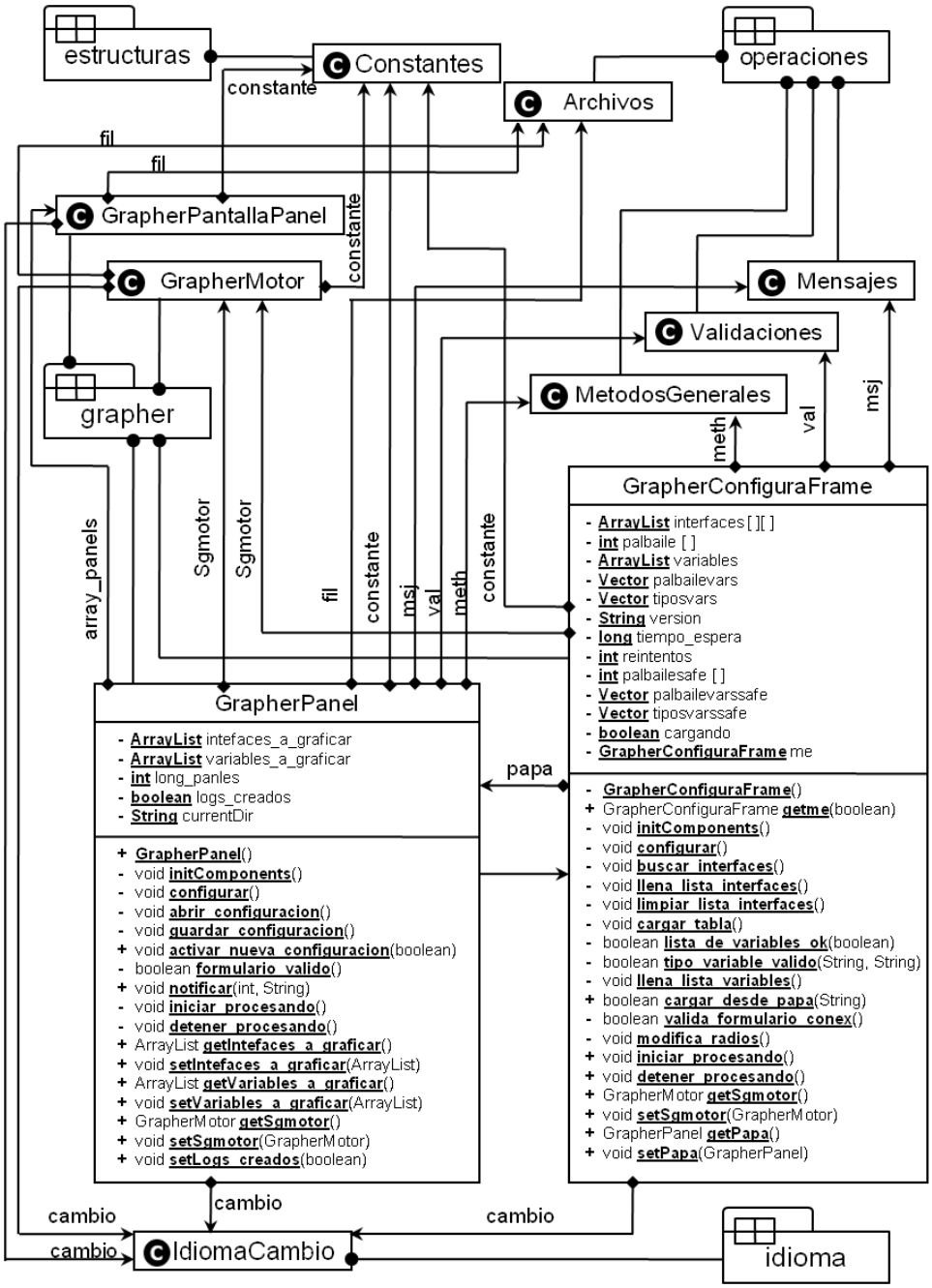


Figura 4.21: Diagrama de clases y relaciones del módulo Grapher SNMP.

4.2.5 Iteración 5: MIB Parser

Esta iteración tiene como objetivo desarrollar una funcionalidad que permita compilar MIBs, facilitando al usuario la consulta de variables. A diferencia de los módulos previos, esta funcionalidad debe ser accedida desde otro módulo. Entre las clases utilitarias empleadas para diseñar dicha funcionalidad se encuentran las clases Archivos, Mensajes, IdiomaCambio y Constantes que fueron descritas previamente.

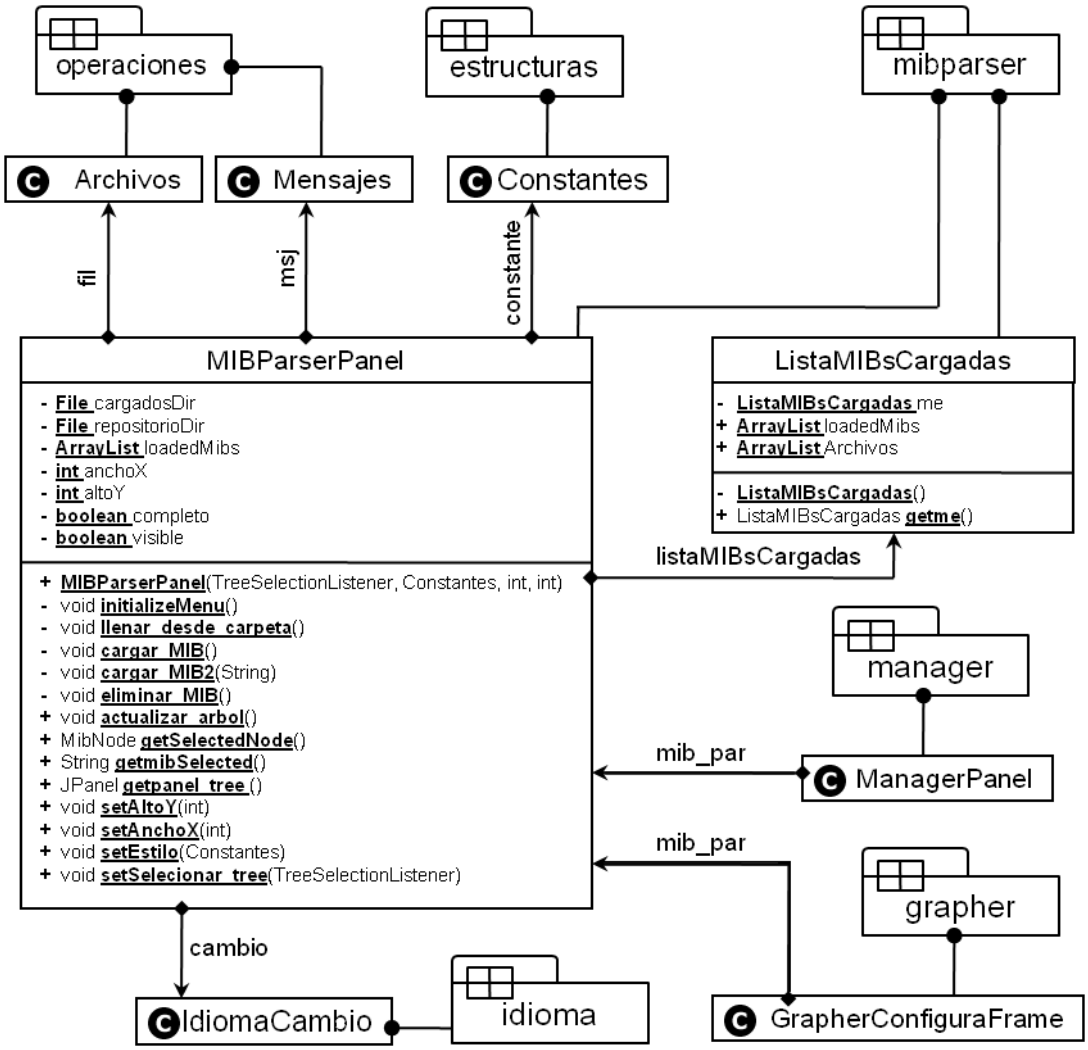


Figura 4.22: Diagrama de clases y relaciones del módulo MIB Parser.

Para el diseño de este módulo se crearon dos clases que son descritas a continuación:

- snmp.mibparser.MIBParserPanel: define la interfaz para acceder a los valores de las MIBs compiladas.

- snmp.mibparser.ListaMIBsCargadas: es accedida desde la clase MIBParserPanel, que la utiliza para mantener una referencia única de la lista de MIBs cargadas en la aplicación.

En la Figura 4.22 se observa el detalle de la definición de las clases MIBParserPanel y ListaMIBsCargadas. Adicionalmente, se puede apreciar las relaciones de estas clases con las clases utilitarias y con las clases ManagerPanel y GrapherConfiguraFrame, desde las cuales se puede acceder a este módulo.

4.2.6 Iteración 6: Idioma

Luego de las primeras cinco iteraciones surgió un nuevo requerimiento, que consiste en agregar un módulo a la aplicación que permita que la misma esté disponible en múltiples idiomas. Para esto se diseñaron las siguientes clases:

- idioma.IdiomaPanel: diseñada para ser una ventana de configuración gráfica que le permita al usuario seleccionar el idioma a utilizar.
- idioma.IdiomaCambio: creada con la finalidad de almacenar la referencia a un objeto de tipo ResourceBundle, el cual contiene el acceso al archivo con el idioma seleccionado.

En la Figura 4.23 se puede observar el detalle de las clases que componen al paquete idioma y la relación que éstas tienen con las clases utilitarias Constantes, Mensajes y Archivos.

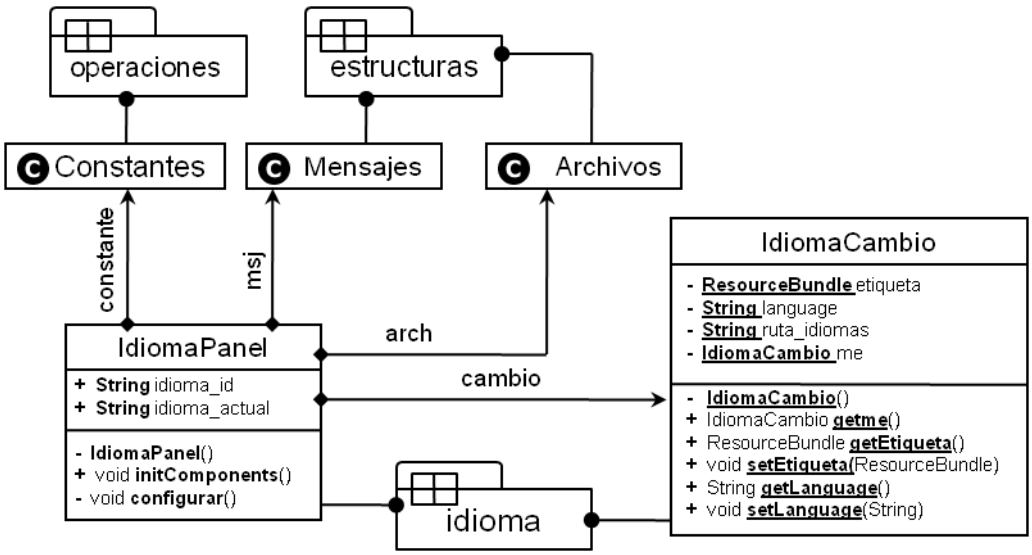


Figura 4.23: Diagrama de clases módulo Idioma.

Luego que se finalizó la creación de este módulo, se refactorizaron los módulos creados en la primeras cinco iteraciones para que los mismos se adaptaran a la nueva funcionalidad del cambio de idioma.

4.2.7 Iteración 7: Tabla Genérica

Otra de las funcionalidades que complementan el diseño de la aplicación es la de ofrecer la capacidad al usuario de ver los resultados de los datos que representan tablas SNMP ordenados en forma columnar.

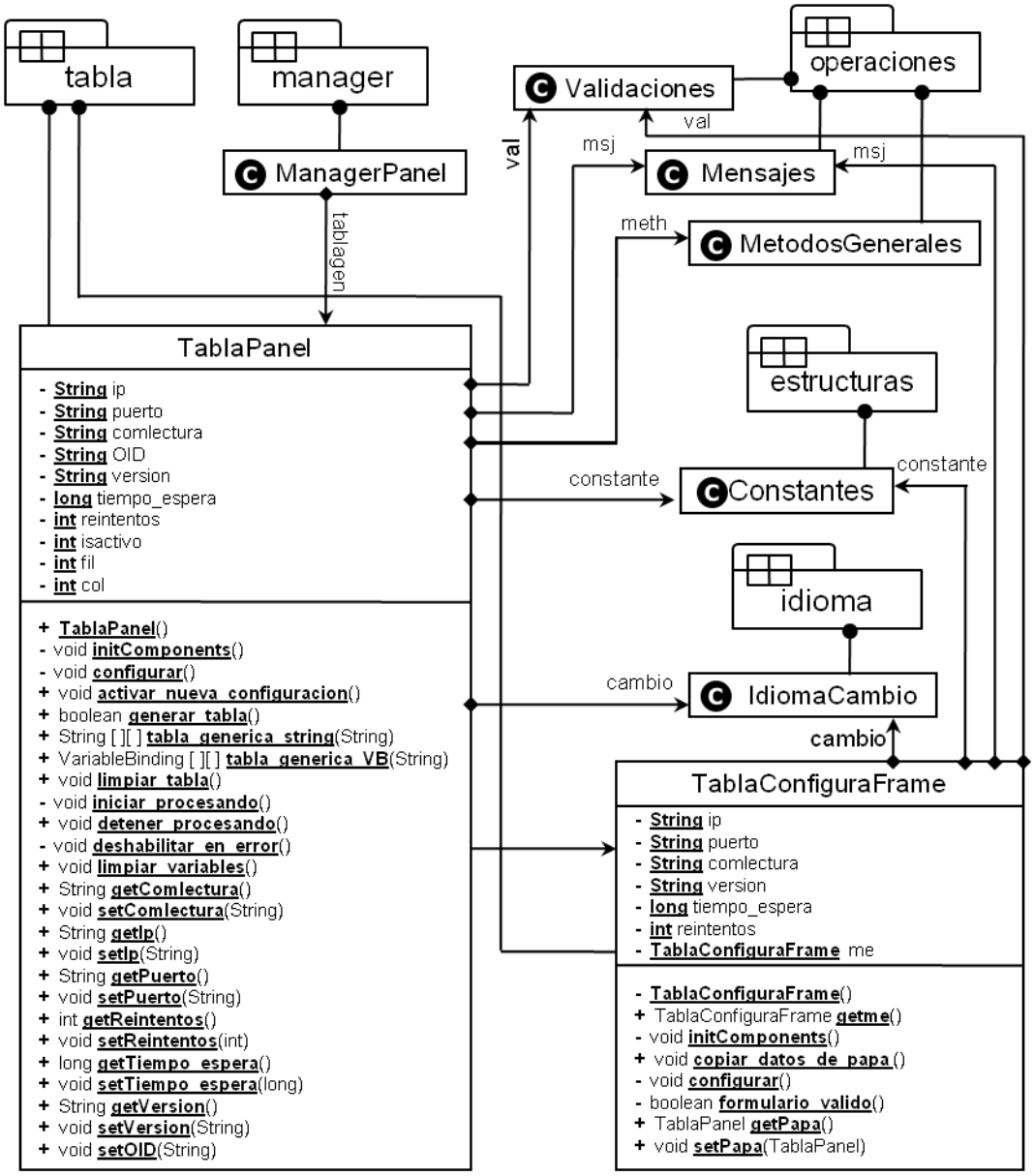


Figura 4.24: Diagrama de clases Tabla SNMP.

Para el diseño de esta funcionalidad se definen dos clases:

- snmp.tabla.TablaPanel: presenta la interfaz gráfica, que le permite al usuario ver los valores de las tablas SNMP y permite invocar a la clase TablaConfiguraFrame.
- snmp.tabla.TablaConfiguraFrame: permite especificar los valores para la conexión vía SNMP. Para esto, se crea una nueva ventana que permite ingresar los datos.

Adicionalmente, la clase TablaPanel puede ser invocada como una operación avanzada desde el módulo Manager SNMP. En la Figura 4.24 se puede apreciar el diseño de las clases TablaPanel y TablaConfiguraFrame y las relaciones que estas clases tienen con las clases ManagerPanel, Validaciones, Mensajes, MetodosGenerales, Constantes e IdiomaCambio que fueron descritas previamente.

4.2.8 Iteración 8: Sniffer

Durante esta iteración se diseñaron las clases apropiadas para la creación de un módulo que permitiera capturar los datos que transitan por la red, para luego analizar los mismos. Estas funcionalidades se agrupan en el módulo llamado Sniffer.

Debido a la cantidad de aspectos que abarca la creación de un sniffer, se decide separar el diseño de las clases en cuatro componentes según el trabajo que realizan:

- Configuración: componente creado para permitirle al usuario especificar los detalles de la captura.
- Captura: este componente contiene las clases diseñadas para capturar los datos.
- Análisis: tiene como finalidad proveer las clases que se encargan de analizar y determinar las características de los datos capturados.
- Estadísticas: este conjunto de clases colaboran para presentarle al usuario las estadísticas de los datos capturados.

En la Figura 4.25 se aprecia el diagrama donde se detallan las clases diseñadas para la creación de los componentes de configuración y estadísticas. A continuación se presenta una breve explicación de cada una de las clases que integran estos dos componentes:

- estructuras.Filtro: define una estructura para almacenar los detalles de un filtro.

- tools.sniffer.SnifferConfiguraFrame: crea una ventana que le permite al usuario configurar los parámetros para la captura de datos y adicionalmente permite acceder a la clase SnifferFiltros.
- tools.sniffer.SnifferFiltros: presenta una ventana al usuario que permite definir y almacenar nuevos filtros de captura. Utiliza a la clase Filtro para almacenar los datos de los filtros configurados en la aplicación.
- estructuras.EstadisticaCaptura: almacena información de la cantidad de paquetes capturados por cada protocolo soportado en el sniffer.
- tools.sniffer.SnifferEstadisticas: muestra de manera gráfica las estadísticas de los datos capturados. Los datos mostrados en las estadísticas se extraen de un objeto de tipo EstadisticaCaptura que como se describió anteriormente, contiene esta información.

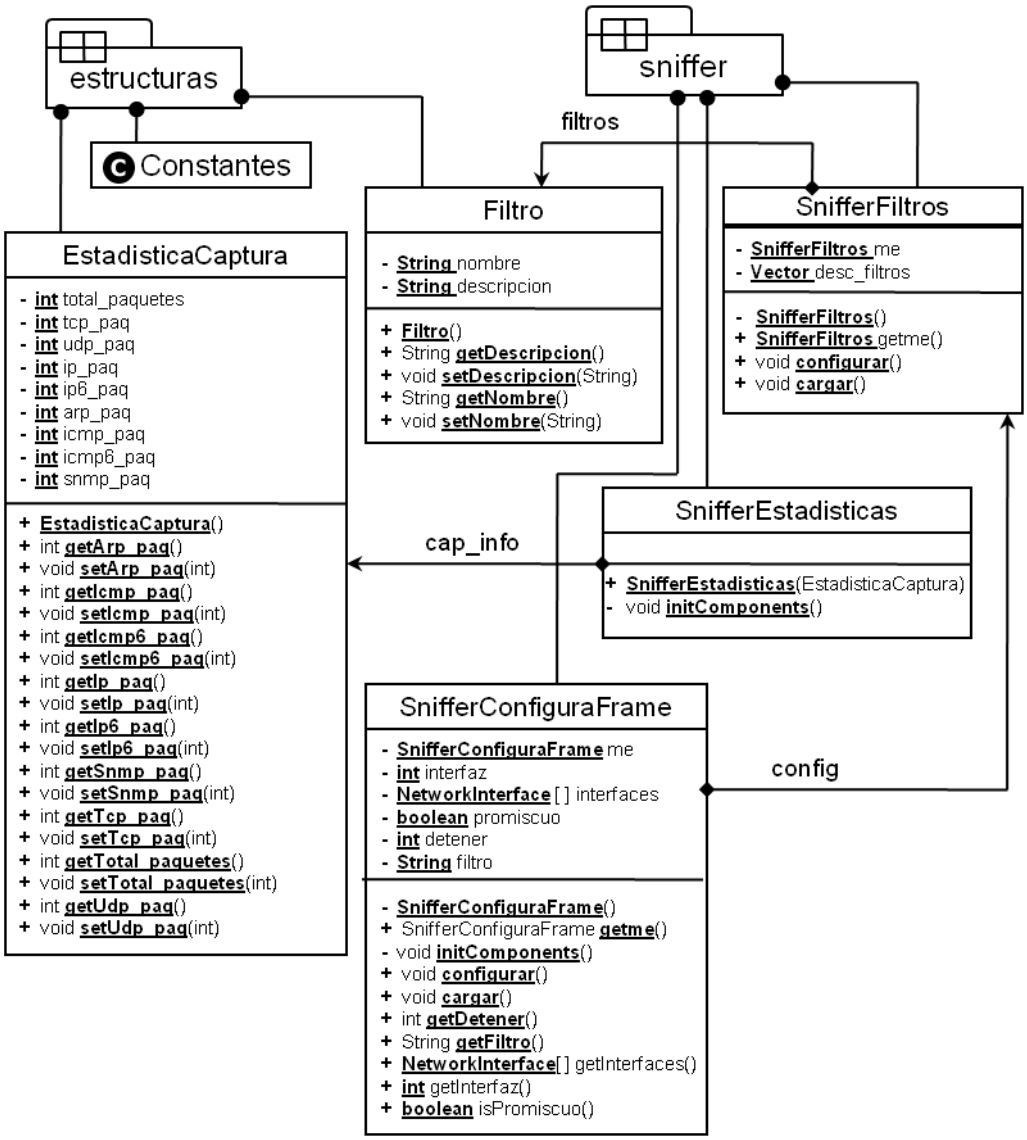


Figura 4.25: Diagrama de clases para la configuración y estadísticas del Sniffer.

Posteriormente, se diseña el componente para la captura de datos, el cual está constituido por las siguientes clases:

- estructuras.PaqueteCapturado: define la estructura para almacenar la información de un paquete capturado.
- tools.sniffer.SnifferMotor: esta clase se ejecuta de manera asíncrona y tiene como finalidad capturar los datos según la configuración especificada. Utiliza a la estructura PaqueteCapturado para almacenar la información de los paquetes que se reciben de la red.

En la Figura 4.26 se presenta el diagrama con la definición de las clases PaqueteCapturado y SnifferMotor.

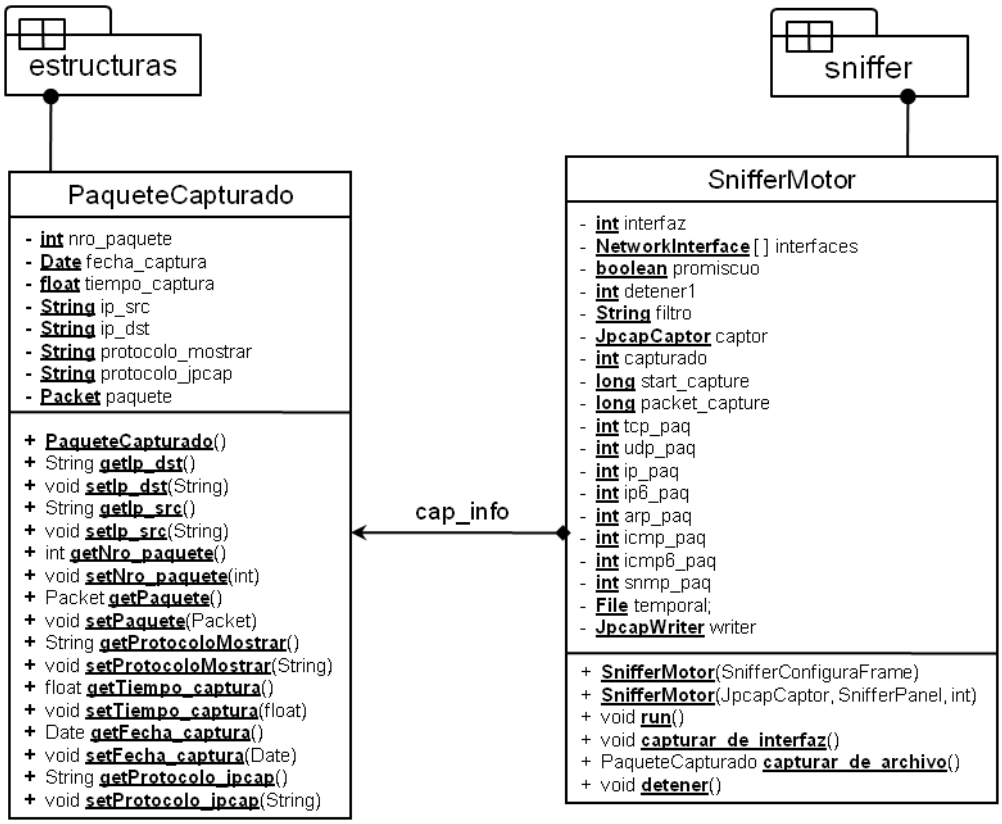


Figura 4.26: Diagrama de clases componente de captura del Sniffer.

Para el diseño del componente de análisis de datos se crean las clases AnalizarPaquetesCapturados y PaqueteICMPv6, el diseño de las mismas se pueden apreciar en la Figura 4.27.

- operaciones.AnalizarPaquetesCapturados: define una serie de métodos para el análisis de los paquetes pertenecientes a los distintos protocolos soportados por la aplicación.

- operaciones.PaqueteICMPv6: debido a que el protocolo ICMPv6 no es soportado por el paquete Jpcap, se diseñó una clase que provea todas las operaciones necesarias para el análisis del mismo.

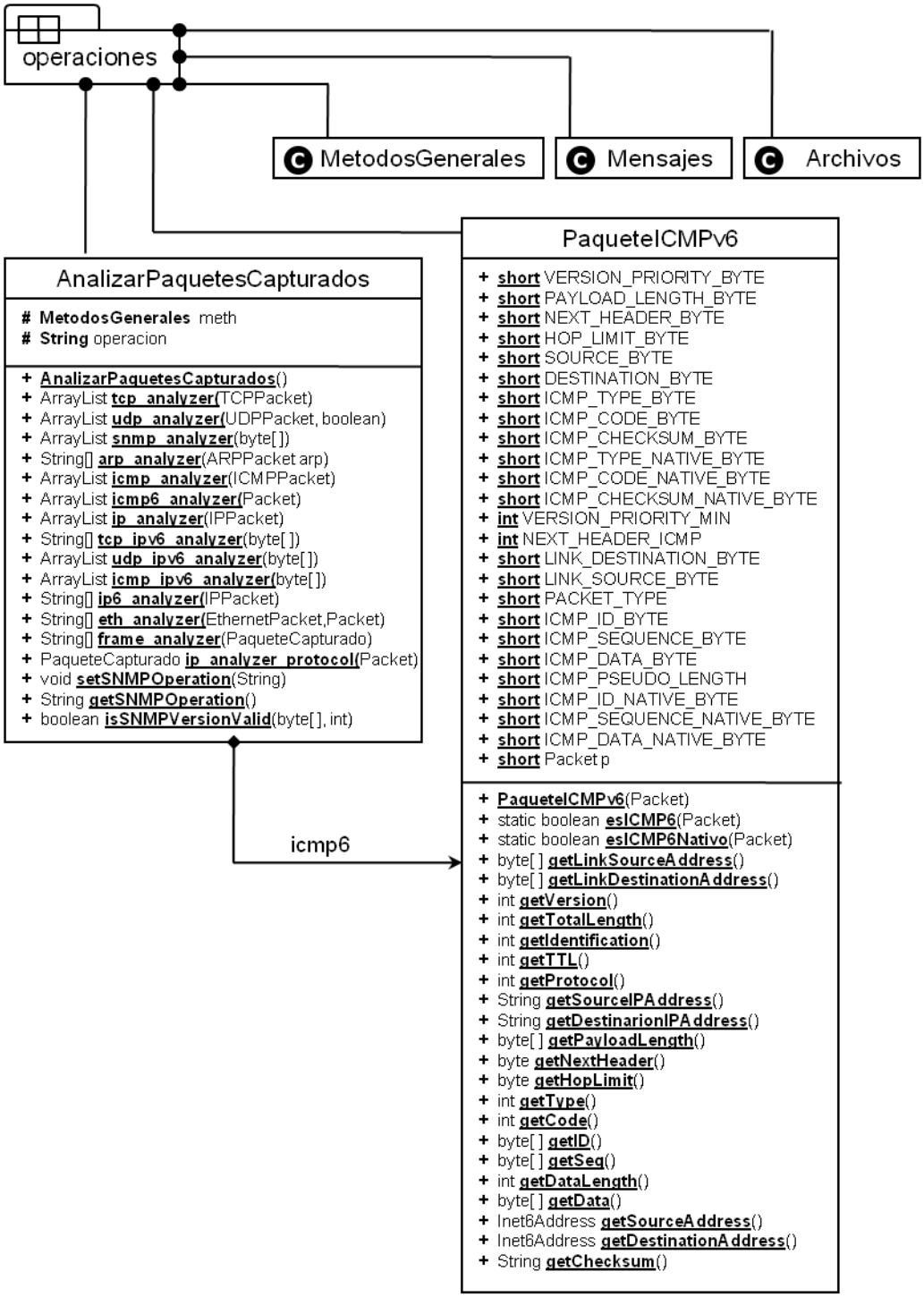


Figura 4.27: Diagrama de clases componente de análisis del Sniffer.

Finalmente, en la Figura 4.28 se muestra el diagrama de clases donde se aprecian las relaciones de las clases que componen al módulo sniffer con las clases utilitarias y se detalla el diseño de las siguientes clases:

- tools.sniffer.SnifferPanel: es la clase principal del módulo y presenta la interfaz gráfica para interactuar con el usuario.
- tools.sniffer.SnifferCapturados: esta clase provee información en tiempo real de los paquetes capturados.

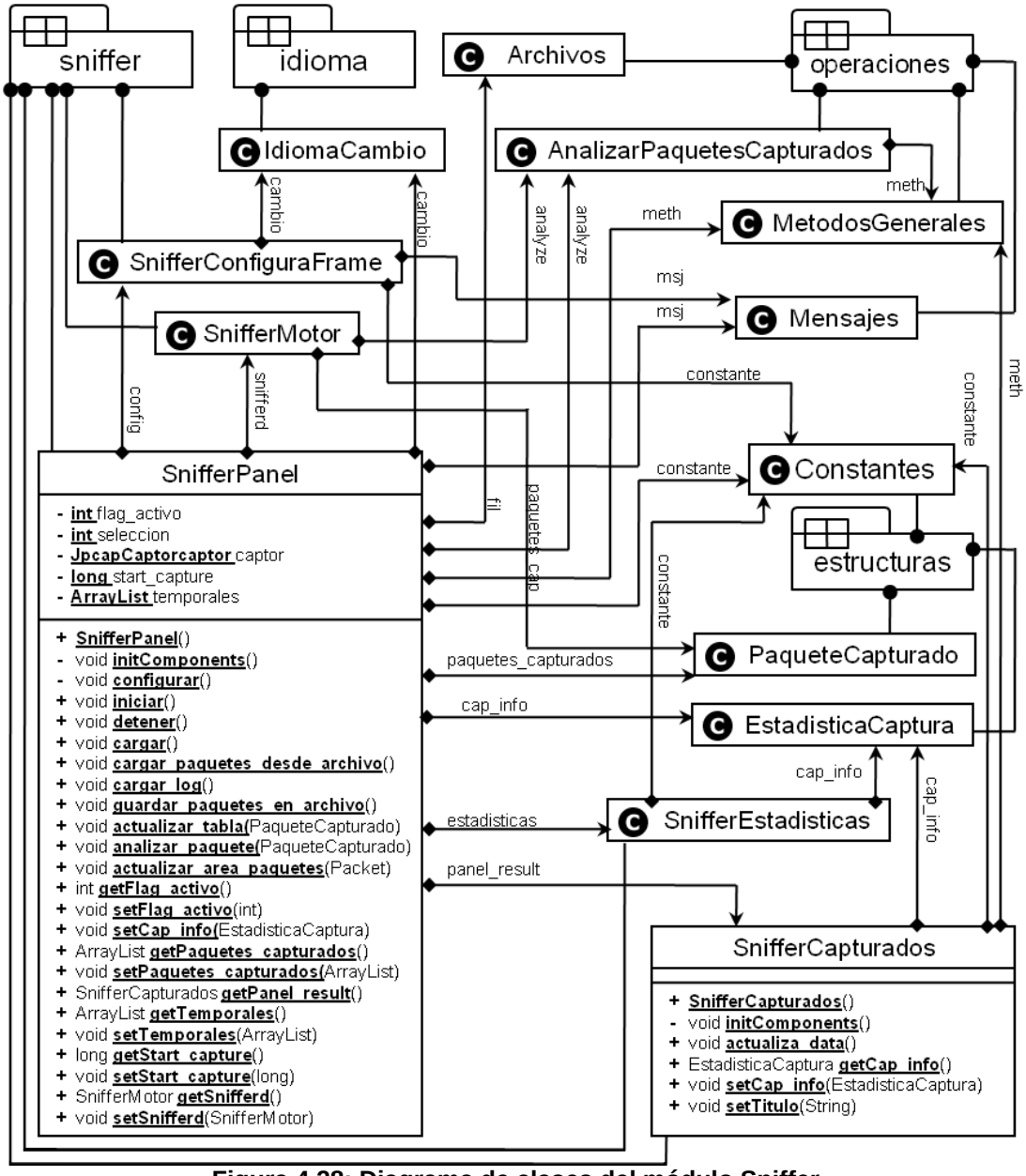


Figura 4.28: Diagrama de clases del módulo Sniffer.

4.2.9 Iteración 9: Tracert

Debido a que el tracert se fundamenta en el envío de mensajes ICMP para determinar la ruta por la cual transita un paquete, utiliza las clases IPResuelta y Direccion definidas al momento de la creación de la herramienta Ping.

Este módulo utiliza además, las clases Constantes, Mensajes e IdiomaCambio definidas previamente. La Figura 4.29 muestra la interacción entre las diferentes clases que se relacionan con este módulo. A continuación, se presenta una breve descripción de las clases TracertPanel y TracerMotor que conforman al módulo tracert:

- tools.tracert.TracertPanel: es la clase principal del módulo tracert y define la interfaz gráfica que le permite al usuario especificar las opciones bajo las cuales se ejecuta la herramienta.
- tools.tracert.TracerMotor: se ejecuta de manera asíncrona y realiza el envío de los distintos mensajes ICMP para determinar la ruta al destino indicado.

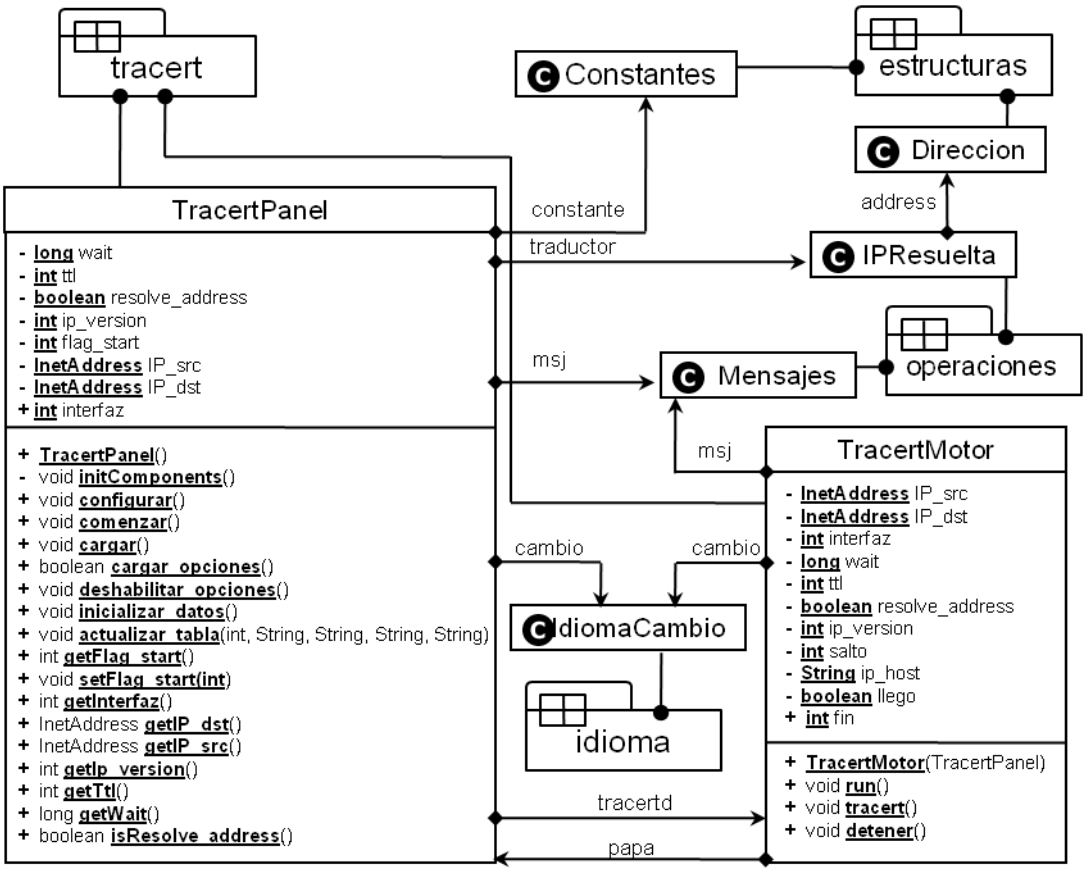


Figura 4.29: Diagrama de clases módulo Tracert.

4.2.10 Iteración 10: Receptor de Traps

Para el diseñó de la funcionalidad Receptor de Traps se utilizan las clases Validaciones, Mensajes, MetodosGenerales, Constantes e IdomaCambio como clases utilitarias.

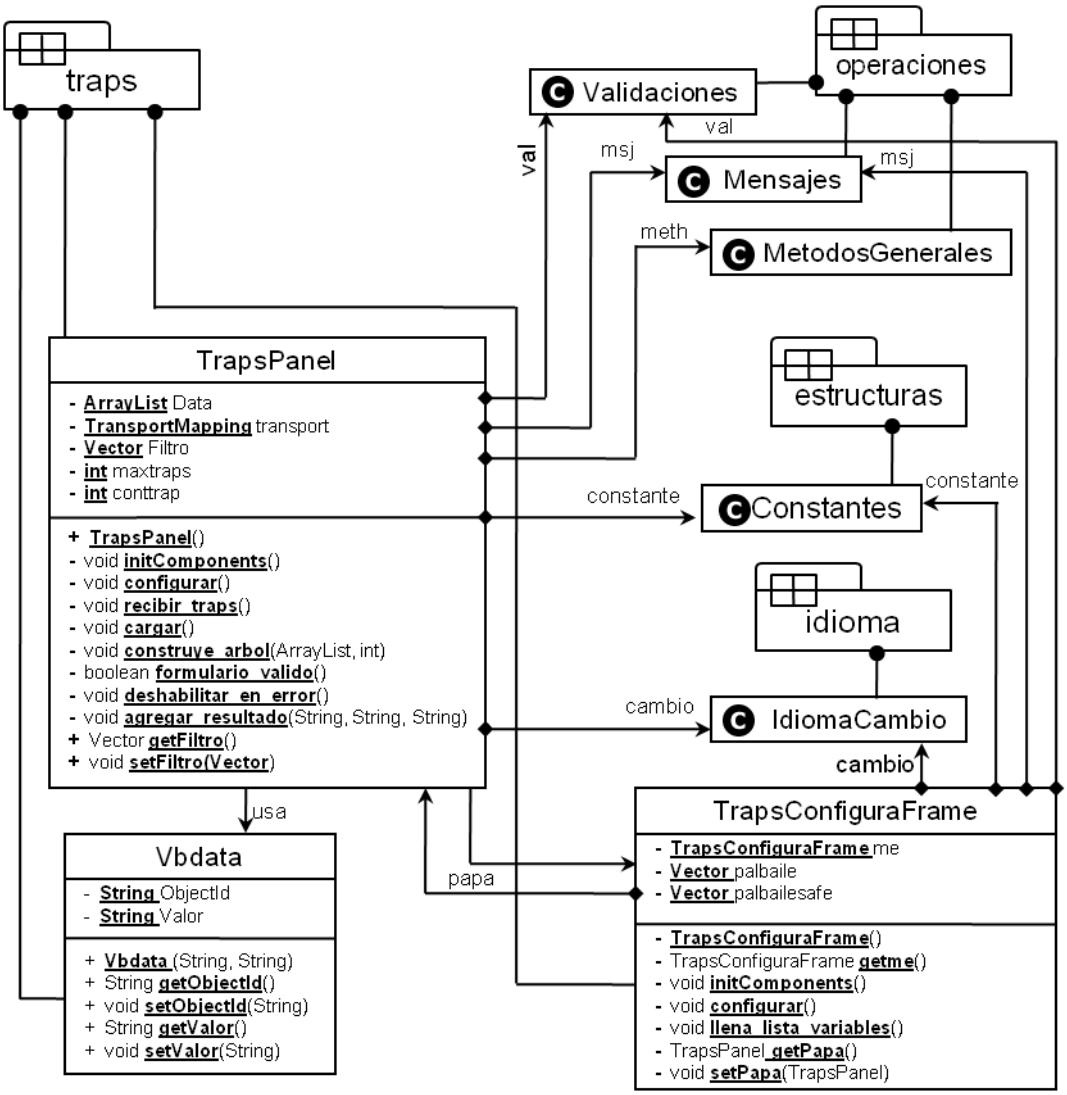


Figura 4.30: Diagrama de clases del módulo Receptor de Traps.

En la Figura 4.30 de puede apreciar el diagrama de clases del módulo Receptor de Traps y adicionalmente se detalla la estructura de las clases creadas durante el diseño de este módulo, las cuales son descritas a continuación:

- snmp.traps.TrapsPanel: clase principal de este módulo la cual permite mostrar los detalles de los traps recibidos de manera gráfica al usuario.

- snmp.traps.TrapsConfiguraFrame: permite al usuario definir las direcciones IP a filtrar durante la recepción de traps.
- snmp.traps.Vb: define una estructura que permite almacenar la información de los traps recibidos.

4.2.11 Iteración 11: Escáner de Puertos

Este módulo utiliza las siguientes clases utilitarias: IdiomaCambio, Constantes, MetodosGenerales, Mensajes y Validaciones, todas ellas descritas en iteraciones anteriores. Adicionalmente, dentro del paquete escaner se definió una clase que representa la clase principal del módulo, la cual se describe a continuación:

- tools.escaner.EscanerPanel: aparte de ser la clase principal del módulo define la interfaz gráfica tanto para la especificación de opciones como para mostrar los resultados.

En la Figura 4.31 se puede apreciar la estructura de la clase EscanerPanel y la relación que la misma tiene con las clases utilitarias.

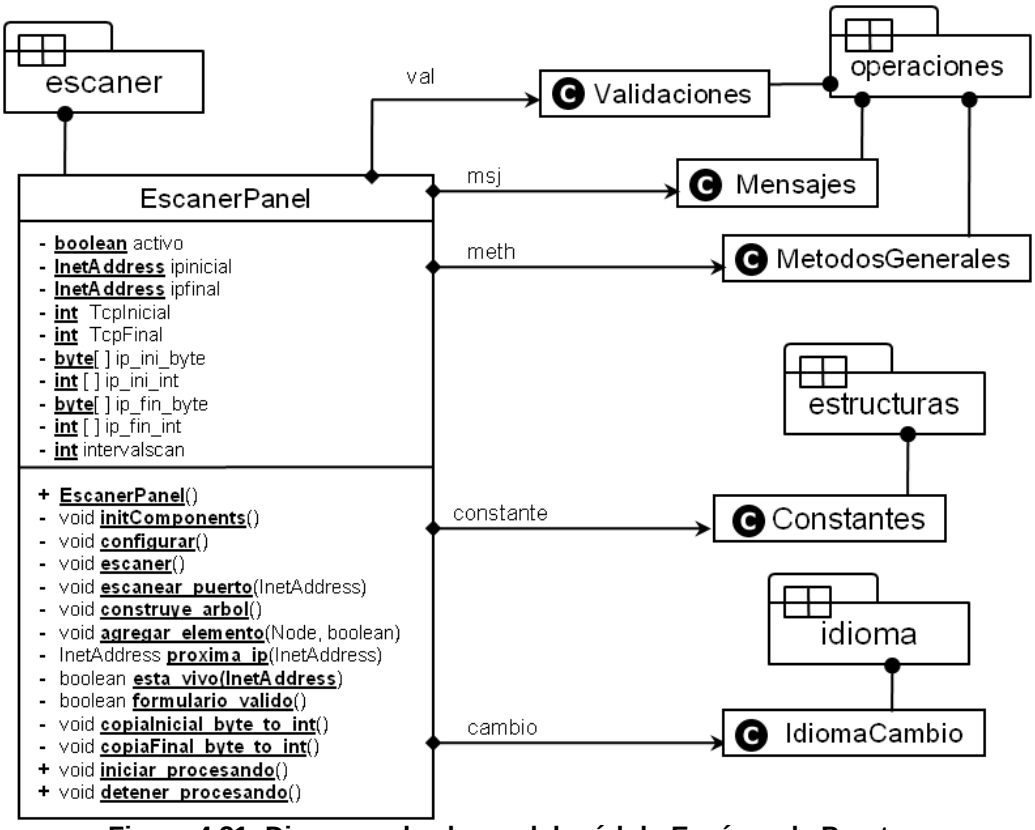


Figura 4.31: Diagrama de clases del módulo Escáner de Puertos.

4.3 Fase de Codificación

Durante la fase de codificación de cada iteración, se implementan las diferentes clases y métodos definidos en la fase de Diseño. Por lo tanto, en cada una de las iteraciones se especifican los aspectos más resaltantes durante la implementación, relativos a cada uno de los módulos de la aplicación.

Es importante destacar, que algunos de los métodos diseñados en el diagrama de clases fueron refactorizados para mejorar su funcionalidad y adaptarse a las necesidades encontradas al momento de la implementación.

4.3.1 Iteración 1: General

Dentro de la iteración 1 de la fase de codificación, se contempló la creación de la ventana principal y la presentación de la aplicación. La interfaz diseñada a pesar de estar basada en el prototipo general de interfaz definido dentro del Marco Metodológico, sufrió continuos cambios hasta obtener la versión final para la aplicación.

```
//  Panel principal
    p = new PanelPrincipal (this);

// Panel SNMP
    snmp_panel      = new ManagerPanel ();
    snmp_panel.setPapa(this);

// Grapher
    grap_panel      = new GrapherPanel ();

// Tabla
    tabla_panel     = new TablaPanel();

// Traps
    trap_panel      = new TrapsPanel();

// Panel Ping
    ping_jpanel     = new PingPanel();

// Panel Tracert
    trace_panel     = new TracertPanel();

// Panel Port Scanner
    port_panel      = new EscanerPanel();

// Panel Sniffer
    sniffer_jpanel  = new SnifferPanel();

tabbedPane.addTab(idioma.getString("principal_tab_menu"),   new ImageIcon(constante.icono_menu),p);
tabbedPane.addTab(idioma.getString("principal_tab_manager"),new ImageIcon(constante.icono_manager),snmp_panel);
tabbedPane.addTab(idioma.getString("principal_tab_tabla"),  new ImageIcon(constante.icono_tabla),tabla_panel);
tabbedPane.addTab(idioma.getString("principal_tab_grapher"),new ImageIcon(constante.icono_grapher),grap_panel);
tabbedPane.addTab(idioma.getString("principal_tab_traps"),  new ImageIcon(constante.icono_traps),trap_panel);
tabbedPane.addTab(idioma.getString("principal_tab_ping"),   new ImageIcon(constante.icono_ping),ping_jpanel);
tabbedPane.addTab(idioma.getString("principal_tab_tracer"), new ImageIcon(constante.icono_tracert),trace_panel);
tabbedPane.addTab(idioma.getString("principal_tab_sniffer"),new ImageIcon(constante.icono_sniffer),sniffer_jpanel);
tabbedPane.addTab(idioma.getString("principal_tab_scaner"), new ImageIcon(constante.icono_portscaner),port_panel);
```

Figura 4.32: Segmento de código para definir pestañas de acceso a los módulos.

Tal y como se planteó en el prototipo de interfaz y en la fase de análisis, cada uno de los módulos deben poder ser accedidos desde el menú de la aplicación. Adicionalmente, se define una segunda modalidad de acceso a

los módulos, a través de pestañas. El código Java asociado a la definición de cada una de las pestañas de acceso a los módulos se puede apreciar en la Figura 4.32.

4.3.2 Iteración 2: Manager SNMP

Para la creación del manager SNMP se definió una interfaz gráfica que permitiera al usuario introducir todos los campos necesarios para establecer la comunicación. Adicionalmente, se desarrollaron métodos para validar los valores de la conexión y con estos valores crear las estructuras apropiadas para el uso del paquete SNMP4j, el cual implementa las primitivas y operaciones básicas del protocolo SNMP.

```
private void snmpwalk(CommunityTarget target , Snmp snmp){
    try{
      Vector < VariableBinding> resultado = null; // Para la lista de resultados
      ResponseEvent response              = null; // Para la respuesta
      PDU responsePDU                     = null; // Para el PDU de los datos de la respuesta
      String oid_act = OID;
      while(oid_act.contains(OID) && isactivo == 2){
          // Valores del PDU
              PDU pdu     = new PDU(); // Para los datos de la PDU de SNMP
              pdu.add(new VariableBinding(new OID(oid_act)));
              pdu.setType(PDU.GETNEXT);
          // Se envían los datos y se espera respuesta
              response        = snmp.send(pdu, target);
              responsePDU     = response.getResponse();
              oid_act ="..";

          // Se analiza el resultado
              if (responsePDU != null) {
                  resultado  = responsePDU.getVariableBindings();
                  oid_act     = resultado.get(0).getOid().toString();
                   if(responsePDU.getErrorStatus() == 0 ){
                       for(int i = 0; i < resultado.size(); i++){
                           if(oid_act.contains(OID))agregar_resultado(resultado.get(i).getOid()+"",
                                   meth.valor_ascii(resultado.get(i)) );
                       }// Fin para
                   }else{
                       int variable_mala = responsePDU.getErrorIndex() - 1;
                       agregar_resultado("Error = " + resultado.get(variable_mala).getOid(),
                               responsePDU.getErrorStatusText() );
                       msj.mensaje_error(-1, "Error " + responsePDU.getErrorStatusText() + ".\n", null);
                   }
                  scrollToEnd();
               }else{
                   msj.mensaje_error(-1, idioma.getString("manager_error_timeout1"), null);
                  }// Fin si
      }//Fin while
      isactivo = 0 ;
    } catch (Exception e1) {
      msj.mensaje_error(-1, idioma.getString("manager_error_nodecode"), null);
          e1.printStackTrace();
    }
}//Fin  snmpwalk
```

Figura 4.33: Función snmpwalk perteneciente a la clase ManagerPanel.

Los valores retornados por los métodos del paquete SNMP4j se muestran de manera gráfica dentro de una tabla. Además, se implementan operaciones avanzadas para SNMP; es decir, operaciones que utilizan las primitivas y otras operaciones básicas para mostrar resultados más avanzados. En la

Figura 4.33 se puede apreciar la función snmpwalk, la cual es una operación avanzada SNMP que utiliza la primitiva GetNextRequest para recorrer un subárbol y mostrar su contenido.

4.3.3 Iteración 3: Ping

Para la creación de este módulo se comenzó por desarrollar diversas herramientas en el leguaje C++ que permitieran realizar el envío y recepción de mensajes de eco ICMP. Posteriormente, se implementaron los métodos definidos para la clase PingPanel y se desarrolló la interfaz gráfica que le permitiera al usuario indicar parámetros adicionales que definan las características de los mensajes ICMP enviados.

Ya que este módulo utiliza las herramientas creadas en C++ para el envío y recepción de solicitudes de eco, se define también la clase PingMotor, la cual sirve como interfaz de comunicación asíncrona entre los métodos de la aplicación creados en Java y los ejecutables generados desde el lenguaje de programación C++.

```
/**
 * Realiza la llamada al ucv_ping.exe adecuado según la versión IP
 * Luego espera los resultados y actualiza las variables según corresponda
 */
public void ping(){
      try{
          //Se crean los valores para la llamada
              String llamada = "././lib/ucv_ping"+ip_version+
                               " -f " + IP_dst.getHostAddress() +
                               " -o " + IP_src.getHostAddress() +
                               " -b " + ttl +
                               " -c " + "1" +
                               " -l " + bufersize +
                               " -p " + data +
                               " -s " + secuencia +
                               " -v " + identificador +
                               " -w " + wait;


              if(ip_version == 6){
                  if(!IP_dst.isLoopbackAddress())
                      llamada += " -x " + interfaz;
                  if(IP_dst.getHostAddress().contains("%")){
                      String[] dir = IP_dst.getHostAddress().split("%");
                      llamada += " -i " + dir[1];
                  }
              }

              //System.out.println(llamada);

          //Se crean el proceso
              Runtime     rt  = Runtime.getRuntime();
              Process     p   = rt.exec(llamada); // Se ejecuta la llamada al .exe
```

Figura 4.34: Segmento de código perteneciente a la función ping, clase PingMotor.

Durante la implementación de la clase PingMotor perteneciente al paquete ping, se realiza una llamada al sistema para ejecutar ucv_ping_4.exe o

ucv_ping_6.exe, de acuerdo a la versión del protocolo IP con la que se esté trabajando. Dicha llamada es realizada dentro del método ping de la clase PingMotor, el cual se ejecuta por cada petición ICMP (ICMPv4 o ICMPv6) realizada.

En la Figura 4.34 se puede observar parte del código definido en la función ping, en la cual se inicializan los parámetros y se realiza la llamada para ejecutar la herramienta correspondiente.

4.3.4 Iteración 4: Grapher SNMP

Durante la fase de codificación se muestra un resumen de cómo se realiza la implementación del Grapher SNMP para monitorizar y graficar el comportamiento de las variables de un dispositivo. Para esto, se debe tomar en cuenta el tipo de variable o elemento que se esté graficando, por lo que a continuación se describen los aspectos más relevantes de este proceso.

El Grapher SNMP permite monitorizar y graficar el comportamiento de dos tipos de elementos: interfaces de red o variables de la MIB. En el primer caso, se muestra al usuario la velocidad de transmisión de la interfaz escogida en bits por segundo. Para el caso donde se grafican variables de la MIB, se muestra la cantidad de elementos en un intervalo de tiempo o la variación de las mismas.

Cuando la creación de las gráficas consiste en mostrar la velocidad de transmisión, por cada interfaz de red del dispositivo se monitoriza la cantidad de octetos enviados y recibidos en un intervalo de tiempo. Luego de esto, se determina el número de bits enviados y recibidos por segundos y se grafica este valor.

En caso de que los resultados se desprendan del comportamiento de una variable de la MIB, existen dos formas para la creación de los gráficos. La primera de éstas consiste en monitorizar la variable y graficar el valor de la misma. Esta forma se utiliza cuando la variable es de tipo INTEGER, Integer32, Gauge o Gauge32.

Existe una segunda forma para graficar variables de la MIB, la cual es utilizada cuando se analizan variables de tipo Counter, Counter32 o Counter64. Este método consiste en calcular la variación en función a la cantidad de elementos muestreados en un período de tiempo, para luego expresar los resultados en elementos por segundos.

En la Figura 4.35 se puede apreciar un segmento del código de la función lee_archivos perteneciente a la clase GrapherPantallaPanel, la cual refleja cómo se leen de un archivo los valores monitorizados correspondientes al

número de octetos enviados y recibidos por una interfaz. Según el período de tiempo entre cada par de muestras se calcula la velocidad de transmisión en bits por segundo.

```
for ( int j = 4 ; j < lectura.size() ; j++) {
// Se separa la línea actual y se leen los 3 valores correspondientes
//a tiempo, Inpaktes y OutPaketes
    String data [] = lectura.get(j).split("\t");
    time_act    = Double.parseDouble(data[0]);
    in_act      = Double.parseDouble(data[1]) * 8;
    // Octetos recibidos, al multiplicarlo por 8 de da en bits
    out_act     = Double.parseDouble(data[2]) * 8 ;
    // Octetos enviados, al multiplicarlo por 8 de da en bits

    if (lastime == -1)  lastime = time_act; // Valor del tiempo ultimo muestreo

// Se calculan los deltas en los valores actuales y los nuevos valores
    diff_in = in_act - in_old  > 0 ? in_act - in_old : 0 ; // Número de bits de entrada
    in_old  = in_act; // Se actualizan los datos

    diff_out = out_act - out_old > 0 ? out_act - out_old : 0 ;// Número de bits de salida
    out_old = out_act; // Se actualizan los datos

    if (Math.abs(time_act-lastime) != 0) diff_in =  diff_in/(time_act-lastime) ;
        //  bits/Tiempo (segundos) /  --> bps
    if (Math.abs(time_act-lastime) != 0) diff_out = diff_out/(time_act-lastime);
        //  bits/Tiempo (segundos)  --> bps

    lastime = time_act;

    // Si el valor está dentro del rango de graficación se incluye en
    // la lista de los datos a llenar la lista
    if(time_limite <= time_act){
        if(cont > 1){ // Si no es la primera vez entonces existe un delta
            diff_in  = diff_in  <0 ? 0 :diff_in ;
            diff_out = diff_out <0 ? 0 :diff_out ;
            lista_entrada.add(diff_in);
            lista_salida.add(diff_out);
            tiempos.add(ultimo_time-time_act);// Diferencia en segundos desde el último
            //al actual. El último quedará en cero
        }//Fin si
        cont = 2 ;
    }//Fin si condicional tiempo afuera
}//Fin para
```

Figura 4.35: Segmento de la función lee_archivos, clase GrapherPantallaPanel.

4.3.5 Iteración 5: MIB Parser

Durante esta iteración se implementa el módulo para compilar MIBs, el cual como se describió durante la fase de análisis permite cargar nuevas MIBs a la aplicación y eliminar aquellas que hayan sido previamente cargadas.

Todas las operaciones descritas anteriormente fueron implementadas dentro de la clase MIBParserPanel que constituye la clase principal de dicho módulo. La Figura 4.36 muestra el código asociado a la función llenar_desde_carpeta, mediante la cual se obtienen la listas de archivos con definiciones de MIB cargados en la aplicación y se inicia el proceso de

compilación invocando al constructor de la clase Loader, a la cual se le pasa por parámetro una lista de archivos con definiciones de MIBs a compilar.

```
private void llenar_desde_carpeta(){
    try {
          File[] ListaDeMibs = cargadosDir.listFiles();
          // Se busca la lista de todos los archivos presentes en cargados
          if (ListaDeMibs.length > 0){
                if(hno_grapher!=null)hno_grapher.iniciar_procesando();
                if(hno_manager!=null)hno_manager.iniciar_procesando();
              Loader loader = new Loader(ListaDeMibs);
              // Se construye el loader con la lista de archivos a parsear
              loader.start();
          }//Fin si
    } catch (Exception e) {
        e.printStackTrace();
        msj.mensaje_error(-1, idioma.getString("parser_error_almacenadas"),null);
        if(hno_manager!=null)hno_manager.detener_procesando();
        if(hno_grapher!=null)hno_grapher.detener_procesando();
    }
}//Fin llenar_desde_carpeta
```

Figura 4.36: Función llenar_desde_carpeta perteneciente a la clase MIBParserPanel.

Esta funcionalidad puede ser utilizada tanto desde el Manager SNMP como desde la herramienta de monitorización (Grapher SNMP) de la aplicación, por lo que la Figura 4.37 muestra el segmento de código necesario para agregar este módulo a otro que lo necesite. Dicho código, define una instancia de la clase MIBParserPanel, la cual en su constructor recibe el controlador de eventos asociado al árbol, una referencia a la clase Constantes, las medidas que debe tener el panel y la referencia al módulo que lo va a instanciar (Manager o Grapher SNMP).

```
mib_par = new MIBParserPanel (Funcion_seleccion_tree, constante , 310 , 440, me,null);
mib_par.getpanel_tree().setBounds(10, 0, 310, 440);
add(mib_par.getpanel_tree());
```

Figura 4.37: Instanciar MIBParserPanel desde otro módulo.

4.3.6 Iteración 6: Idioma

El objetivo de la implementación del módulo de idioma es permitir que la aplicación sea multilenguaje. Por tanto, el idioma de todas las funcionalidades de AdminUCV NGN pueden ser vistas tanto en inglés como en español.

Durante la fase de análisis de la iteración 6, se describió que el proceso asociado al cambio de idioma comienza al momento de que el usuario selecciona el idioma de su preferencia a través de la interfaz mostrada por la clase IdiomaPanel. Luego que se pulsa el botón Aceptar, se actualizará el archivo de configuración de idioma de la aplicación, denominado

admin_ngn.cfg y a través del cual se carga el idioma configurado por el usuario cada vez que se reinicia la aplicación.

En la Figura 4.38 se puede apreciar el constructor de la clase IdiomaCambio, que describe la implementación realizada para conocer el idioma configurado para la aplicación y la selección del archivo *.properties* adecuado que contiene la definición de las variables en el idioma configurado.

```
private IdiomaCambio() {
    arch        = Archivos.getme();                          // Se inicializa el objeto Archivos
    ArrayList<String> idioma = new ArrayList<String>();     // Se inicializa el ArrayList

    try {
        idioma = arch.leer_archivo_arraylist(ruta_idiomas+"admin_ngn.cfg", true);
        // Se abre el archivo con la configuración del idioma
        language = idioma.get(0);           // Se lee el string que indica el idioma configurado
        Locale locale = null;               // Se inicializa un objeto de tipo Locale según el idioma configurado,
        //para posteriormente saber qué archivos .properties se va a abrir
        if(language.equalsIgnoreCase("espanol"))    locale = new Locale("");
        else if(language.equalsIgnoreCase("ingles"))locale = new Locale("en","US");

        // Se inicializa el objeto ResourceBundle con la ubicación del archivo .properties (paquete idioma)
        // y ubicación del mismo (idioma y país)
        etiqueta = ResourceBundle.getBundle("idioma.idioma",locale);
    } catch (IOException e) {
        e.printStackTrace();
    }

}//Fin constructor
```

Figura 4.38: Constructor de la clase IdiomaCambio.

Adicionalmente, en la Figura 4.39, se muestra la estructura de los archivos *.properties*, los cuales como se mencionó anteriormente, contienen la definición de todas las variables de la aplicación que requieran cambio de idioma. El formato de estos archivos consiste en indicar el nombre de la variable (debe ser igual en ambos idiomas), seguido del carácter '=' y el texto en el idioma correspondiente.

```
principal_menu_menu         = Menú
principal_menu_inicio       = Inicio
principal_menu_salir        = Salir
principal_menu_ayuda        = Ayuda
principal_menu_idioma       = Idioma
principal_menu_creditos     = Desarrolladores
principal_menu_acerca       = Acerca de...
principal_tab_menu          = Menú
principal_tab_manager       = Manager SNMP
principal_tab_grapher       = Grapher SNMP
principal_tab_tabla         = Tabla Genérica SNMP
principal_tab_traps         = Receptor de traps
principal_tab_ping          = Ping
principal_tab_tracer        = Tracert
principal_tab_sniffer       = Sniffer
principal_tab_scaner        = Escáner de puertos
principal_title_menu1       = Administración SNMP
principal_title_menu2       = Otras Herramientas
```

```
principal_menu_menu         = Menu
principal_menu_inicio       = Home
principal_menu_salir        = Exit
principal_menu_ayuda        = Help
principal_menu_idioma       = Language
principal_menu_creditos     = Credits
principal_menu_acerca       = About...
principal_tab_menu          = Home
principal_tab_manager       = SNMP Manager
principal_tab_grapher       = SNMP Grapher
principal_tab_tabla         = SNMP Generic Table
principal_tab_traps         = Traps Receiver
principal_tab_ping          = Ping
principal_tab_tracer        = Tracert
principal_tab_sniffer       = Sniffer
principal_tab_scaner        = Port Scanner
principal_title_menu1       = SNMP Administration
principal_title_menu2       = Other Administration Tools
```

Figura 4.39: Estructura de archivos .properties para definir idioma de las variables.

4.3.7 Iteración 7: Tabla Genérica

Este módulo define una de las operaciones avanzadas SNMP, la cual consiste en ejecutar GetNextRequest sucesivos para consultar el conjunto de valores que componen una tabla SNMP.

Por tanto, esta herramienta recibirá el OID de la tabla que se desea consultar. Los resultados son mostrados en una tabla ordenados de forma columnar, donde cada columna representa un OID de la misma.

Esta herramienta puede ser ejecutada de 2 formas: (1) acceder directamente a la funcionalidad (a través del menú o el conjunto de pestañas) y especificar la configuración dentro de la ventana mostrada por la clase TablaConfiguraFrame o (2) seleccionar la operación Table View desde el Manager SNMP, el cual pasará los parámetros al paquete snmp.tabla para mostrar los resultados correspondientes con el OID indicado desde el manager.

4.3.8 Iteración 8: Sniffer

Debido a los diversos aspectos de funcionalidad que debe cubrir el sniffer, esta iteración se dividió en componentes o niveles que fueron descritos en la fase anterior junto a la explicación de cada una de las clases que lo componen.

Se debe destacar que la herramienta provee soporte para los protocolos: ARP, IPv4, IPv6, ICMPv4, ICMPv6, TCP, UDP y SNMP. Por cada paquete capturado perteneciente a dichos protocolos, se ofrece al usuario el análisis detallado de los campos más importantes y la especificación del paquete en bytes.

La implementación de los filtros se basa en la sintaxis de tcpdump para la creación de filtros de precaptura y se ofrece al usuario una sección donde se le permite salvar, modificar y eliminar los filtros creados.

Todas las capturas realizadas, pueden ser almacenadas y consultadas nuevamente. El formato definido para estos archivos fue la extensión *.cap* y *.pcap* que son compatibles con otros softwares reconocidos de captura, lo cual brinda libertad al usuario para visualizar las capturas realizadas de otras aplicaciones con AdminUCV NGN y viceversa.

Debido a la cantidad de memoria que llegaba a consumir la herramienta, se refactorizó el módulo para implementar paginación al momento de mostrar los resultados de la captura. En el producto final de la fase de codificación se logró que dicho módulo muestre 10.000 paquetes capturados por página, añadiendo a la interfaz los elementos necesarios para navegar entre las diferentes páginas. Adicionalmente, se estableció un límite para la captura, el cual se fijó en 500.000 paquetes.

4.3.9 Iteración 9: Tracert

El módulo tracert funciona de manera similar al ping, ya que ambos se basan en el envío de peticiones eco ICMP a un host, con la diferencia de que el tracert determina los saltos (hops) por los que transita un paquete antes de llegar al destino.

```
//Se crean el proceso
    Runtime     rt  = Runtime.getRuntime();
    Process     p   = rt.exec(llamada); // Se ejecuta la llamada al .exe
//Se obtiene la referencia a la entrada y salida del proceso
    InputStream in  =  p.getInputStream();
// Se captura el contenido de la entrada estándar
    BufferedReader reader = new BufferedReader(new InputStreamReader(in));
//Se hace un ciclo sobre la respuesta
    String data;
    while((data = reader.readLine()) != null){// Mientras que existan datos por leer

        String[] result = data.split("=");// Se comienzan a parsear los resultados
        //System.out.println("datos son: "+data);

        String[] resp = result[1].split("~");// Se parsean los datos que vienen en la respuesta

        if(result[0].equalsIgnoreCase("Llego")){// Se ha alcanzado al host destino
            llego = true;
            if(IP_dst.isLoopbackAddress()){
                if(ip_version == 4) ip_host = "127.0.0.1";
                if(ip_version == 6) ip_host = "0:0:0:0:0:0:0:1";
            }else ip_host = resp[3];
            if(resolve_address)
                papa.actualizar_tabla(salto, resp[0], resp[1], resp[2],
                        (InetAddress.getByName(ip_host)).getHostName()+" ["+ip_host+"]");
            else
                papa.actualizar_tabla(salto, resp[0], resp[1], resp[2], ip_host);
        }if(result[0].equalsIgnoreCase("LlegoErr")){// Cuando el host destino no respondio
            llego = true;
            papa.actualizar_tabla(salto, resp[0], resp[1], resp[2], idioma.getString(resp[3]));
        }else if(result[0].equalsIgnoreCase("Info")){// Respuesta de un nodo intermedio
            ip_host = resp[3];
            if(resolve_address)
                papa.actualizar_tabla(salto, resp[0], resp[1], resp[2],
                        (InetAddress.getByName(ip_host)).getHostName()+" ["+ip_host+"]");
            else
                papa.actualizar_tabla(salto, resp[0], resp[1], resp[2], ip_host);
        }else if(result[0].equalsIgnoreCase("InfoErr")){// Cuando un host intermedio no responde
            papa.actualizar_tabla(salto, resp[0], resp[1], resp[2], idioma.getString(resp[3]));
        }else if(result[0].equalsIgnoreCase("Error")){// Error durante la traza
            msj.mensaje_error(-1, idioma.getString(result[1]), null);
            detener();
        }
    }//Fin mientras
```

Figura 4.40: Segmento de código de la función tracert, clase TracertMotor.

En base al diseño realizado en la fase anterior, dicho módulo consta de 2 clases. TracertPanel define la interfaz de usuario para mostrar los resultados de la traza obtenida y TracertMotor instancia un hilo que se encarga de ejecutar de forma asíncrona una llamada a las herramientas complementarias creadas en el lenguaje C++ (tracert_ucv_4.exe y tracert_ucv_6.exe) dependiendo de la versión del protocolo IP seleccionado por el usuario.

Una vez que se ejecutan dichas herramientas complementarias, es necesario parsear los resultados obtenidos. Este proceso puede ser observado en la Figura 4.40, que muestra un segmento de código de la función tracert perteneciente a la clase TracertMotor, donde luego que se realiza la llamada al ejecutable adecuado, se obtiene los resultados y se parsean de acuerdo al tipo de respuesta obtenida.

4.3.10 Iteración 10: Receptor de Traps

La iteración 10 consistió en implementar el módulo para recepción de traps, el cual como indica su nombre permite la captura de traps o alertas enviadas por un agente SNMP. Para el uso de la herramienta se debe indicar el número de puerto configurado para la recepción de traps y adicionalmente, provee una sección de filtrado para restringir la captura a las direcciones IP de las cuales desee recibir alertas el usuario.

4.3.11 Iteración 11: Escáner de Puertos

La clase EscanerPanel diseñada durante la fase 2 compone la clase principal de este módulo, la cual es la encargada de presentar la interfaz gráfica de esta herramienta y manejar los eventos asociados a cada una de las funcionalidades ofrecidas por el módulo. Dichas funcionalidades, consisten básicamente en permitir el escaneo de un rango de puertos TCP definido por el usuario y asociados a una o a un rango de direcciones IP consecutivas. Adicionalmente, permite probar la conexión hacia un host y forzar el escaneo sin conexión.

```
private void escanear_puerto(InetAddress direccion, DefaultMutableTreeNode host){
    try {
    DefaultMutableTreeNode tcp = new DefaultMutableTreeNode("TCP");
    host.add(tcp);
    for (int i = TcpInicial; i < (TcpFinal+1) && activo; i++) {
          try {
              label_info1.setText(idioma.getString("scaner_info_testport") + i);
              Socket retomo = new Socket();
              retomo.bind(null);
              retomo.connect(new InetSocketAddress(direccion, i),intervalscan);
              tcp.add(new DefaultMutableTreeNode("Open Port: " + i));
              agregar_elemento(tcp,false);
          }catch (ConnectException e) { }
          catch (IOException e) { }
     } // fin para
    }catch (Exception e1) {
      e1.printStackTrace();
    }
}//Fin escanear_puerto
```

Figura 4.41: Función escanear_puerto perteneciente a la clase EscanerPanel.

Para cada una de las direcciones IP (IPv4 o IPv6) indicadas por el usuario, se realizará el escaneo en el rango de puertos señalado, lo cual se logra mediante la función escanear_puerto mostrada en la Figura 4.41. Esta función instancia un socket que intenta conectarse a la IP dada en cada uno de los puertos pertenecientes al rango señalado y además, agrega un nodo a la interfaz en caso de que un puerto esté activo.

4.4 Fase de Pruebas

Al finalizar cada iteración se realizan pruebas de funcionamiento, las cuales consisten en verificar que los datos arrojados por el módulo evaluado sean correctos. Por tanto, para cada iteración y luego de la implementación se lleva a cabo el proceso de validación de las distintas funcionalidades de la aplicación.

En esta subsección se agrupan las pruebas realizadas a cada iteración y se explica el detalle de las mismas junto a los resultados obtenidos.

Para esta fase se realiza una iteración adicional a las mostradas en etapas anteriores, la cual refleja el comportamiento de la aplicación luego de la unificación de los distintos módulos. Las pruebas de integración verifican que exista consistencia en la interfaz gráfica de usuario y aseguran que se mantenga el correcto funcionamiento de las herramientas evaluadas en iteraciones anteriores.

Durante el desarrollo de la fase de pruebas de cada iteración se utilizan múltiples aplicaciones con el fin de contrastar y comparar algunos de los resultados obtenidos en AdminUCV NGN:

- AdventNet Simulation Toolkit 5.0: software que permite la simulación de dispositivos con soporte para SNMP y otras funcionalidades. Fue utilizado bajo la versión de prueba de 30 días disponible en la página Web de AdventNet.
- Wireshark 1.0: herramienta de código abierto capaz de capturar y analizar el tráfico que transita por la red.
- Ireasoning MIB Browser v4.2 Personal Edition: aplicación propietaria para la administración de redes vía SNMP, que permite cargar MIBs y realizar distintas operaciones SNMP v1/2c/3.
- SolarWinds Toolset Launchpad: ofrece un conjunto de herramientas para administración de redes. Utilizado en su versión de prueba de 30 días para envío de traps y contrastar resultados obtenidos con AdminUCV NGN.

En la Figura 4.42 se muestra la topología de la red utilizada para realizar las pruebas, la cual está formada por seis hosts que tienen instalados distintos sistemas operativos.

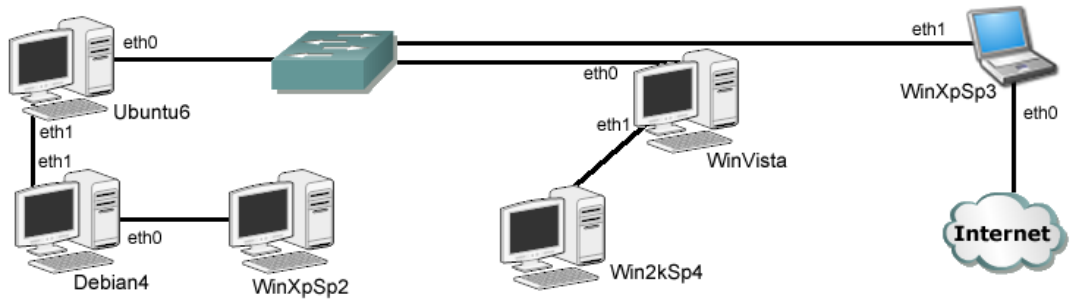


Figura 4.42: Topología de la red de pruebas.

En la Tabla 4.33 y la Tabla 4.34 se detallan las características (referentes a sistema operativo y direccionamiento) de cada uno de los hosts presentes en la red de pruebas.

	WinXpSp2	WinVista	Win2KSp4
Sistema	Windows XP SP2	Windows Vista Ultimate	Windows 2000 SP4
IPv4- eth0	10.10.20.2/24	192.168.30.139/24	172.18.0.2/16
IPv6- eth0	2AF5:F30::2/64	2001:FA::3/64	n/a
IPv4 - eth1	n/a	172.18.0.1/16	n/a
IPv6 - eth1	n/a	2001:DB8::1/64	n/a

Tabla 4.33: Características de los hosts de la red de pruebas parte 1.

	Ubuntu6	Debian4	WinXpSp3
Sistema	Ubuntu 6.10	Debian KDE 4.0	Windows XP SP3
IPv4 - eth0	192.168.30.129/24	10.10.20.1/24	201.248.23.252/19
IPv6 - eth0	2001:FA::2/64	2AF5:F30::1/64	2002:C9F8:17FC::C9F8:17FC
IPv4 - eth1	172.17.20.1/16	172.17.20.2/16	192.168.30.1/24
IPv6 - eth1	2003:F20::1/64	2003:F20::2/64	2001:FA::1/64

Tabla 4.34: Características de los hosts de la red de pruebas parte 2.

Todos los hosts excepto el WinXpSp3 están representados por máquinas virtuales, y los hosts WinVista, Ubuntu6 y Debian4 tienen habilitado el enrutamiento tanto para IPv4 como para IPv6.

4.4.1 Iteración 1: General

Las pruebas unitarias de funcionamiento realizadas para la primera iteración, consisten en verificar que cada una de las opciones del menú mostradas en el panel inicial y en la barra de herramientas corresponden con el módulo adecuado al ser pulsadas. Adicionalmente, se valida que los atajos para usuarios expertos conduzcan a la pestaña indicada.

En la Figura 4.43 se puede apreciar la ventana principal de la aplicación, junto al panel inicial y el menú de la barra de herramientas que presentan las opciones para acceder a los diferentes módulos del programa.

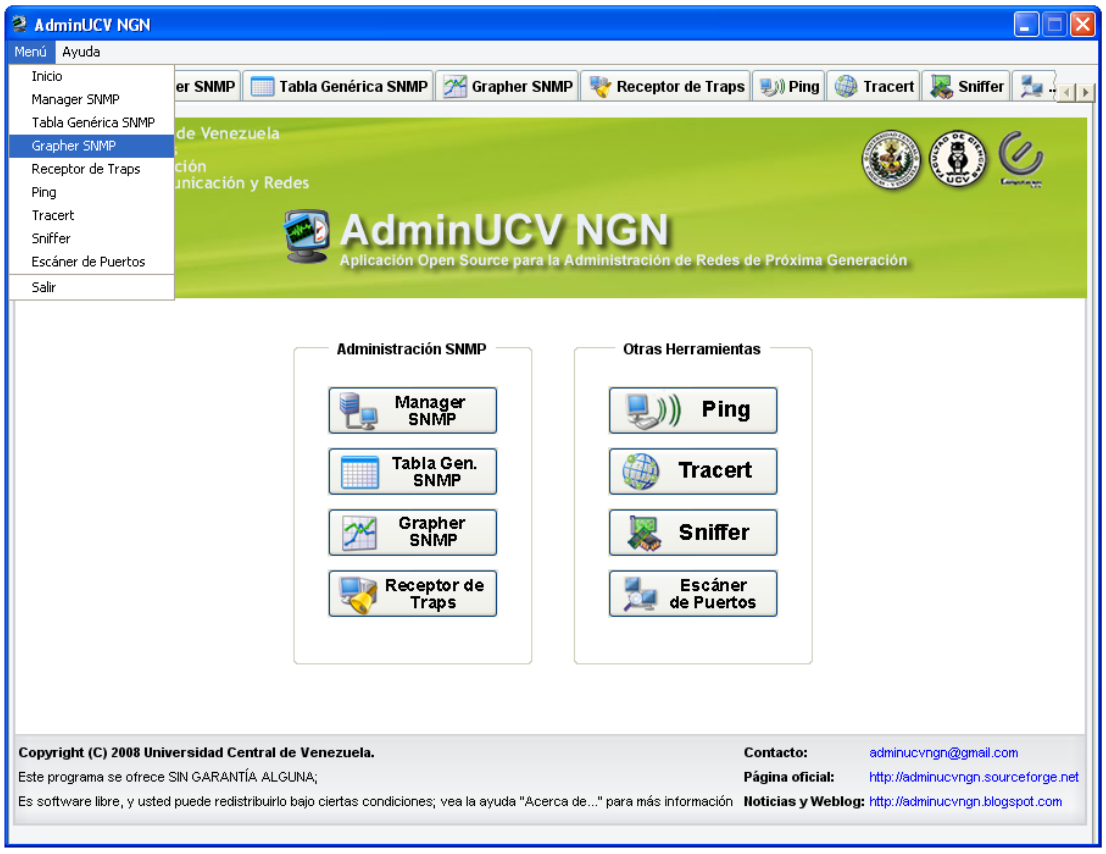


Figura 4.43: Ventana principal, panel inicial y barra de herramientas.

4.4.2 Iteración 2: Manager SNMP

Las pruebas unitarias para el manager SNMP se inician utilizando valores simulados generados mediante la aplicación AdventNet Simulation Toolkit 5.0. Dichos valores, sirvieron para verificar que mediante los datos de entrada predefinidos se obtiene la respuesta esperada en las diferentes operaciones. En cada una de las operaciones realizadas, se capturaron las tramas enviadas mediante Wireshark para validar el formato de las mismas y las respuestas obtenidas.

Al confirmar que el formato de las tramas enviadas por el manager es correcto y que los resultados son interpretados de manera adecuada, se llevan a cabo consultas SNMP a agentes pertenecientes a la red de pruebas con direcciones IPv4 e IPv6. Al igual que en la prueba descrita anteriormente, se capturan todas las tramas enviadas con Wireshark para verificar que los datos enviados y recibidos son correctos.

La Figura 4.44 muestra los resultados obtenidos por el manager luego de realizar una consulta GetRequest desde el host WinXpSp3 (2001:FA::1) hacia el host WinVista (2001:FA::3), los cuales fueron validados al realizar la captura de las tramas enviadas y recibidas con Wireshark. La Figura 4.45 permite apreciar la captura de la respuesta obtenida del host WinVista, donde se observa que los datos mostrados por el manager y la captura coinciden.

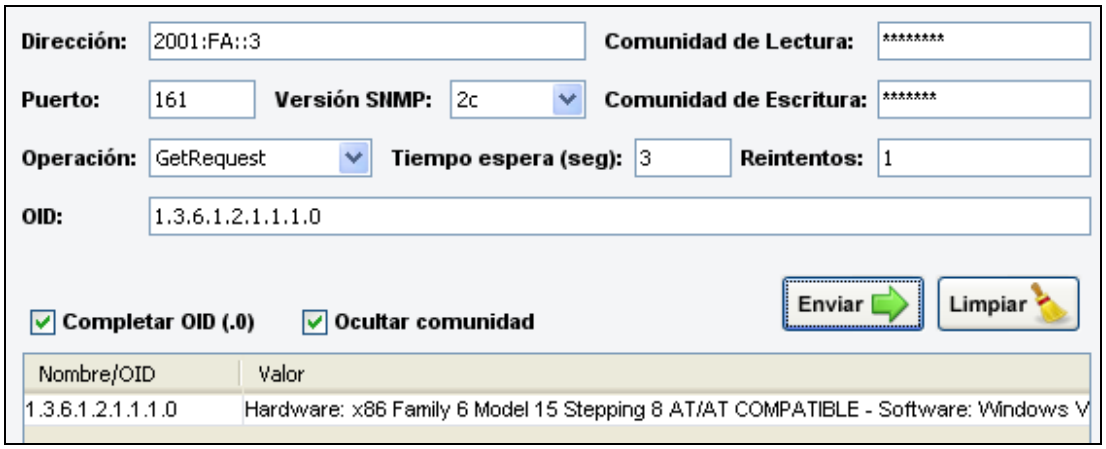


Figura 4.44: Módulo Manager SNMP luego de ejecutar la operación GetRequest.

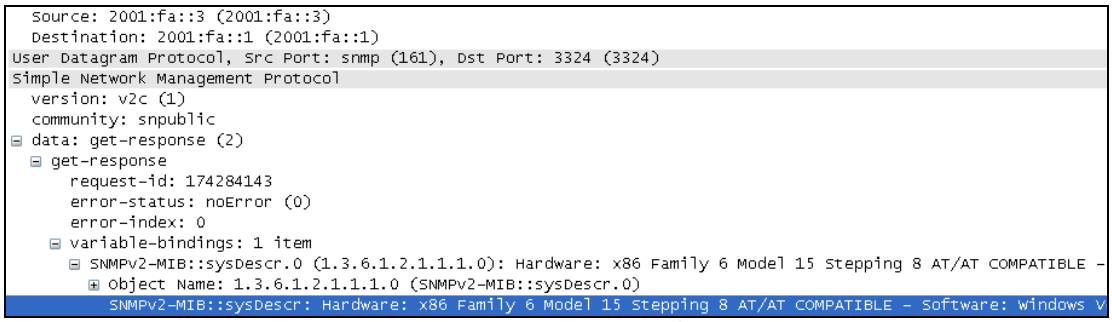


Figura 4.45: Captura de la respuesta a la operación GetRequest con Wireshark.

4.4.3 Iteración 3: Ping

Para validar el comportamiento de la herramienta ping, AdminUCV NGN fue instalado en hosts con Windows XP SP2, Windows XP SP3 y Windows Vista, haciendo ping desde cada uno de ellos hacia otros dispositivos con direcciones IPv4 y direcciones IPv6 nativas, link-local, túneles automáticos y túneles 6to4. Mediante el uso de Wireshark se realizan capturas de los mensajes enviados y se analiza la estructura de los mismos para garantizar que las opciones seleccionadas en la herramienta modifican el comportamiento de las solicitudes de eco.

Debido a que para la creación de este módulo fue necesario utilizar herramientas programadas bajo el lenguaje C++, no es posible ejecutar el

ping bajo sistemas operativos distintos a Windows ya que dichas herramientas hacen uso de algunas librerías específicas para el sistema operativo mencionado.

En la Figura 4.46 se puede observar una vista de la herramienta ping, luego de haberse ejecutado con las siguientes opciones:

- Dirección destino: www.ipv6.org.
- Versión del Protocolo de Internet (IP): IPv6.
- Identificador de la petición: 9.
- Número de solicitudes a realizar: 9.
- Carga útil: 248 bytes.
- Patrón de datos a enviar: “adminucvngn”.

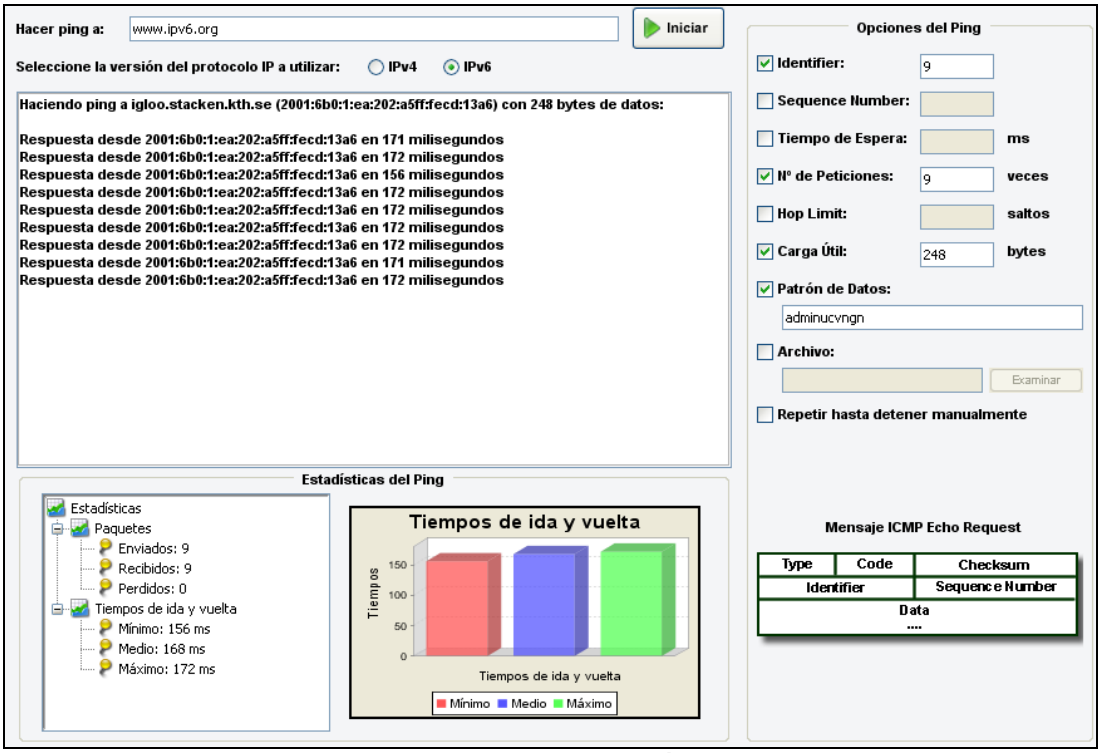


Figura 4.46: Resultados obtenidos al probar el módulo Ping hacia www.ipv6.org.

La Figura 4.47 muestra la captura realizada con Wireshark de uno de los mensajes de solicitud de eco enviados por la herramienta ping de AdminUCV NGN. En dicha figura, se aprecia que el mensaje cumple con los parámetros establecidos para la dirección destino, identificador, tamaño de la carga útil igual y el patrón datos hexadecimal 61 64 6D 69 6E 75 63 76 6E 67 6E (que se repite hasta completar los 248 bytes configurados) que corresponde con la cadena “adminucvngn”.

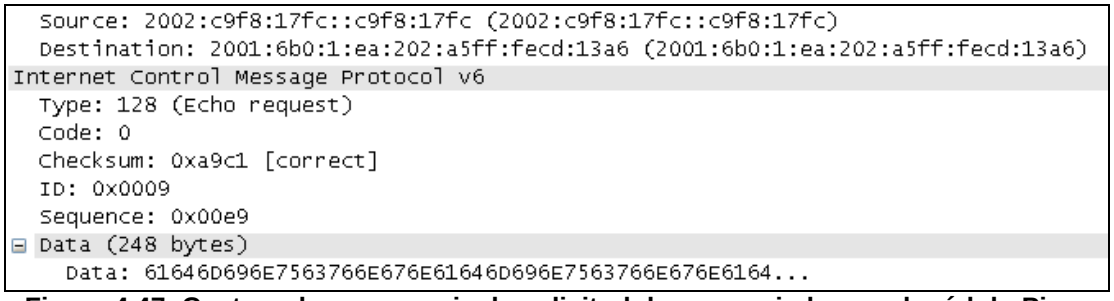


Figura 4.47: Captura de un mensaje de solicitud de eco enviado por el módulo Ping.

4.4.4 Iteración 4: Grapher SNMP

Mediante la fase de pruebas de este módulo se desea verificar que los gráficos obtenidos coincidan con los datos monitorizados. Debido a la complejidad que representa analizar un conjunto de datos provenientes de un agente real, se decide iniciar el proceso de pruebas con datos simulados. Estos son generados a partir de funciones matemáticas que permiten conocer el comportamiento de los datos con antelación y facilitan el análisis de los mismos para la creación de la gráfica y comparación de los resultados mostrados por AdminUCV NGN. Estos valores son preconfigurados a partir de un dispositivo de red simulado mediante la aplicación AdventNet Simulation Toolkit 5.0.

La primera prueba realizada a este módulo consiste en validar las gráficas mostradas al monitorizar la velocidad de transmisión de una interfaz de red, tomando una muestra por minuto durante 60 minutos. Para esto, se configuró el simulador instalado en el host WinXpSp2 de manera que la cantidad de octetos recibidos variara según la fórmula siguiente:

Número de Octetos (t) = 0,6 × exp (0,33 × t), donde t es una muestra.

En la Tabla 4.35 se presenta un estimado de la cantidad de octetos, la cantidad de octetos por minuto y bits por segundo recibidos por la interfaz en las últimas cinco muestras.

Muestra	Octetos	Octetos por minuto	Bits por segundos
55	45.771.600,31	12.865.310,53	1.715.374,74
56	63.666.837,22	17.895.236,91	2.386.031,59
57	88.558.541,42	24.891.704,19	3.318.893,89
58	123.182.108,61	34.623.567,20	4.616.475,63
59	171.342.387,08	48.160.278,47	6.421.370,46
60	238.331.799,48	66.989.412,40	8.931.921,65

Tabla 4.35: Velocidad de recepción estimada en la prueba del Grapher SNMP.

De manera similar, se configura el simulador para que el número de octetos enviados por la interfaz de red varíe según la siguiente fórmula:

Número de Octetos (t) = 54.000.000,00 + t × 100.000, donde t es una muestra.

En la Tabla 4.36 se detalla el estimado de la cantidad de octetos, octetos por minuto y bits por segundo enviados por la interfaz en las últimas cinco muestras.

Muestra	Octetos	Octetos por minuto	Bits por segundos
55	2.970.100.000,00	54.000.000,00	7.200.000,00
56	3.024.100.000,00	54.000.000,00	7.200.000,00
57	3.078.100.000,00	54.000.000,00	7.200.000,00
58	3.132.100.000,00	54.000.000,00	7.200.000,00
59	3.186.100.000,00	54.000.000,00	7.200.000,00
60	3.240.100.000,00	54.000.000,00	7.200.000,00

Tabla 4.36: Velocidad de transmisión estimada en la prueba del Grapher SNMP.

Un vez calculados los valores y estimada la gráfica se procede a monitorizar el dispositivo simulado desde el módulo Grapher SNMP y se comparan los resultados obtenidos. El gráfico generado por la herramienta se puede apreciar en la Figura 4.48, donde se observa que la misma, presenta el comportamiento esperado (gráfica similar a la función exponencial, sobre la cual se basó el cálculo).

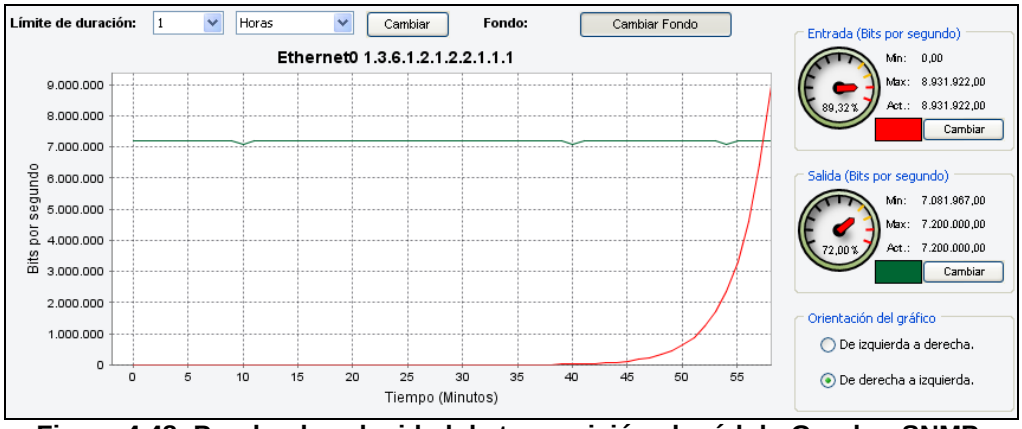


Figura 4.48: Prueba de velocidad de transmisión al módulo Grapher SNMP.

En la Figura 4.49 se puede apreciar la gráfica generada por AdminUCV NGN luego de monitorizar una variable de la MIB del dispositivo de red simulado, cuyo comportamiento varía según la siguiente fórmula para generar una onda sinusoidal:

Número de Elementos (t) = 2.147.483.640 × sin(100 × t), donde t es una muestra.

La variable que se está graficando es de tipo INTEGER, por lo que la Figura 4.49 muestra los valores monitorizados sin ninguna conversión.

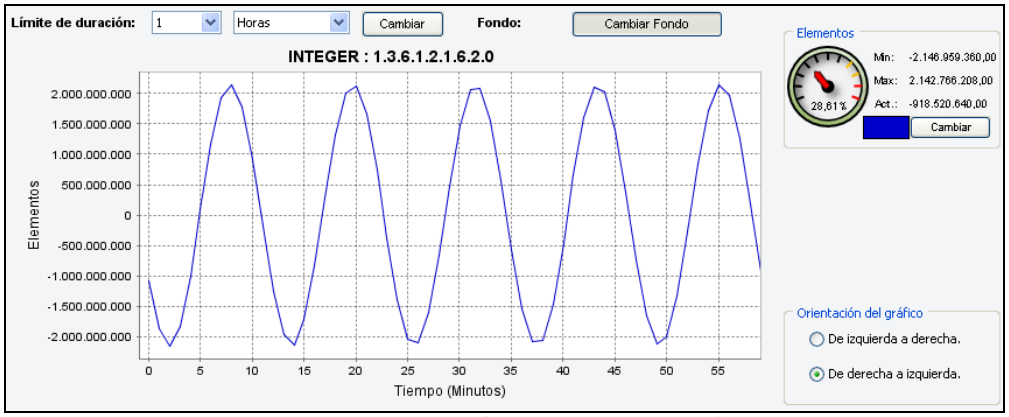


Figura 4.49: Prueba de monitorización de variables, módulo Grapher SNMP.

Luego de validar que los gráficos obtenidos se corresponden con los datos monitorizados, se proceden a verificar realizando consultas a un dispositivo real. A diferencia del caso anterior no se conoce con antelación el comportamiento de ninguna de las variables del dispositivo, por lo que para la validación se lleva control de los datos monitorizados mediante la captura de datos con Wireshark y se contrastan con los resultados obtenidos en AdminUCV NGN.

En la Figura 4.50 se puede apreciar la gráfica generada por el módulo Grapher SNMP luego de monitorizar de manera remota al host 200.84.35.6 durante 6 horas. Dicho host se comunica a través de Internet mediante una conexión ADSL de 1024 Kbps para la velocidad de bajada y 512 Kbps para la velocidad de subida.

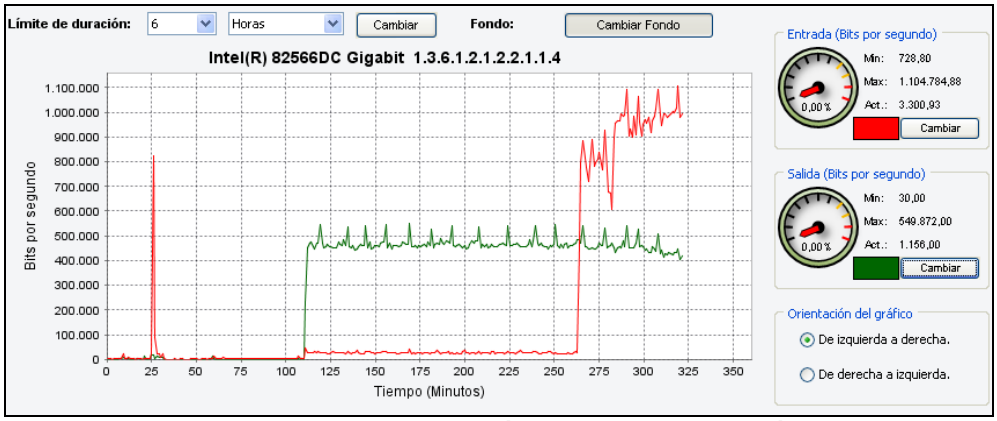


Figura 4.50: Prueba de monitorización de variables a través de Internet.

4.4.5 Iteración 5: MIB Parser

El objetivo de este módulo consiste en permitir al usuario incorporar nuevas MIBs a la aplicación facilitando el trabajo con las diferentes variables SNMP. Por lo tanto, el proceso de verificación para este módulo radica en probar dicha funcionalidad, intentando cargar MIBs de diversos fabricantes así como archivos erróneos, y comparar los resultados obtenidos con otras herramientas.

La Figura 4.51 muestra una sección del árbol generado luego de cargar la MIB descrita en la RFC 2465 conocida como IPv6-MIB.

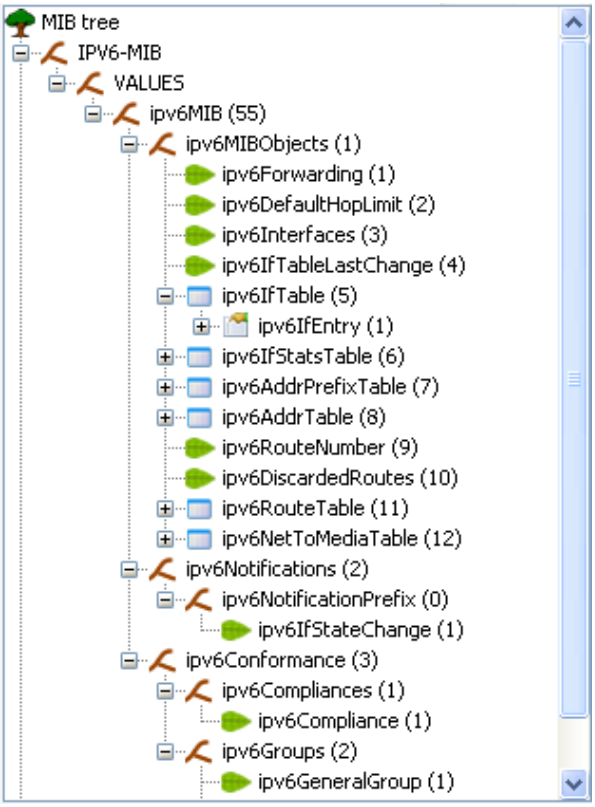


Figura 4.51: Segmento del árbol de la MIB definida en la RFC 2465 (IPv6-MIB).

4.4.6 Iteración 6: Idioma

El módulo de idioma provee las funcionalidades necesarias para que la aplicación sea multilenguaje. La versión actual de AdminUCV NGN provee soporte para los idiomas inglés y español, con la intención de que posteriormente pueda ser complementado con nuevas traducciones.

Para llevar a cabo las pruebas de funcionamiento se verifica el comportamiento de la herramienta con ausencia de alguno de los archivos de

configuración de idioma, asegurando que en dicho caso se informe al usuario del problema y no se limite el uso de la aplicación. Del mismo modo, se prueban todas las funcionalidades en ambos idiomas para garantizar que la traducción de todos los mensajes es la adecuada.

4.4.7 Iteración 7: Tabla Genérica

La tabla genérica debe permitir visualizar elementos de la MIB que conforman tablas organizados por columnas. Para garantizar la veracidad de las tablas mostradas se comparan los resultados generados por este módulo con los mostrados por otras herramientas.

En la Figura 4.52 se aprecia la tabla hrSWRunTable del host WinXpSp2 (10.10.20.2), que muestra los programas que están actualmente en ejecución.

1.3.6.1.2.1.25.4.2.1.1	1.3.6.1.2.1.25.4.2.1.2	1.3.6.1.2.1.25.4.2.1.3	1.3.6.1.2.1.25.4.2.1.4	1.3.6.1.2.1.25.4.2.1.5	1.3.6.1.2.1.25.4.2.1.6	1.3.6.1.2.1.2
1	System Idle Process	0.0			2	1
4	System	0.0			2	1
240	ntvdm.exe	0.0	C:\WINDOWS\system32\	-f -i1 -w -a C:\WINDOWS\...	4	1
256	PRTG Traffic Grapher.exe	0.0	C:\Archivos de programa\P...		4	1
348	PRTG Traffic Grapher.exe	0.0	C:\Archivos de programa\P...		4	1
468	smss.exe	0.0	\SystemRoot\System32\		4	1
512	snmp.exe	0.0	C:\WINDOWS\System32\		4	1
640	VMwareService.exe	0.0	C:\Archivos de programa\...		4	1
796	csrss.exe	0.0	C:\WINDOWS\system32\	ObjectDirectory=\Window...	4	1
828	winlogon.exe	0.0			4	1
872	services.exe	0.0	C:\WINDOWS\system32\		4	1
884	lsass.exe	0.0	C:\WINDOWS\system32\		4	1
1044	svchost.exe	0.0	C:\WINDOWS\system32\	-k DcomLaunch	4	1
1156	svchost.exe	0.0	C:\WINDOWS\system32\	-k rpcss	4	1
1248	svchost.exe	0.0	C:\WINDOWS\System32\	-k netsvcs	4	1
1340	svchost.exe	0.0	C:\WINDOWS\system32\	-k NetworkService	4	1
1380	svchost.exe	0.0	C:\WINDOWS\system32\	-k LocalService	4	1
1448	cmd.exe	0.0	C:\WINDOWS\system32\		4	1
1612	spoolsv.exe	0.0	C:\WINDOWS\system32\		4	1
1860	explorer.exe	0.0	C:\WINDOWS\		4	1
1904	SWLauncher.exe	0.0	C:\Archivos de programa\...		4	1
1920	alg.exe	0.0	C:\WINDOWS\System32\		4	1
1948	jusched.exe	0.0	C:\Archivos de programa\J...		4	1
1960	VMwareTray.exe	0.0	C:\Archivos de programa\...		4	1
1968	VMwareUser.exe	0.0	C:\Archivos de programa\...		4	1

Figura 4.52: Tabla hrSWRunTable del host WinXpSp2.

4.4.8 Iteración 8: Sniffer

Para la verificación del Sniffer se realizan capturas de todo el tráfico que transita por la red perteneciente a distintos protocolos y simultáneamente se captura con Wireshark para comparar el análisis realizado por ambas herramientas.

El proceso de verificación se extiende al generar tramas erróneas para verificar que la herramienta es capaz de interpretarlas. Del mismo modo, se generan tramas con características especiales y se verifica que son analizadas correctamente. En la Figura 4.53 se muestra el análisis realizado por el Sniffer al capturar una trama SNMP GetResponse, originada por el

host Ubuntu6 (2001:FA::2/64) ante una operación del tipo GetBulkRequest enviada por el host WinXpSp3 (2001:FA::1/64).

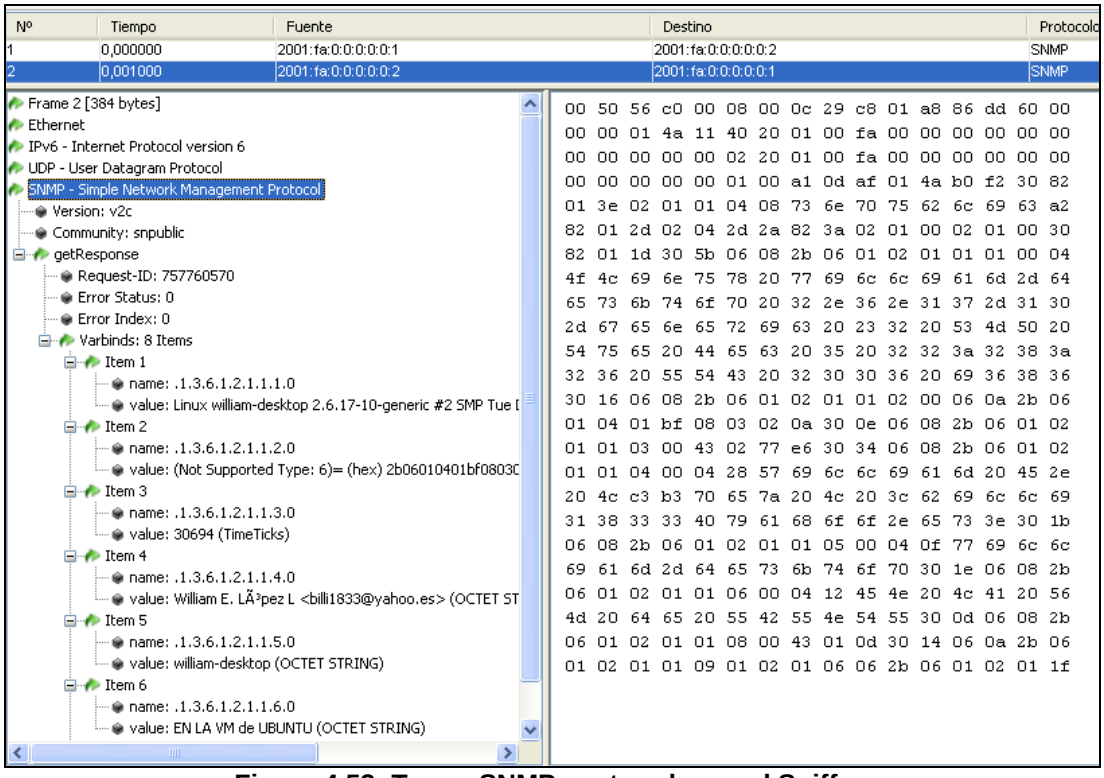


Figura 4.53: Trama SNMP capturada por el Sniffer.

En la Figura 4.54 se muestran los datos arrojados por Wireshark a partir de la captura descrita anteriormente, apreciando la similitud con las tramas capturadas por el Sniffer de AdminUCV NGN.

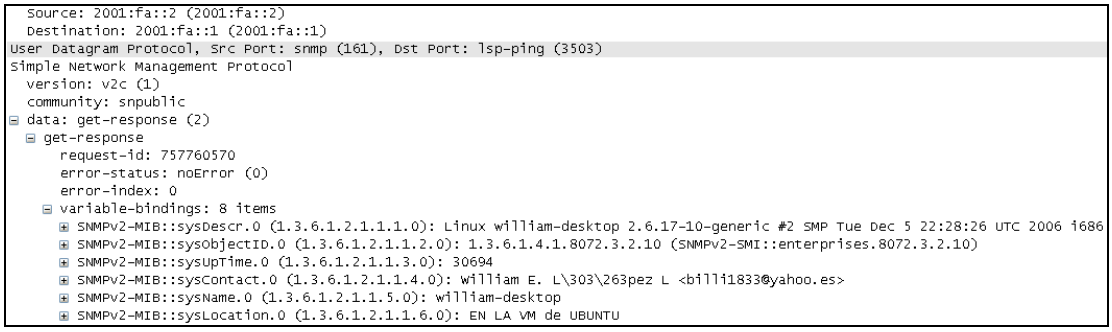


Figura 4.54: Trama SNMP capturada por Wireshark.

4.4.9 Iteración 9: Tracert

La fase de pruebas para el módulo tracert es similar a la realizada durante la verificación del módulo Ping. Este módulo fue probado en los hosts

WinXpSp2, WinXpSp3 y WinVista, desde los cuales se realizan trazas a distintos nodos. Las pruebas comienzan haciendo trazas dentro de la red de pruebas sobre la cual ya se tiene conocimiento de los resultados esperados.

Posteriormente, se realizan trazas hacia hosts externos a la red de pruebas y se comparan los resultados generados por AdminUCV NGN con los resultados obtenidos por otras herramientas, como tracert6.exe de Windows XP y la herramienta tracert.exe de Windows Vista.

En la Figura 4.55 se puede apreciar claramente que los resultados obtenidos luego de realizar una traza desde el host WinXpSp3 (2001:FA::1/64) hacia el host WinXpSp2 (2AF5:F30::2/64) corresponden con los resultados esperados según la topología de la red de pruebas.

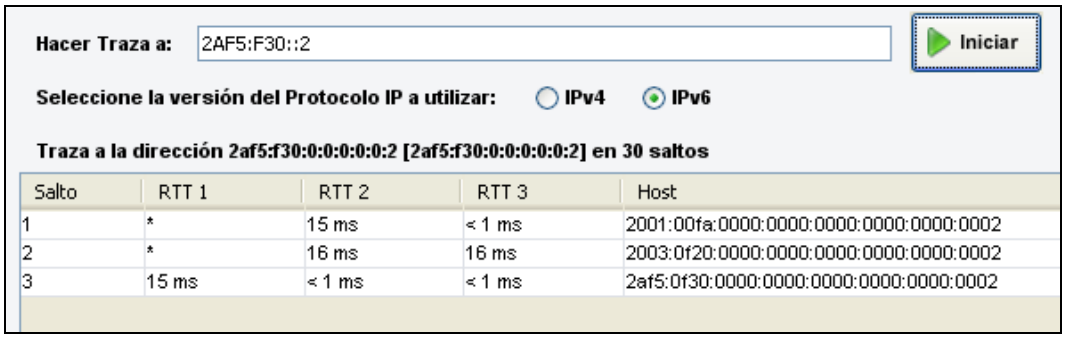


Figura 4.55: Traza IPv6 hacia el host WinXpSp2 desde el host WinXpSp3.

4.4.10 Iteración 10: Receptor de Traps

El proceso de verificación para esta funcionalidad comienza con la creación de diversos mensajes de alerta (traps) desde la herramienta SolarWinds Toolset Launchpad instalada en el host WinXpSp2. Estos se construyen con características especiales de manera tal que permitan constatar que la información mostrada por el módulo es la correcta.

Durante las pruebas se generan traps de SNMP v1/2c, mensajes InformRequest y diversos mensajes erróneos; verificando así la interpretación adecuada por parte del módulo.

Posteriormente, se configuran todos los dispositivos de la red de pruebas para que envíen alertas hacia el host WinXpSp3, que es donde se encuentra instalado AdminUCV NGN para llevar a cabo las pruebas del módulo.

En la Figura 4.56 se aprecian los detalles mostrados por el módulo al seleccionar un trap (de SNMPv2c) capturado, que fue enviado desde el host WinXpSp2 (10.10.20.2) hacia el host WinXpSp3 (192.168.30.1) e incluye el OID 1.3.6.1.2.1.1.3.0 (sysUpTime) para indicar el tiempo que tiene activo el

agente SNMP y el OID 1.3.6.1.6.3.1.1.4.1.0 con valor 1.3.6.1.6.3.1.1.5.5 para indicar que hay una falla de autenticación.

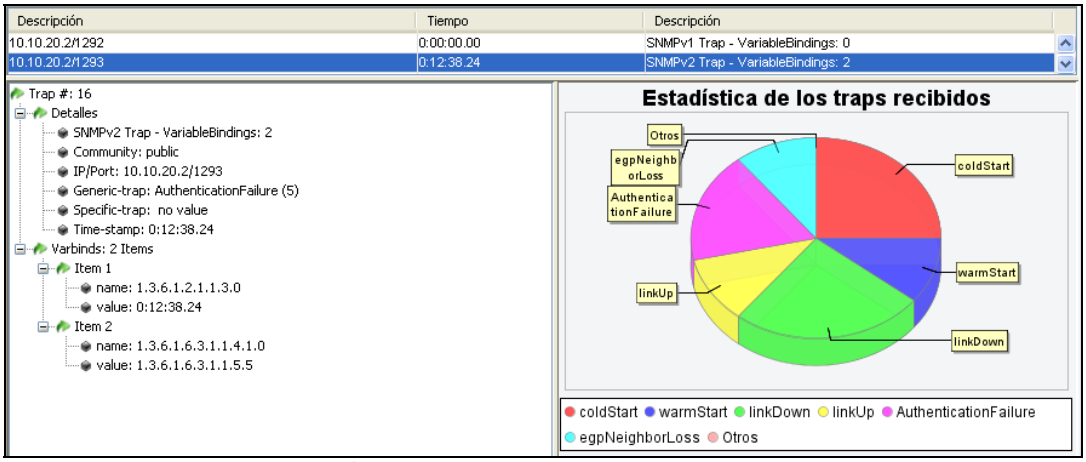


Figura 4.56: Prueba del módulo Receptor de Traps, SNMPv2 AuthenticationFailure.

Adicionalmente, en la Figura 4.56 se puede apreciar el gráfico de estadísticas generado por este módulo luego de recibir diferentes tipos de alertas.

4.4.11 Iteración 11: Escáner de Puertos

Para garantizar el correcto funcionamiento del escáner de puertos se analizan los distintos hosts de la red de pruebas, habilitando previamente en cada uno de ellos diferentes puertos TCP. Al finalizar el escaneo se comparan los resultados obtenidos con la configuración de los hosts.

Posteriormente, se realizan pruebas sobre servidores públicos de los cuales se tiene conocimiento de posibles puertos activos. Algunos de los servidores públicos consultados son:

- Servidor FTP de Cisco: ftp.cisco.com.
- Servidor SMTP de Cantv: mail.cantv.net
- Servidor HTTP de Google: www.google.com.
- Servidor HTTPS de Banesco: www.banesconline.com.

En la Figura 4.57 se muestra una vista del módulo, luego de haber finalizado el escaneo sobre el host WinXpSp2 (10.10.20.2), al cual se le habilitaron los siguientes servicios:

- FTP en el puerto 21.
- Telnet en el puerto 23.
- SMTP en el puerto 25.
- POP3 por el puerto 110.

- HTTP en el puerto 80.

Figura 4.57: Prueba del módulo Escáner de Puertos al analizar el host WinXpSp2.

En la Figura 4.58 se puede apreciar que los resultados mostrados en la Figura 4.57 coinciden con la información provista al ejecutar el comando netstat con los parámetros -a -n -p tcp en el host WinXpSp2.

```
C:\Documents and Settings\Administrador>netstat -a -n -p tcp

Conexiones activas

  Proto  Dirección local        Dirección remota       Estado
  TCP    0.0.0.0:21             0.0.0.0:0              LISTENING
  TCP    0.0.0.0:23             0.0.0.0:0              LISTENING
  TCP    0.0.0.0:25             0.0.0.0:0              LISTENING
  TCP    0.0.0.0:110            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING
  TCP    10.10.20.2:80          0.0.0.0:0              LISTENING
  TCP    10.10.20.2:139         0.0.0.0:0              LISTENING
  TCP    127.0.0.1:1027         0.0.0.0:0              LISTENING
```

Figura 4.58: Resultado de la ejecución comando netstat en WinXPSp2.

4.4.12 Iteración 12: Integración

Tal como se mencionó al inicio de la fase de pruebas, esta iteración tiene como objetivo verificar la consistencia de la interfaz gráfica de usuario y el correcto funcionamiento de cada uno de los módulos luego de haber sido integrados en la aplicación.

Para la verificación de la interfaz gráfica de usuario, se valida que cada uno de los módulos (y la aplicación en general) cumpla con algunos criterios de usabilidad que son descritos a continuación:

- **Facilitar reconocimiento instantáneo de las distintas funcionalidades de la aplicación mediante el uso de metáforas:** Consiste en validar que los iconos e imágenes utilizadas en los elementos de interfaz de cada uno de los módulos presenten una analogía clara con la funcionalidad que desempeñan, lo cual facilita al usuario el uso de las distintas herramientas y procura que este se familiarice rápidamente con el entorno de la aplicación.

  Por esta razón fueron evaluadas como metáforas, los iconos usados para la identificación de los módulos en las pestañas y las imágenes usadas en los botones.

  En la Figura 4.59 se pueden observar las metáforas utilizadas en algunos de los botones definidos dentro de la interfaz gráfica de usuario. Además, se puede notar que dichas metáforas coinciden con los símbolos utilizados de manera estándar en distintos programas, lo cual facilita al usuario la rápida identificación de las funcionalidades de cada uno de los módulos.

Figura 4.59: Metáforas utilizadas en algunos de los botones de la aplicación.

- **Mantener el diseño entre las interfaces gráficas de la aplicación:** Pueden ser definidas como pruebas de consistencia y garantizan que todos los módulos presenten una estructura similar (en lo que se refiere a la interfaz gráfica de usuario), lo cual brinda seguridad al usuario al momento de interactuar con los diferentes módulos de la aplicación. Es importante destacar, que para asegurar este aspecto se mantuvo el prototipo de interfaz definido en el capítulo 3.

  Los aspectos de consistencia también son validados al garantizar que las metáforas utilizadas sean comunes entre los módulos.

- **Retroalimentación:** Representa una característica importante en toda interfaz gráfica de usuario, ya que garantiza que a lo largo de la ejecución de la aplicación, se mantendrá informado al usuario de los eventos que están ocurriendo. Para validar esta característica se debe asegurar que cada módulo indique de manera específica cuándo se encuentra ejecutando una tarea y al finalizarla, indicar si se llevó a cabo con éxito o si se produjo algún error.

- **Prevención de errores:** Se debe reducir al mínimo la posibilidad de error mientras se ejecuta la aplicación. Para garantizarlo se pueden tomar distintas medidas, como: habilitar sólo las opciones válidas según la tarea

que se desee ejecutar en cada módulo, verificar el contenido de los datos ingresados en cualquiera de los formularios de configuración, tomando en cuenta tipos de datos permitidos para cada campo, selección de opciones obligatorias, entre otros.

- **Mensajes de error e información descriptivos utilizando un lenguaje sencillo de entender:** Consiste en verificar que cada uno de los mensajes mostrados por la aplicación indican de manera específica la información que se desea transmitir. Se debe garantizar que los errores sean expresados en un tono positivo y que se muestren sugerencias para corregir cualquier eventualidad, de forma que el usuario no se sienta reprendido o cohibido de utilizar la aplicación.

  La verificación de este aspecto consiste en procurar que se activen diversos errores y validar que el contenido de los mensajes sea adecuado. En la Figura 4.60 se aprecian algunos de los mensajes de error, advertencia e información mostrados en la aplicación.

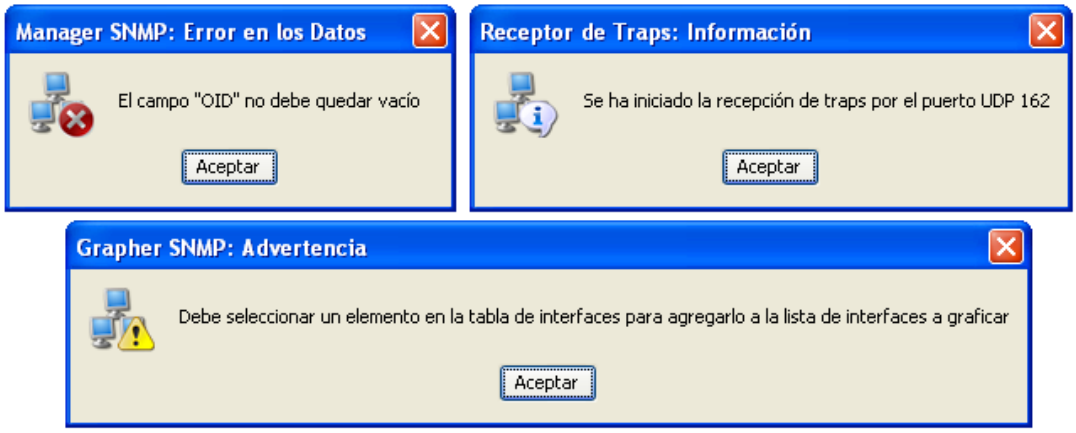


Figura 4.60: Mensajes mostrados por AdminUCV NGN.

- **Pruebas de funcionamiento integradas**

Luego de verificar los distintos aspectos que abarcan las pruebas de interfaz gráfica, se procede a realizar pruebas de funcionamiento integradas para garantizar que los módulos continúan funcionando correctamente después de haber realizado la integración.

La mayoría de las pruebas de integración son similares a las pruebas de funcionamiento, con la diferencia que en este caso se realizan en paralelo. Para esto, se ejecutan de forma simultánea los módulos y se validan los resultados obtenidos. A continuación se detalla las pruebas realizadas en paralelo a cada uno de los módulos.

- Manager SNMP y MIB Parser:

  Los módulos Manager SNMP y MIB Parser son probados simultáneamente mientras el resto de los módulos están en ejecución. En la Figura 4.61 se muestra una imagen con los resultados recibidos por el Manager SNMP, el cual se usa de manera combinada con el módulo MIB Parser para realizar una serie de consultas desde el host WinXPSp3 hacia los hosts Win2KSp4 (172.18.0.2/16) y Debian4 (172.17.20.2/16).

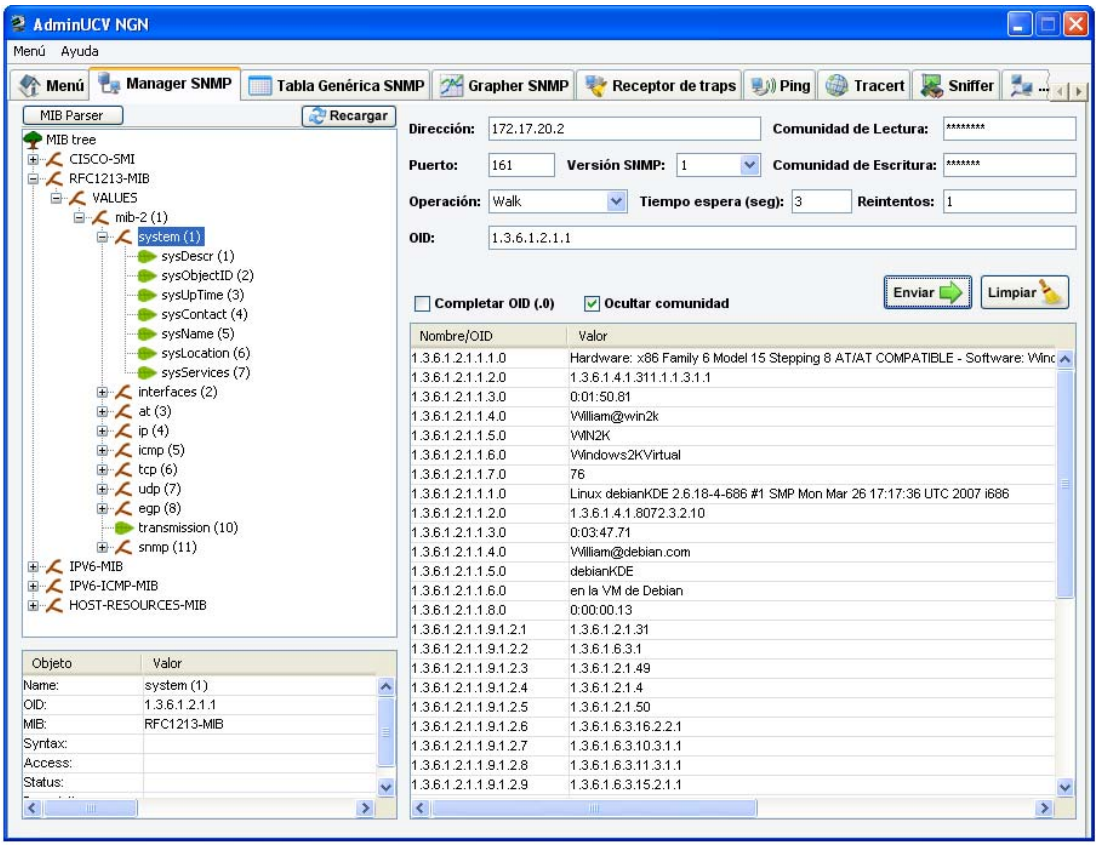


Figura 4.61: Prueba de integración para el Manager SNMP y el MIB Parser.

- Tabla Genérica SNMP:

  De manera similar se prueba el módulo para la creación de tablas SNMP, en el que se generan diversas tablas tanto de hosts pertenecientes a la red de pruebas como de hosts externos.

  En la Figura 4.62 se detalla una vista de este módulo luego de generar la tabla de enrutamiento (OID: 1.3.6.1.2.1.4.21) del host Ubuntu6 (2001:FA::2/64), la cual coincide con la tabla de enrutamiento configurada para la topología de la red de pruebas.

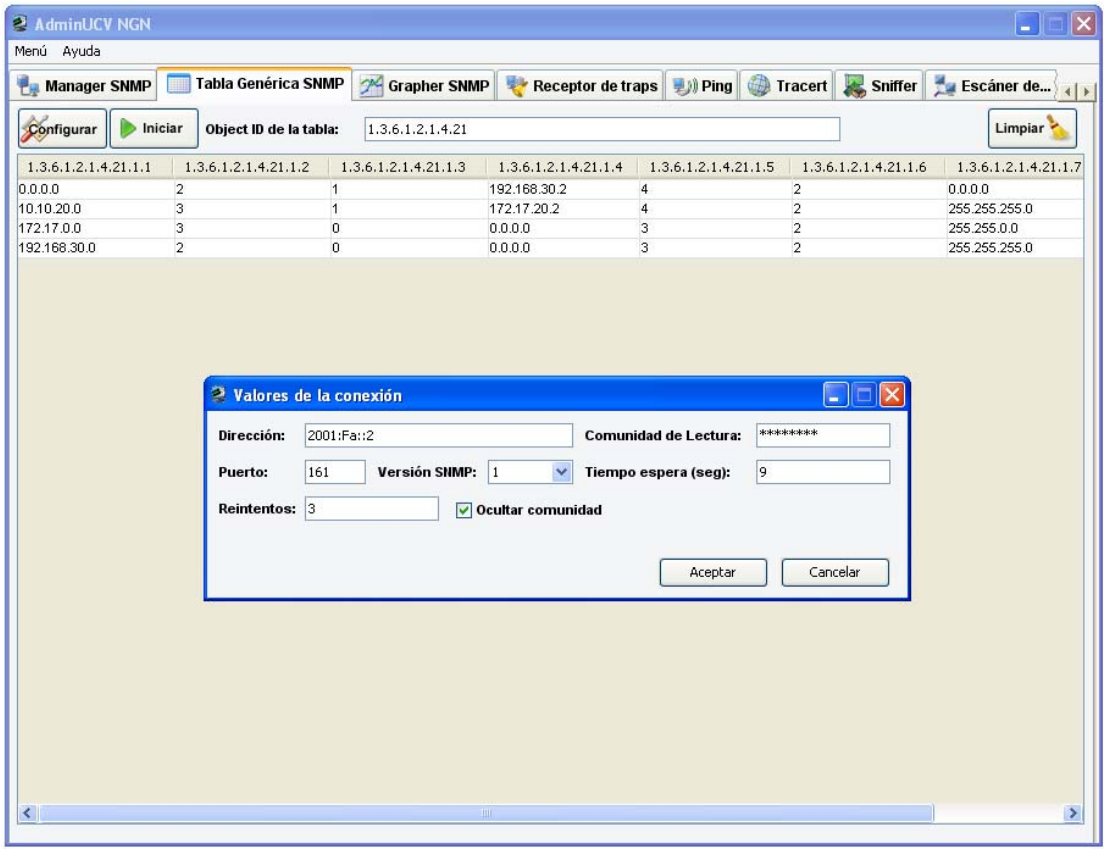


Figura 4.62: Tabla de enrutamiento del host Ubuntu6.

- Grapher SNMP:

  Este módulo se configuró al inicio de las pruebas de integración, de manera que durante dicho proceso se esté monitorizando la velocidad de transmisión de las dos interfaces de red del host WinXPSp3. La primera de estas interfaces se encuentra conectada hacia Internet mediante una conexión ADSL de 1024 Kbps para la velocidad de bajada y 512 Kbps para la velocidad de subida, la segunda interfaz está conectada a la red de pruebas mediante una conexión Ethernet de 100 Mbps.

  En la Figura 4.63 se muestran los gráficos generados por este módulo luego de monitorizar las interfaces de red del host WinXPSp3 (192.168.30.1/24) durante tres horas.

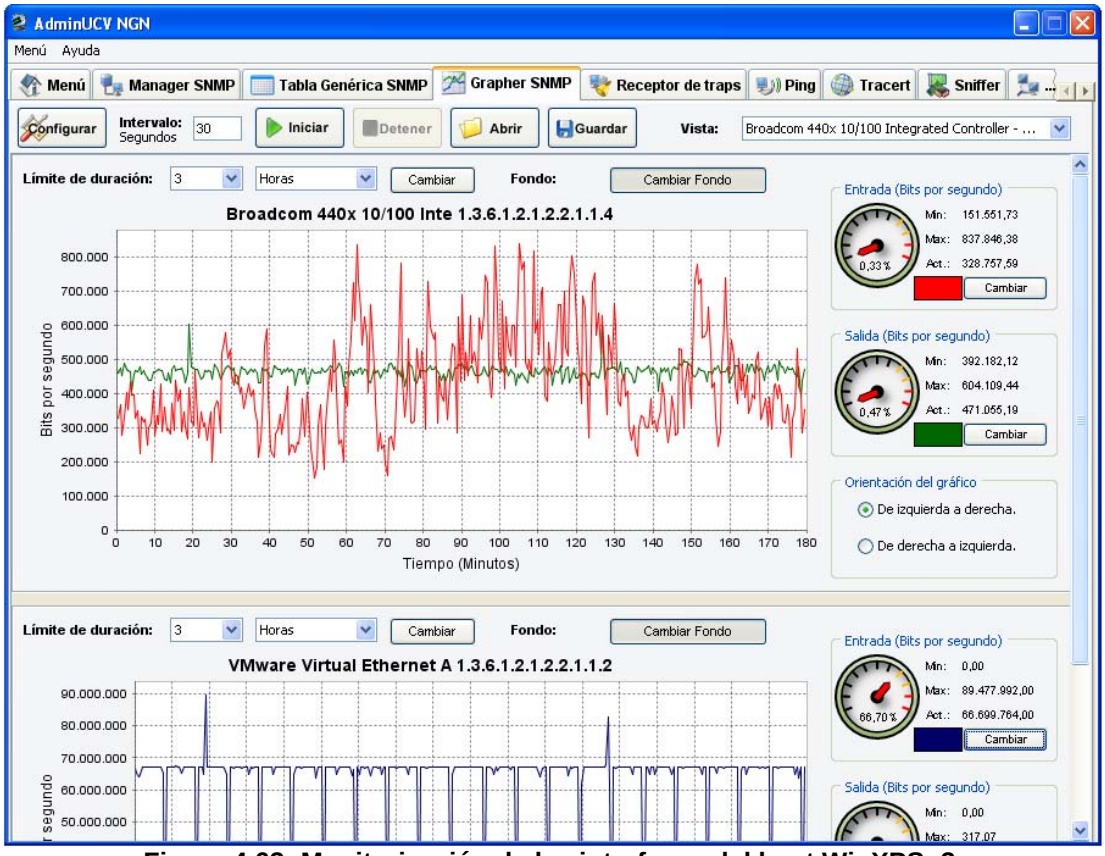


Figura 4.63: Monitorización de las interfaces del host WinXPSp3.

- Receptor de Traps:

  El receptor de traps se inicia al comienzo de las pruebas de integración para que se encuentre en ejecución durante todo el proceso de pruebas. A lo largo de este período se recibieron diversos traps (algunos generados de manera intencional a través del AdventNet Simulation Toolkit y otros de manera automática) para luego verificar los datos mostrados por el módulo.

  En la Figura 4.64 se puede apreciar la vista del módulo luego de haber recibido un total de 2204 capturas durante un período aproximado de dos horas y media. Las capturas recibidas varían entre los diversos traps soportados y algunos traps propietarios, tanto para la versión 1 como para la versión 2c.

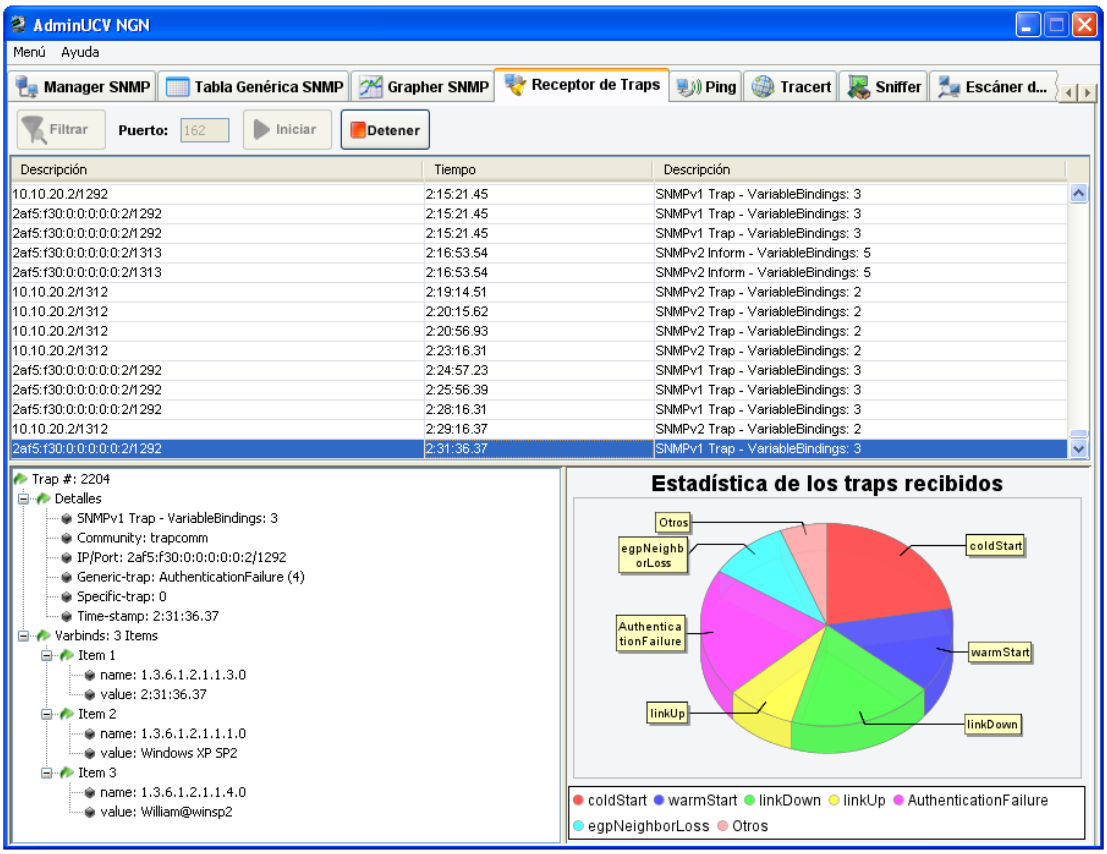


Figura 4.64: Capturas recibidas por el módulo Receptor de Traps.

- Ping:

  Para verificar el funcionamiento de este módulo se realizan diversas pruebas donde se comprueba que el formato de los mensajes enviados sea correcto y que las respuestas recibidas puedan ser interpretadas. Luego se realiza una prueba de volumen donde se configura la herramienta para que se ejecute repetidamente hasta ser detenida manualmente.

  En la Figura 4.65 se muestra una vista de este módulo luego de haberse ejecutado durante una hora consecutiva enviando mensajes hacia el host www.google.com.

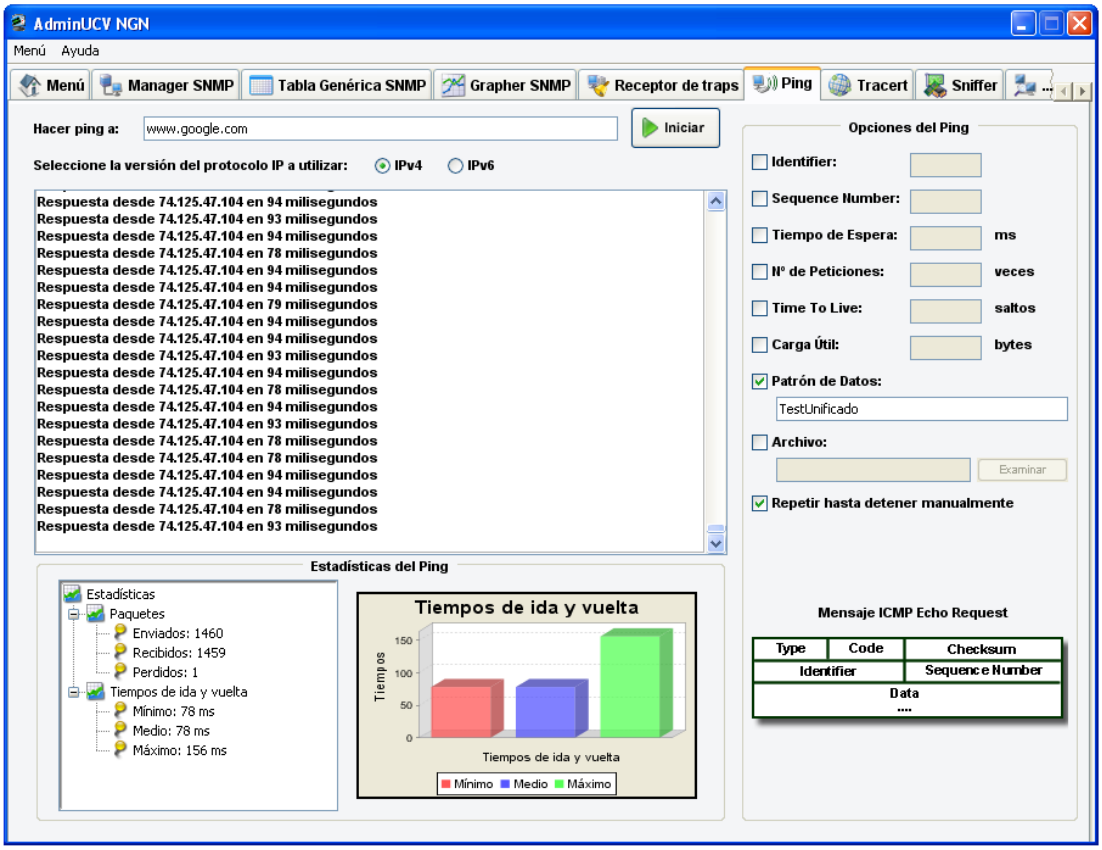


Figura 4.65: Prueba del módulo Ping hacia el servidor Web www.google.com.

- Tracert:

  De manera similar al módulo descrito anteriormente, se realizan trazas a hosts tanto de la red de pruebas como hosts externos y se verifican los resultados contrastándolos con herramientas como tracert.exe y tracert6.exe de Windows.

  En la Figura 4.66 se muestran los resultados obtenidos al realizar una traza desde el host WinXPSp3 bajo IPv4 hacia al servidor web www.ciens.ucv.ve, comparándola con la traza resultante al ejecutar la herramienta tracert.exe de Windows y que se muestra en la Figura 4.67.

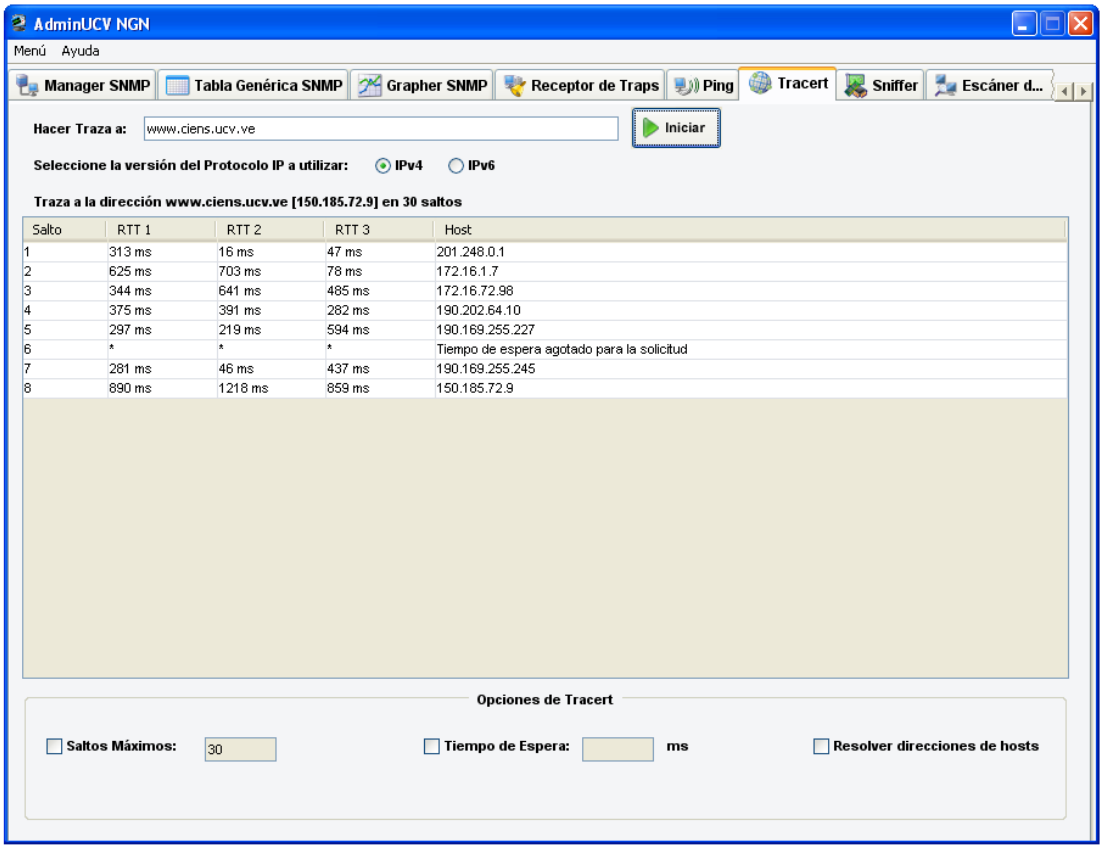


Figura 4.66: Prueba del módulo Tracert hacia el host www.ciens.ucv.ve.

```
C:\>tracert www.ciens.ucv.ve -d

Traza a la dirección www.ciens.ucv.ve [150.185.72.9]
sobre un máximo de 30 saltos:

  1    676 ms    748 ms    998 ms     201.248.0.1
  2    910 ms   1056 ms    664 ms     172.16.1.7
  3    813 ms    663 ms    354 ms     172.16.72.98
  4   1224 ms    574 ms   1027 ms     190.202.64.10
  5    487 ms    325 ms   1071 ms     190.169.255.227
  6      *         *         *        Tiempo de espera agotado para esta solicitud.
  7    979 ms    361 ms    405 ms     190.169.255.245
  8    537 ms    443 ms   1164 ms     150.185.72.9
```

Figura 4.67: Resultados obtenidos por tracert.exe de Windows XP SP3.

- Sniffer:

  Durante el proceso de integración, se dejó capturando el Sniffer mientras se ejecutaban los módulos restantes, de manera de verificar el análisis de las distintas tramas generadas por las herramientas de AdminUCV NGN y el tráfico que durante ese período transita por la red. Al mismo tiempo, se

estuvo ejecutando Wireshark con la finalidad de validar las tramas capturadas. En la Figura 4.68 se aprecia un conjunto de las 78825 tramas capturadas durante dos horas y media aproximadamente.

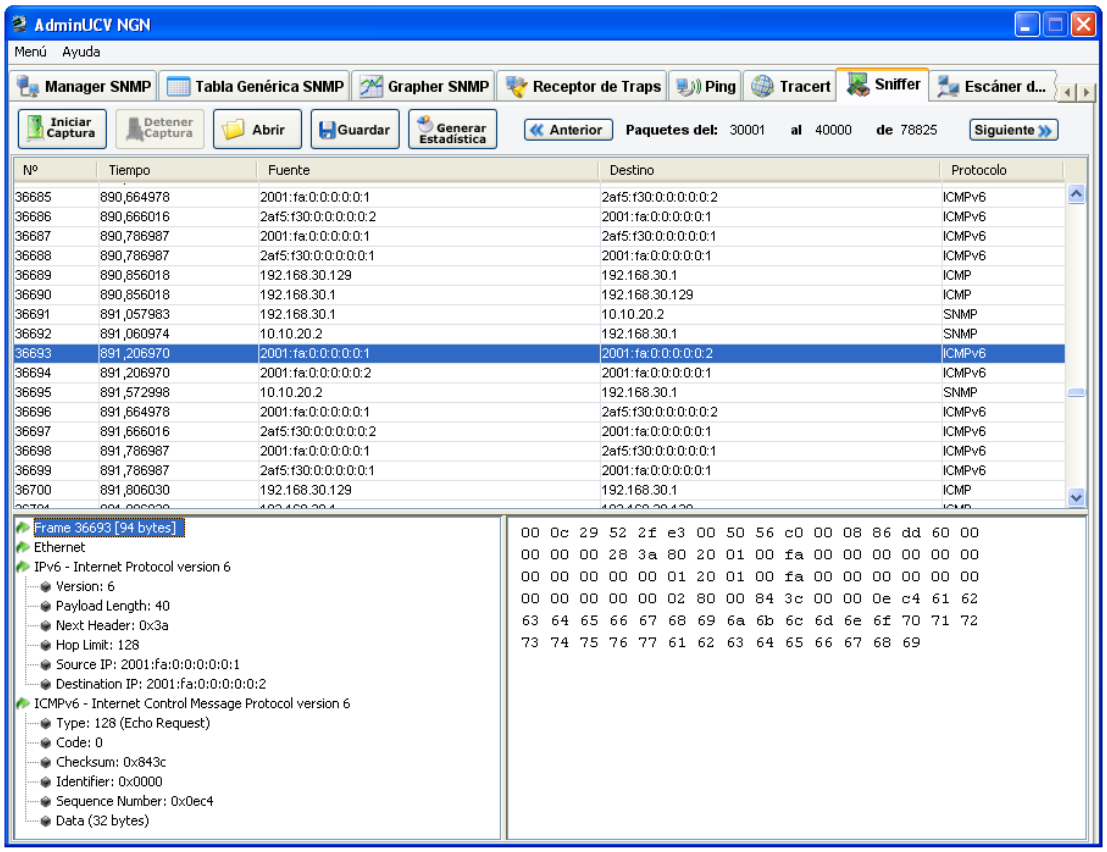


Figura 4.68: Captura realizada por el Sniffer durante las pruebas de integración.

- Escáner de Puertos:

  Este módulo se valida al escanear el rango de direcciones IPv6 comprendido entre la dirección 2001:FA:: y la dirección 2001:FA::5 perteneciente a la red de pruebas, donde se activaron previamente el siguiente conjunto de puertos TCP: 66, 77, 88, 99, 123 y 456 en el host Ubuntu6 (2001:FA::2/64) y los puertos 45, 56, 67, 78, 89, 91, 321 y 654 en el host WinVista (2001:FA::3/64).

  En la Figura 4.69 se muestran los puertos activos luego de la monitorización de dicho rango.

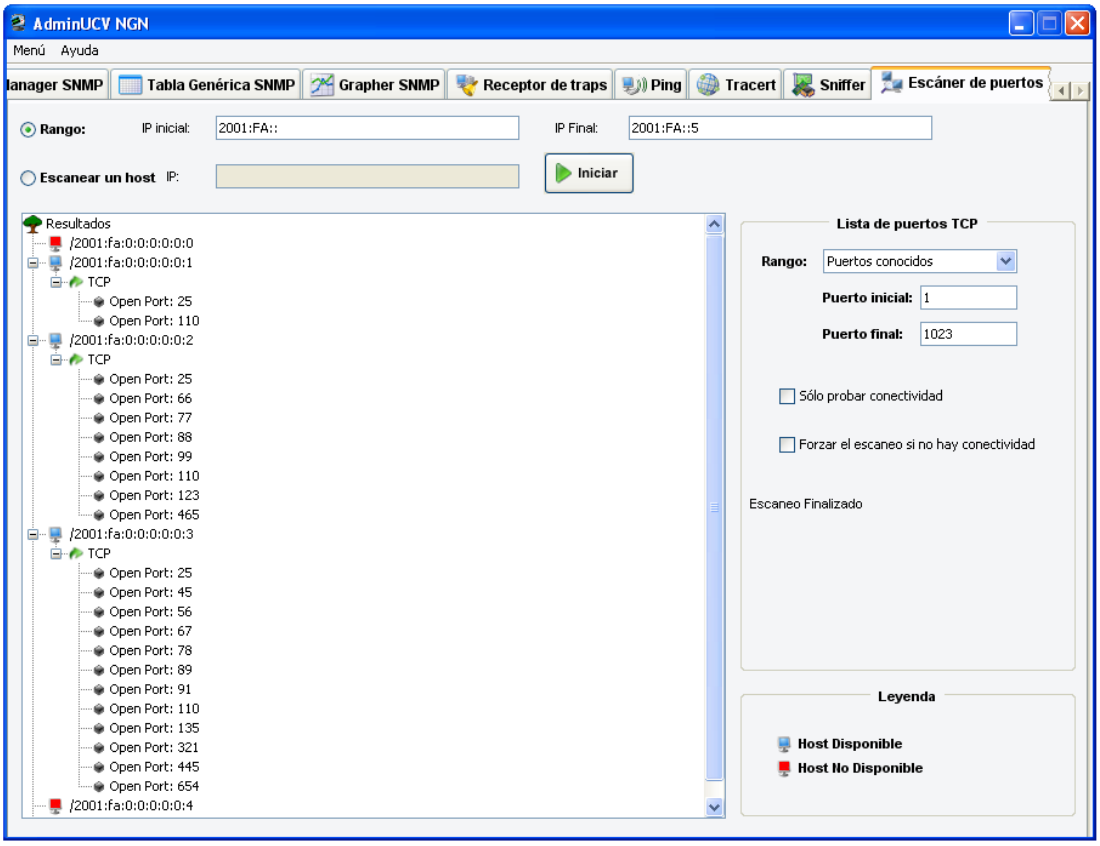


Figura 4.69: Resultados de la prueba de integración para módulo Escáner de Puertos.

Para finalizar las pruebas de integración se verifica el contenido de todas las variables de AdminUCV NGN en el idioma inglés.

Al culminar esta iteración, se puede afirmar el correcto funcionamiento de cada uno de los módulos trabajando en paralelo y la robustez de la aplicación al soportar grandes cantidades de datos, manteniendo tiempos de respuesta y resultados adecuados.

5. Conclusiones

AdminUCV NGN es una aplicación de código abierto para la configuración y administración de redes vía SNMP con soporte para IPv4 e IPv6, siendo este último el protocolo pensado para ser la base que fundamente a las Redes de Próxima Generación.

Para su implementación fue utilizado Java como lenguaje de programación, el cual permitió cubrir la mayoría de los objetivos planteados como Trabajo Especial de Grado. Dicho lenguaje cuenta con un API extenso, del cual se utilizó Swing como paquete para generar interfaces gráficas basadas en criterios de usabilidad.

Por las características de Java como lenguaje, existen múltiples librerías de código abierto que amplían las capacidades del mismo y que facilitaron el desarrollo de algunas de las herramientas provistas por la aplicación. Algunas de estas librerías son:

- SNMP4j, para dar soporte al protocolo SNMP en su versión 1 y 2c, del cual dependen los módulos: Manager SNMP, Grapher SNMP, Tabla Genérica y Receptor de Traps.
- Mibble, para facilitar el proceso de leer y compilar MIBs.
- Jpcap, que permite la captura y análisis de la información que pasa por la red. Para el análisis de paquetes ICMPv6 y SNMP se implementaron los métodos necesarios para dar soporte a dichos protocolos dentro de la aplicación.
- JFreeChart, del cual se utilizó un amplio conjunto de funcionalidades para generar gráficas dentro de la aplicación.

Es importante destacar, que durante la implementación se encontró que el protocolo ICMP no es soportado por Java, por lo que se solventó dicha limitación creando algunas herramientas bajo el lenguaje C++, que aparte de ser muy amplias en cuanto a las funcionalidades que ofrecen, pudieron ser integradas perfectamente a la aplicación. Estas herramientas, utilizan algunos paquetes específicos para Windows, lo cual restringe la portabilidad de la aplicación a dicho sistema operativo. Sin embargo, la aplicación fue diseñada e implementada de manera que al adaptar las herramientas creadas en C++ por herramientas desarrolladlas bajo sistemas Unix sea completamente portable.

Gran parte del trabajo realizado, se centró en el estudio de IPv6 como un protocolo que a pesar de ser relativamente nuevo, posee suficiente potencial como para permitir la integración de múltiples usuarios y servicios en una misma red. Algunas de las propiedades que lo definen son la

autoconfiguración, soporte mejorado para QoS, características de seguridad integrada que proveen la base para el desarrollo de nuevos protocolos de capa superior y al ser extensible, garantiza su crecimiento para sustentar cada vez más requerimientos.

El conjunto de características que describen IPv6 incrementan la posibilidad de que protocolos ya existentes y sencillos como SNMP puedan adaptarse fácilmente a las mejoras que trae consigo dicho protocolo y mantener su simplicidad aprovechando las ventajas del mismo. Debido a que SNMP se fundamenta en la consulta y modificación de la información provista por las variables definidas en los agentes, se garantiza que con la aparición de nuevas MIBs se permitirá manipular los valores de las variables de prácticamente cualquier dispositivo, lo cual garantiza que SNMP continúe siendo considerado un estándar dentro de los protocolos de administración de redes que promete avanzar para dar soporte a las Redes de Próxima Generación.

Como parte de los objetivos planteados al inicio de la investigación, AdminUCV NGN ha sido publicada con licencia GNU GPL y puede ser descargada en cualquiera de sus formatos a través del enlace: http://adminucvngn.sourceforge.net/ perteneciente al sitio Web SourceForge.net, el cual permite la distribución de proyectos de software libre y código abierto. Este hecho, brinda la oportunidad de que colaboradores a nivel mundial prueben la aplicación y publiquen mejoras de la misma.

AdminUCV NGN constituye una herramienta ideal para ser utilizada a nivel didáctico en universidades, complementar el estudio de algunos protocolos y evaluar el comportamiento de la red. Adicionalmente, es recomendada a administradores de redes para monitorizar el estado de la misma, teniendo a su alcance herramientas complementarias que faciliten su labor. A pesar de que todos los objetivos propuestos al inicio de la investigación fueron alcanzados con éxito, a continuación se plantean algunos trabajos complementarios que enriquecerían de gran manera a la aplicación:

- Análisis de diversos protocolos de capa de aplicación adicionales a SNMP para complementar el Sniffer.
- Manipulación de más de una variable por cada operación SNMP.
- Monitorización de múltiples dispositivos simultáneamente.
- Ampliación de los idiomas soportados por el sistema.
- Adaptación a los cambios relacionados con protocolos como SNMP e IPv6 y los diferentes sistemas operativos.

Referencias Bibliográficas

[01] R. Rodríguez, L. Sánchez. Implantación y Evaluación de un Sistema de Gestión de Redes (NMS) basado en Código Abierto. Universidad Metropolitana. Agosto, 2003.

[02] V. Mendillo. Discos Digitales para la Gestión de Redes. Agosto, 2007.

[03] A. Clemm. Network Management Fundamentals. Cisco Press. Noviembre, 2006.

[04] A. Pras. Network Management. Architectures. Henuelo. Enero, 1995.

[05] D. Mendiola Oñate. Introducción a la Gestión de Redes y Servicios. Universidad de Murcia. Junio, 1997.

[06] V. Juárez, et al. Conjunto de Protocolos de Open System Interconnection. http://www.geocities.com/macsite3/trabajo.htm. Enero, 2004.

[07] M. Ramos. Modelos de Gestión de Red. Universidad de Vigo. Enero, 2001.

[08] R. Calcagno, et al. Funciones de Soporte de la Administración de Sistemas. Universidad Tecnológica Nacional. Febrero, 2002.

[09] D. Mauro, K. Schmidt. Essential SNMP, 2nd Edition. O'Reilly. Septiembre, 2005.

[10] M. Rose, K. McCloghrie. Structure and Identification of Management Information for TCP/IP-based Internets. RFC 1155. Mayo, 1990.

[11] M. Miller. Managing Internetworks with SNMP. M & T Books. Mayo, 1997.

[12] K. McCloghrie, D. Perkins, J. Schoenwaelder. Structure of Management Information Version 2 (SMIv2). RFC 2578. Abril, 1999.

[13] S. Hagen. IPv6 Essentials, 2nd Edition. O'Reilly. Mayo, 2006.

[14] S. Deering, R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460. Diciembre, 1998.

[15] R. Hinden, S. Deering. IP Version 6 Addressing Architecture. RFC 4291. Febrero, 2006.

[16] J. Davies. Understanding IPv6. 2nd Edition. Microsoft Press.Enero, 2008.

[17] A. Conta, A. Deering, M. Gupta. Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification. RFC 4443. Marzo, 2006.

[18] Free Software Foundation, Inc., The Free Software Definition. http://www.gnu.org/philosophy/free-sw.html. Mayo, 2008.

[19] Open Source Initiative, The Open Source Definition. http://www.opensource.org/docs/osd. Mayo, 2008.

[20] Fundación Copyleft, ¿Qué es copyleft?. http://fundacioncopyleft.org/es/9/que-es-copyleft. Mayo, 2008.

[21] M. Fowler. The New Methodology. http://www.martinfowler.com/articles/newMethodology.htm. Mayo, 2008.

Glosario de Términos

Administración de Redes: Es el conjunto de actividades que se basan en el acoplamiento de diversas tecnologías para planificar, modelar, diseñar, configurar y desarrollar redes informáticas con la finalidad de obtener un óptimo desempeño a un costo razonable y con la máxima eficiencia; dando así la capacidad de mantener, corregir, contabilizar, evaluar y expandir los recursos que la componen.

Agente: El agente es el rol asignado al dispositivo que va a ser administrado. Responde a las peticiones realizadas por el manager y envía notificaciones no solicitadas para alertar al manager de algún comportamiento específico de la red.

ARP (Address Resolution Protocol): El Protocolo de Resolución de Direcciones es un protocolo encargado de encontrar la dirección física (MAC) que corresponde a una determinada dirección IP dada.

ASN.1 (Abstract Syntax Notation One): Es un lenguaje formal utilizado para describir mensajes que pueden ser intercambiados entre gran cantidad de aplicaciones. En SNMP se utiliza ASN.1 para la representación de los objetos administrados.

BER (Basic Encoding Rules): Representan uno de los formatos de codificación definidos como parte del estándar ASN.1.

C++: Es un lenguaje de programación, diseñado a mediados de los años 1980, por Bjarne Stroustrup, como extensión del lenguaje de programación C. Abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos.

Calidad de Servicio: Calidad de Servicio (QoS, Quality of Service) se refiere a la capacidad de una red de proveer mejor servicio a un tráfico en particular sobre diversas tecnologías incluyendo: Frame Relay, ATM, Ethernet , 802.1, SONET, y redes IP.

Checksum: Una suma de verificación o checksum es una forma de control de redundancia muy simple para proteger la integridad de datos, verificando que no hayan sido corrompidos.

CMIP (Common Management Information Protocol): Es un protocolo para la administracion de Redes, basado en el modelo OSI y diseñado por IUT-T.

CMOT (CMIP over TCP/IP): es la versión del protocolo CMIP para trabajar sobre redes TCP/IP.

Código Abierto: Código Abierto se define como todo programa que es desarrollado y distribuido libremente bajo la filosofía de que al compartir el código, el programa resultante tiende a ser de mayor calidad.

Copyleft: Copyleft es el tipo de protección jurídica que otorgan un grupo de licencias para garantizar el derecho de cualquier usuario a utilizar, modificar y redistribuir un programa o sus derivados, siempre que se mantengan las mismas condiciones de utilización y difusión.

Counter: Counter es un tipo de dato representado por un entero no negativo que puede incrementar hasta alcanzar un valor máximo, donde se reinicia a cero. La RFC 1155 especifica que su valor máximo permitido es 2^32 – 1 (4294967295).

DNS (Domain Name Server): Es un protocolo que define un servicio automatizado que coincide con nombres de recursos que tienen la dirección de red numérica solicitada.

Escáner de Puertos: Un escáner de puertos es una herramienta que permite analizar el estado de los puertos de una máquina conectada a una red de comunicaciones.

Ethernet: Ethernet es un protocolo de capa de enlace de datos, que brinda acceso al medio por contención mediante la técnica CSMA/CD.

FTP (File Transfer Protocol): Es un protocolo de la capa de aplicación que permite las transferencias de archivos entre un cliente y un servidor.

Gauge: Gauge es tipo de dato representado por un entero no negativo, que puede incrementar o decrementar. La RFC 1155 especifica que su valor máximo permitido es 2^32 – 1 (4294967295).

GNU: GNU es un acrónimo recursivo que significa GNU No es Unix (GNU is Not Unix). El proyecto GNU fue iniciado por Richard Stallman con el objetivo de crear un sistema operativo completamente libre: el sistema GNU.

GPL (General Public License): Es un tipo de licencia copyleft, que establece un conjunto específico de términos de distribución que protegen un software para garantizar la libertad de compartirlo y modificarlo, asegurando que sea software libre para todos sus usuarios.

HTTP (Hyper Text Transfer Protocol): El protocolo de transferencia de hipertexto es uno de los protocolos del grupo TCP/IP que se desarrolló para publicar y recuperar las páginas HTML, y en la actualidad se utiliza para sistemas de información distribuidos y de colaboración. HTTP se utiliza a través de la World Wide Web para transferencia de datos y es uno de los protocolos de aplicación más utilizados.

HTTPS: HTTPS o HTTP seguro es el protocolo utilizado para proveer una comunicación segura a través de Internet, para acceder o subir información al servidor Web. HTTPS puede utilizar autenticación y encriptación para asegurar los datos cuando viajan entre el cliente y el servidor; especifica reglas adicionales para pasar los datos entre la capa de Aplicación y la capa de Transporte.

ICMP (Internet Control Message Protocol): Es un protocolo utilizado para el control y notificación de errores en el Protocolo Internet.

ICMPv6 (Internet Control Message Protocol version 6): Es la versión 6 del protocolo ICMP.

IETF (Internet Engineering Task Force): Grupo de Trabajo de Ingeniería de Internet. Grupo responsable del desarrollo, mantenimiento y difusión de los RFCs.

INTEGER: Es un tipo de dato en SNMP representado por un número entero.

IPv4 (Inernet Protocol version 4): Representa la versión 4 del Protocolo de Internet. Es el protocolo de capa de red ampliamente utilizado y que en la actualidad está siendo reemplazado por su sucesor IPv6, debido al agotamiento de direcciones en IPv4.

IPv6 (Internet Protocol version 6): Es un nuevo protocolo perteneciente a la capa de red, diseñado para reemplazar al actual Protocolo de Internet (IPv4) con la finalidad de solventar los problemas de direccionamiento que existen en la actualidad.

ISO (International Organization for Standardization): Organización Internacional para la Estandarización. Es el organismo encargado de promover el desarrollo de normas internacionales de fabricación, comercio y comunicación para todas las ramas industriales a excepción de la eléctrica y la electrónica.

Java: Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más

simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

JRE (Java Runtime Environment): Corresponde con un conjunto de utilidades que permite la ejecución de programas Java sobre todas las plataformas soportadas.

Manager: Es el rol que se asigna al dispositivo donde se encuentra instalado el NMS o Sistema para Administración de Redes. El Manager es el encargado de realizar consultas al agente.

MIB (Management Information Base): Base para la Información de Administración. Es la definición de una base de datos conceptual (utilizando la sintaxis SMI) en donde se mantiene información de todos los objetos administrados (MOs) presentes en el agente.

MO (Managed Object): Un objeto administrado es una abstracción en la que se representan las propiedades relevantes para la administración de un recurso.

MTU (Maximum Transfer Unit): La Unidad Máxima de Transferencia es un término que expresa el máximo tamaño en bytes que puede pasar por una capa de un protocolo.

NGN (Next Generation Network): Es un nuevo concepto que surge para proponer la integración de múltiples servicios que trabajan en distintas plataformas en una arquitectura común basada en una red IP.

NMS (Network Management System): Es un software que unifica una serie de funcionalidades para administrar una red.

Objeto Administrado: Ver *MO*.

OID: Un OBJECT IDENTIFIER es la representación asociada al nombre de cada uno de los objetos administrados.

PDU (Protocol Data Unit): Es el término utilizado para describir datos mientras se mueve de una capa del modelo OSI a otra.

Ping: Es una herramienta que permite probar la conectividad entre dos hosts. Se basa en el envío de mensajes ICMP de Echo Request para conocer si un dispositivo se encuentra activo.

POP3: Es un protocolo de la capa de aplicación que se encarga del manejo de correos entrantes, utilizando el puerto 110 por defecto para establecer una conexión TCP.

QoS: Ver *Calidad de Servicio*.

RFC (Requests for Comments): Las Solicitudes de Comentarios son una serie de documentos y memorandos que abarcan nuevas investigaciones, innovaciones y metodologías aplicables a las tecnologías de Internet. Las RFCs son una referencia sobre la forma en que deberían funcionar las tecnologías.

Sistema para Administración de Redes: Ver *NMS*.

SMI (Structure Management Information): Estructura de Información de Administración. Especifica una manera para definir los objetos administrados y su comportamiento.

SMTP (Simple Mail Transfer Protocol): Protocolo Simple para Transferencia de Correo. Es un protocolo de la capa de aplicación, se basa en el modelo cliente-servidor y se encarga del manejo del correo saliente, donde un cliente envía un mensaje a uno o varios receptores. Normalmente, utiliza el puerto 25 en el servidor para establecer la conexión.

Sniffer: Un sniffer es un programa que permite la captura de los paquetes que transitan por la red y el análisis de los protocolos que componen el mismo.

SNMP (Simple Network Management Protocol): Protocolo Simple para Administración de Redes. Es el protocolo para administración de redes más popular. Se encuentra definido en la RFC 1157, está diseñado para trabajar en la capa de aplicación y utiliza los servicios de transporte UDP, por medio de los puertos 161 para intercambio de datos y el puerto 162 para alertas.

Software Libre: Software Libre se considera todo aquel programa que ha sido creado para que cualquier usuario que lo adquiera pueda ejecutar, copiar, distribuir, estudiar, modificar y mejorar dicho software. Es importante destacar que “libre” no implica gratis, y que todo software libre garantiza a los usuarios 4 libertades, como: Usar el programa con cualquier propósito, estudiar cómo funciona el software y adaptarlo a sus necesidades, distribuir copias para ayudar a otros y mejorar el programa y hacer públicas dichas mejoras

TCP (Transmission Control Protocol): Protocolo de Control de Transmisión. Es un protocolo de capa de transporte orientado a la conexión,

descrito en la RFC 793. Se caracteriza por proveer entrega confiable y control de flujo.

Telnet: Es un protocolo que permite acceder en forma remota a los sistemas informáticos de la misma manera en que se puede hacer con las terminales conectadas en forma directa. El protocolo y el software del cliente que implementa el protocolo comúnmente se definen como Telnet.

TOS (Type of Service): Es uno de los campos que conforma la cabecera de IPv4. Fue definido para permitir el manejo de calidad de servicio en esta versión del protocolo y se basa en la asignación de prioridades de acuerdo al tipo de paquete.

Tracert: Es una utilidad que permite observar la ruta entre dos hosts. El rastreo genera una lista de saltos alcanzados con éxito a lo largo de la ruta. Al igual que la herramienta ping, se basa en el envío de mensajes ICMP Echo Request variando el campo TTL o Hop Limit (según la versión IP) para conocer cada uno de los saltos por los que transita un paquete para llegar a su destino.

Trap: Es una operación utilizada por el agente de forma asíncrona (sin haberse realizado una solicitud previa), para indicar al manager que ha ocurrido un evento inesperado.

UDP (User Datagram Protocol): Protocolo de Datagramas de Usuario. Es un protocolo simple, sin conexión, descrito en la RFC 768. Cuenta con la ventaja de proveer la entrega de datos sin utilizar muchos recursos.

Wireshark: Es un analizador de protocolos utilizado para realizar análisis y solucionar problemas en redes de comunicaciones para desarrollo de software y protocolos, y como una herramienta didáctica para educación.