ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

DISEÑO DE APLICACIONES PARA LA OPTIMIZACIÓN DE LA GESTIÓN DE RED CON LA PLATAFORMA NETRAMET

AUTOR: José María Carrillo Martínez
DIRECTOR: José Fernando Cerdán Cartagena

Julio / **2003**

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

PROYECTO FIN DE CARRERA

DISEÑO DE APLICACIONES PARA LA OPTIMIZACIÓN DE LA GESTIÓN DE RED CON LA PLATAFORMA NETRAMET

AUTOR: JOSÉ MARÍA CARRILLO MARTÍNEZ
DIRECTOR: JOSÉ FERNANDO CERDÁN CARTAGENA

MIEMBROS DEL TRIBUNAL:

PRESIDENTE: PABLO LÓPEZ-MATENCIO PÉREZ

SECRETARIO: JOSÉ FERNANDO CERDÁN CARTAGENA

VOCAL: CRISTINA VICENTE CHICOTE

DEPARTAMENTO: TECNOLIGÍAS DE LA INFORMACIÓN Y LAS COMUNICACIONES

TITULACION: INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN, ESPECIALIDAD TELEMÁTICA

Autor	José María Carrillo Martínez
E-mail del Autor	jose.carrillo@ono.com
Director	José Fernando Cerdán Cartagena
E-mail del Director	fernando.cerdan@upct.es

Título del PFC	DISEÑO DE APLICACIONES PARA LA OPTIMIZACIÓN DE LA GESTIÓN DE RED CON LA PLATAFORMA NETRAMET

Resúmen

La plataforma NeTraMet adolece de ciertas deficiencias, que hacen de su uso poco amigable al usuario. En este proyecto aportaremos las aplicaciones software necesarias, que faciliten y mejoren la configuración y utilización de la plataforma NeTraMet en cualquier tipo de red, y de la forma mas transparente para el usuario.

Titulación	INGENIERIA TÉCNICA DE TELECOMUNICACIÓN, ESPECIALIDAD TELEMÁTICA
Departamento	TECNOLOGÍA DE LA INFORMACIÓN Y LAS COMUNICACIONES
Fecha de Presentación	Julio - 2003

INDICE

Capítulo 1

INTRODUCCION

La popularidad de Internet ha traído consigo un crecimiento del tráfico (no solo en volumen, sino también en su naturaleza) y las aplicaciones utilizadas para el acceso a la red. Todo esto a su vez provoca la evolución de su infraestructura. La naturaleza del tráfico de Internet puede ser impredecible y muy variable, por lo que existen muchos estudios teóricos acerca de su comportamiento. Estas medidas de comportamiento del tráfico nos pueden servir para tomar decisiones en el caso de demanda de la población (la cuál aumenta de forma exponencial).

La gestión de red nos permite actuar de forma estratégica para obtener una mejor tecnología y ofrecer un mejor servicio (aprovechando el ancho de banda, el soporte físico, etc.). Estos sistemas de gestión abarcan cuatro niveles:

- Detección de fallos.
- Gestión de la configuración de la red.
- Realización de la gestión del tráfico.
- Administración de los sistemas de red.

Para gestionar las redes necesitamos utilizar algún tipo de aplicación que nos ayude a controlar el volumen de tráfico y poder saber así el estado de la red. En este proyecto nos hemos basado en las aplicaciones RTFM (Realtime Traffic Flow Measurement) [1] [9] [11], plataformas como NeTraMet [8] [9].

Todas las plataformas basadas en RTFM tienen en común una misma forma de trabajar, consta de un medidor, un colector y un administrador que interactúan entre ellos, y cuyo fin es el estudio del tráfico para ayudar a la gestión de la red.

- La plataforma cuenta con un medidor en cada punto de red donde vayamos a medir el tráfico.
- Un colector para recoger los datos medidos por los medidores y almacenarlos en algún fichero.

- Un manager para controlar los medidores y el colector.

El desarrollo de este proyecto ha seguido los siguientes puntos:

En primer lugar pretendemos conocer el funcionamiento de las aplicaciones RTFM centrándonos en la plataforma NeTraMet43 que podemos descargarla del FTP de NeTraMet [18].

Estudiar y entender el lenguaje SRL [16] que nos proporciona esta plataforma para la generación de fichero de reglas, que son cargados en el medidor para el filtrado del tráfico.

Una vez comprendido el lenguaje SRL ha sido realizada una aplicación que facilitara la creación de estas reglas sin necesidad de entender el lenguaje.

Finalmente se han diseñado mejoras para la gestión del tráfico y liberar a los ordenadores de la carga de todo el procesado de la información, distribuyéndola entre varios ordenadores para mejorar así la eficiencia.

Capítulo 2

RTFM (REALTIME TRAFFIC FLOW MEASUREMENT)

2.1 INTRODUCCIÓN

RTFM [1] [9] [11] incorpora una nueva filosofía para la medición de flujo de tráfico en tiempo real, que se comenzó a desarrollar a principios del 1992 por un grupo de trabajo llamado "Internet Acounting Working Group" dedicado en esos tiempos a la medición de tráfico para su posterior estudio. Todo esto lo podemos ver en la referencia [19].

Este grupo de trabajo comenzó a trabajar en 1991 en el área de administración de redes y en sus comienzos ya empiezan a tener los primeros problemas con respecto a los modelos de flujo de tráfico y en su diseño para medirlos.

En 1993 aparece como libre distribución la primera aplicación RTFM desarrollada en la universidad de Auckland por el profesor Nevil Browlee, esta aplicación se llama NeTraMet, la cual se desarrollara mas en detalle en el siguiente capitulo. En 1996 desaparece el grupo IAWG y pasa a llamarse RTFM Working Group [19]. Sobre 1997 este grupo de trabajo empezó a publicar los primeros RFC's relacionados con RTFM [10]-[17]. Los objetivos de RTFMWG en su primer RFC son los siguientes:

- El modelo de gestión de la filosofía RTFM debía ser compatible con todas las capas OSI [3] y compatible con cualquier modelo software y hardware existente en el mercado.
- Esta filosofía debía de ser capaz de procesar todos los atributos de un flujo de tráfico.
- Los atributos de interés para un usuario debe ser simple para el usuario, de tal forma que al usuario le sea sencillo separar el tráfico de interés del resto.
- Generación de ficheros de registros (ficheros .log o históricos) ya que son interesantes desde el punto de vista del proveedor de servicios que los utilizara para la gestión de red.
- El entorno RTFM debería incluir una serie de filtros para capas superiores a IP que ayude a minimizar el tráfico que circula por la red.

2.2 ARQUITECTURA RTFM

El sistema de medida de flujo de tráfico es usado por el personal que maneja la red para administrarla y desarrollarla. Esto suministra herramientas para medir y entender los flujos de tráfico de la red. Esta información es usada para muchos propósitos que mencionaremos en los siguientes apartados.

2.2.1 Definición de flujo

Antes de empezar a desarrollar la explicación de la arquitectura RTFM [11] debemos comprender qué es un flujo y cual es nuestro interés en ellos.

Un flujo es una entidad artificial que es comparable a una llamada o a una conexión entre dos puntos, delimitada por un tiempo de inicio y otro de fin, que fue generada por una entidad responsable. El tiempo de comienzo queda almacenado en una tabla de un medidor como un valor fijo y cuyo tiempo final va creciendo conforme el medidor va modificando los valores de los atributos medidos en su tabla.

En términos prácticos, un flujo es una cadena de paquetes (cada uno de los cuales es completamente independiente del resto) pasando a través de una red entre dos puntos terminales (o que se estén enviando desde un solo punto) el cual ha sido resumido (filtrado) por un medidor de tráfico para su posterior estudio.

Los medidores de tráfico tienen como parte de su configuración un conjunto de reglas las cuales especifican que flujo y atributos de éste son interesantes.

2.2.1.1 Los atributos de los flujos

Cada medidor de trafico mantiene una tabla de “registro de flujos” para flujos “interesantes” observados por el medidor. Un registro de flujo recoge los valores de los atributos que el medidor considera interesante. Estos atributos pueden ser:

- Direcciones de origen y destino del flujo. Este distingue entre distintos tipos de protocolos y capas de red, de las cuales se extraen los paquetes que han sido observados.

- Primera y última vez que el paquete fue visto en el flujo, es decir, creación y última actividad de éste en el flujo.
- Contadores en paquetes y bytes para los distintos sentidos (origen→destino o destino→origen).
- Otros atributos de información generada por el medidor.

Una entidad responsable de un flujo está especificada por los valores de sus atributos de direcciones. Por ejemplo:

Si prestamos atención a paquetes IP con dirección origen 192.168.1.0/24 (red 1) solo contará los paquetes con dirección origen la red 1 y cualquier destino, pero si a parte se indica que los paquetes tengan como dirección IP destino 192.168.2.0/24 (red 2), solo serán contados los paquetes que vayan de la red 1 a la red 2.

Estos atributos responsables pueden incluir uno o más de los siguientes tipos:

- *Numero de interfaz* en la que el medidor mide el tráfico.
- *Direcciones adyacentes*, estas se refieren a la segunda capa de red en el modelo de referencia OSI [3], en el caso de una red Ethernet es la dirección MAC (Control de Acceso al Medio) también llamadas direcciones físicas, cuyo tamaño es de 6 octetos.
- *Las direcciones de extremos* identifican el origen y destino de una pareja de niveles en la capa OSI [3]. La forma de estas direcciones, dependerán del protocolo del nivel de red en uso, en la cual se están realizando las medidas de tráfico. En nuestro caso este protocolo es IP.
- *El protocolo de transporte* identifica el origen y destino de un paquete.

Los cuatro puntos anteriores hacen referencia a los cuatro primeros niveles del sistema del modelo de referencia OSI (nivel físico, enlace, red, transporte). Un registro de flujo almacena los valores de origen y destino para cada una de sus atributos con sus respectivas mascaras que indican que bits son utilizados y cuales son ignorados.

Uno de los rasgos más importantes de esta arquitectura, es que los atributos tienen el mismo significado para cualquier protocolo de la misma capa, es decir que las aplicaciones de análisis podrán utilizar el mismo formato para todos los protocolos que existan para la misma capa OSI.

2.3 Descripción de la arquitectura

La arquitectura RTFM [11] es una arquitectura distribuida en la que varios procesos remotos están ejecutándose independientemente el uno del otro, e interactúan entre ellos como se muestra en la figura siguiente:

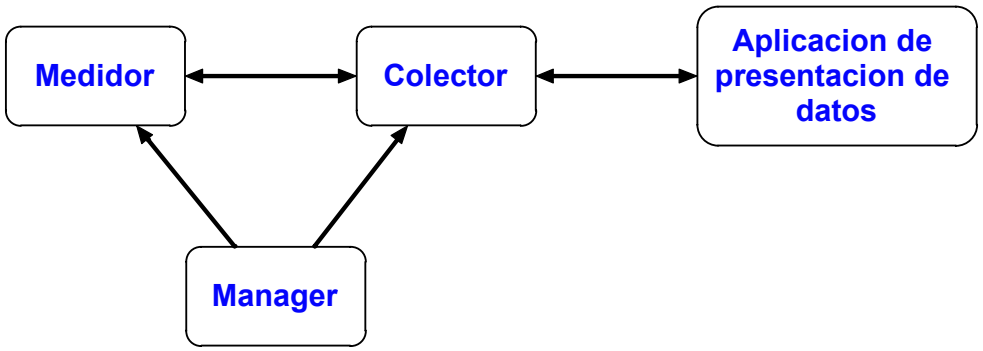


Figura 2-1.

El modelo de estudio de tráfico se basa fundamentalmente en el medidor de tráfico. Los medidores generan flujos de información a partir de los paquetes que pasan por el. Estos flujos son clasificados, almacenados y contados mediante un criterio de selección. Estos medidores son administrados por entidad responsable de la red que cargará los criterio de filtrado comentado anteriormente, este proceso se le denomina manager. Un último elemento clave en la filosofía RTFM es el colector que se encarga de leer la información recolectada por el medidor de forma asíncrona.

2.3.1 Medidores.

Los medidores son colocados en los puntos de medición determinados por el administrador de la red. Selectivamente el medidor guarda en su tabla registros la información de los paquetes que pasan, esta información guardada es seleccionada con respecto a la configuración que se le ha cargado. Esta información puede ser transformada o procesada antes de ser almacenada en la tabla de registros.

El medidor puede ser implementado de varias formas:

- En un pequeño ordenador conectado a una red LAN (el cual ve todos los paquetes que pasan a través de su interfaz) y en el cual esta corriendo un medidor.
- Un ordenador de altas prestaciones (WorkStation), con varias interfaces y un programa medidor ejecutándose. Este suministra múltiples puntos de medida.

- Un router o un switch que actúan igual que en punto anterior, pero reenviando los paquetes.

2.3.1.1 Estructura del medidor.

- Los paquetes que llegan al medidor entran al procesador de paquetes.
- El procesador de paquetes los pasa al PME (Packet Matching Engine) donde son clasificados.
- El PME es una maquina virtual en la que se encuentra corriendo un programa de clasificación de paquetes, comparando un patrón con la cabecera del paquete. Este programa es un fichero de reglas que es cargado por el administrador, y es invocado por el procesador de paquetes para decidir si los paquetes serán ignorados o clasificados.
- Algunos paquetes son clasificados, y otros son ignorados o descartados por el procesador de paquetes.
- Otros paquetes son emparejados por el PME, que devuelve un "FlowKey" describiendo el flujo al cual los paquetes pertenecen.
- El FlowKey es usado para colocar las entradas de los flujos en la tabla de flujos, cuando no existe entrada es porque el flujo es la primera vez que le medidor lo ve, y crea una nueva entrada. Los paquetes y bytes de entrada contados son actualizados.
- El medidor consta con una tabla de registro de flujo o base de datos, donde se incluirá una tabla de flujos. Esta tabla esta generada como un array de estructuras que contienen los atributos de los flujos, estas estructuras son rellenadas por el procesador de paquetes por la información de los paquetes clasificados.
- Posee un modulo de exportación de datos hacia los colectores, que será el encargado de pasar la tabla de flujos al colector de datos.

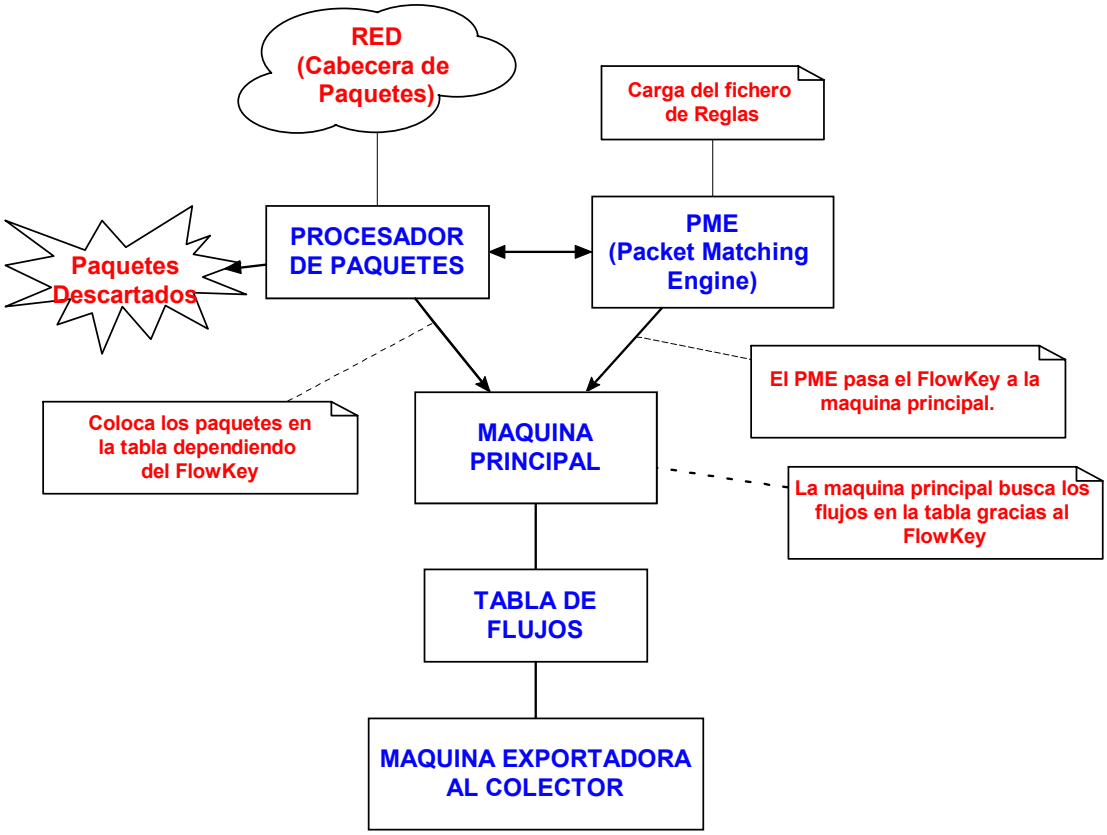


Figura 2-2.

2.3.1.2 Tablas de flujo.

Cada medidor mantiene una tabla de registros de flujo de tráfico, para los flujos vistos por el medidor. Un flujo contiene valores de atributos, incluyendo:

- Direcciones origen y destino de los flujos. Este atributo incluye tanto la dirección como la mascara asociada para los distintos niveles de una jerarquía de red.
- Primera y última vez que el paquete fue visto en el flujo.
- Contadores tanto para los paquetes de ida como los de vuelta desde un punto de red.
- Otros atributos que nos indican el estado del registro de flujos.

El estado de un registro de flujo puede ser:

- Inactivo: El registro de flujo no esta siendo usado por el medidor.
- Activo: El registro esta en uso y describe un flujo que pertenece a un conjunto que el medidor ha visto.

- Desocupado: En este momento el registro esta en uso, pero el flujo descrito no se ha vuelto a ver en ningún conjunto que ha visto el medidor, en un periodo de tiempo especificado por este.

2.3.1.3 Procesamiento de los paquetes.

Cada paquete recibido por el medidor es procesado de la siguiente forma:

- Extrae los valores de los atributos de la cabecera de los paquetes y los usa para crear un MATCH KEY. El MATCH KEY del paquete contiene el número de interfaz, direcciones origen y destino, etc. Pero solo contiene los valores de los atributos, no contiene las mascaras para cada uno de ellos.
- Empareja este MATCH KEY de los paquetes con los que aparece en el patrón definido en las reglas. Esta comparación genera un código que identifica al flujo llamado FLOWKEY y que es devuelto al procesador de paquetes.

Este FLOW KEY es utilizado para encontrar el registro de flujo en la tabla; si este registro no existe, será creado cuando llegue un flujo para ese registro.

En la Figura 2-3 se describe el algoritmo de funcionamiento del PME con cada paquete que le llega al medidor. Los flujos atraviesan el diagrama de forma descendente.

Hay varios casos a considerar:

- Los paquetes son ignorados.
- Los paquetes coinciden en ambas direcciones.
- Los paquetes emparejados en una sola dirección.

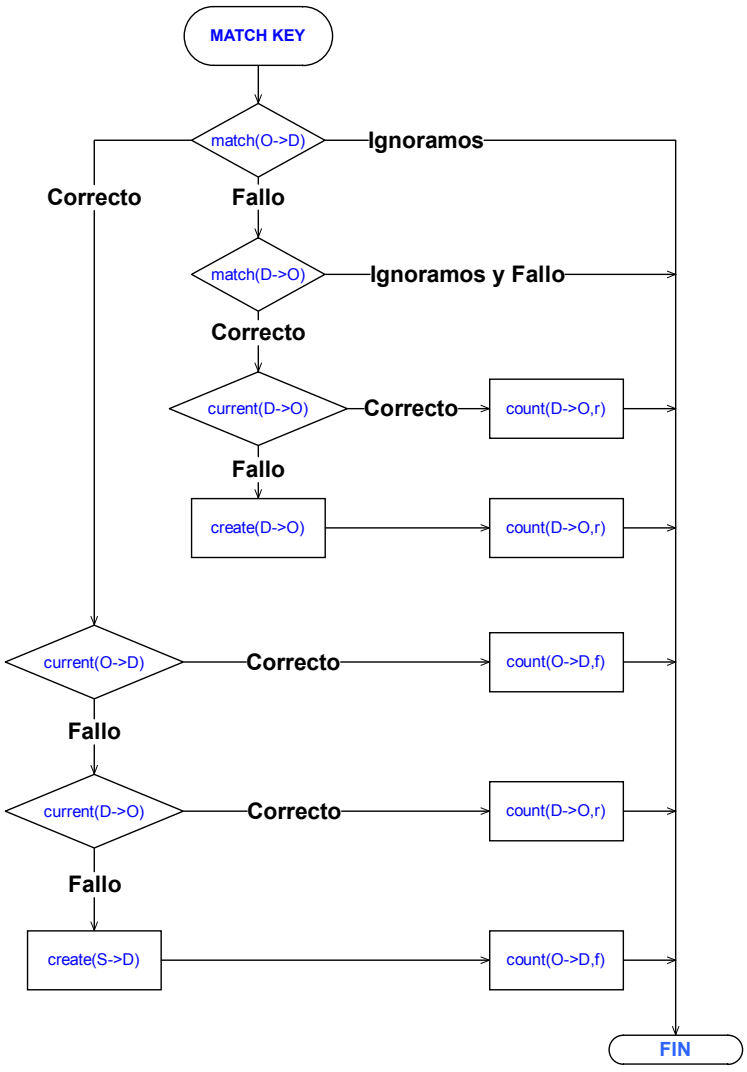


Figura 2-3.

El algoritmo usa cuatro funciones:

- Match(origen -> destino): Implementa el PME, este usa el conjunto de reglas para emparejar los valores de los atributos, en el match key del medidor. Match devuelve tres posibles resultados:
    - Ignorado: Lo que significa que el flujo fue emparejado, pero este no fue contado.
    - Fallo: Significa que el paquete no fue emparejado, aunque intercambiemos el origen y el destino.
    - Correcto: Significa que el paquete se emparejó, y por lo tanto se ha contado.

- Current(origen->destino): Devuelve éxito si existe un registro en la tabla de flujo, y falla en otro caso.
- Create(origen->destino): Añade el flujo a la tabla.
- Count(origen->destino, f o r): Incrementa los contadores en uno u otro sentido dependiendo de la opción f o r.

2.3.2 Colectores.

Un colector usa los datos que son acumulados por un medidor, estos datos son recolectados a intervalos regulares de tiempo por el colector (este tiempo es especificado por el administrador). La recolección de datos es registrada en un fichero llamado *Fichero de Flujo de Datos*.

- El colector identifica los flujos en el registro de flujos del medidor de la siguiente manera: Una vez que el paquete ha sido clasificado y preparado por el medidor para ser contado, este se añade al registro de flujo. Este registro de flujo tiene un formato flexible y algunos atributos que no son necesarios pueden ser omitidos. Cada flujo que entra tabla presenta un único identificador, indiferente al resto de los atributos.
- Fichero de flujo: La recolección de datos será almacenada en un fichero de flujo único para cada medidor. Un colector compondrá un fichero del registro mediante la recolección regular de flujo de datos desde el medidor, usando sus datos para construir un registro de uso (Figura 2-4) y añadiéndolo al final de este fichero. El registro de uso contiene la siguiente información:

Identificador de Registro: *Meter ID* *Timestamp* *Collection Rules ID*	
Identificador de Flujo: *Ardes List* *Subscriber ID* *Attributer (Opcional)*	Contadores: *Packet Count* *Byte Count* *Flow Star / Stop Time*

Figura 2-4.

- Los contenidos de este registro de uso son la razón de ser del sistema. La precisión, la fiabilidad y la seguridad son los temas primarios entre el intercambio de información entre el medidor y el colector. Sin embargo pueden producirse errores en la red y algunos paquetes se pueden perder, por los que necesita de algunos mecanismos para

asegurar que la información es transmitida correctamente. La información del flujo es movida desde un medidor a un colector a través de un protocolo de intercambio de información entre ellos, este protocolo es SNMP explicado la referencia [20]. Este puede llevarse a cabo de varias maneras; moviendo los datos de los atributos individualmente, moviendo flujos completos o moviendo las tablas de flujo completas. La referencia [13] nos proporciona documentación detallada sobre este tema.

2.3.3 Administradores o manager.

Un administrador de medidas de tráfico es una aplicación que configura entidades y controles del medidor y entidades del colector. Este utiliza datos de configuración para determinar la configuración apropiada para cada medidor y una correcta operación para cada colector. Esto lo realiza con las opciones que describimos a continuación:

- *Carga el fichero de reglas:* Los medidores pueden almacenar este conjunto de reglas [8]. El medidor lleva un conjunto de reglas por defecto y que no pueden ser modificadas. En cualquier momento el medidor puede alternar si lo necesita entre los dos conjuntos de reglas. El manager utiliza algún protocolo de transferencia de ficheros para cargar el conjunto de reglas.
- *Conmuta entre las reglas que tiene cargadas:* El manager puede indicar al medidor que conjunto de reglas debe tener arrancadas, este puede conmutar entre las reglas por defecto o las que el administrador le ha cargado.
- *Set High Water Mark:* Es el valor interpretado por el medidor que le dice cuando debe pasar de un conjunto de reglas a otro. También le dice si debe cambiar la granularidad (parámetro asociado al sistema que, expresa el nivel de detalle exigido a las mediciones tomadas) de los flujos.

2.3.4 Aplicaciones de análisis.

Una aplicación de análisis procesa los datos de uso (fichero creado por el colector mediante los registros de uso) así como suministra información que es usada para ingeniería de redes y propósitos de administración. Un ejemplo de aplicación de análisis lo incorpora la plataforma NeTraMet, este es la aplicación NiFty. Otra aplicación de análisis muy usada es MRTG (Multi Router Traffic Grapher) como se puede ver en la Figura 2-5.

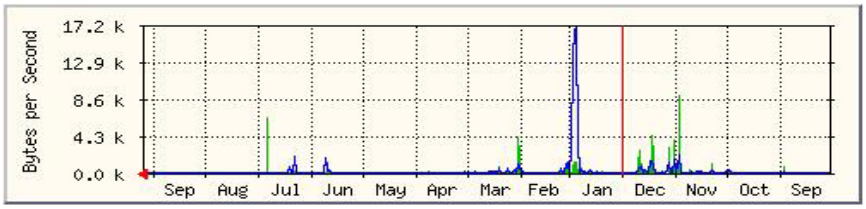


Figura 2-5.

2.4 Interacción entre los elementos de la arquitectura.

Estos elementos de la arquitectura de RTFM interactúan entre ellos mandándose mensajes y datos de configuración:

2.4.1 Interacción entre el medidor y el colector:

Un medidor mantiene los datos de uso en un array de registros de flujo de datos conocido como TABLA DE FLUJO. Un colector puede recolectar los datos de la siguiente manera. Por ejemplo, puede cargar una copia de la tabla de flujo completa usando un protocolo de transferencia de ficheros (FTP). El colector no necesita leer completamente los registros de los datos de flujo, un subconjunto de estos atributos puede ser suficiente.

Un colector puede recolectar datos de más de un medidor. Estos datos pueden ser recolectados en cualquier momento, por lo que la recolección no es sincronizada.

2.4.2 Interacción entre el medidor y el manager:

Los administradores (managers) son los responsables de configurar uno o más medidores. Un medidor puede ser controlado por un solo manager. La configuración de los medidores incluye información tal como:

1. Especificación del flujo: Que flujos de tráfico van a ser medidos y como van a ser agregados, y cualquier dato que requiere el medidor para procesar cada flujo que va a ser medido.
2. Control de parámetros del medidor: El máximo tamaño de su tabla de flujo, el tiempo de inactividad de los flujos.

3. Porcentaje de muestreo: Frecuencia de observación de cada paquete. A veces puede ser necesario usar estrategias de prueba para observar solo algunos de los paquetes. Algunos de estos algoritmos de muestreo no son aconsejados por la arquitectura, éstos algoritmos deberán ser validados estadísticamente antes de ser usados.

2.4.3 Interacción entre el manager y el colector:

Un manager o aplicación de administración es responsable de controlar uno a o más colectores, y cada colector solo puede ser controlado por un solo administrador. Un colector necesita saber al menos el seguimiento de cada medidor, para este fin requiere:

1. Que el medidor sea de identidad única. Por ejemplo: su nombre de red o dirección.
2. La frecuencia con que los datos de uso son recolectados por el medidor.
3. Cuales de los registros de flujos están siendo recolectados.
4. Cuales de los valores de los atributos están siendo recolectados para el registro de flujo requerido

Capítulo 3

PLATAFORMA NETRAMET

3.1 INTRODUCCIÓN

NeTraMet es la primera implementación de un medidor de tráfico en tiempo real (RTFM) [8] [9]. Éste salió por primera vez con la versión 2.0 en 1993 y desde entonces como software de libre distribución. NeTraMet fue desarrollado por el profesor Nevil Brownlee (integrante del Departamento de Tecnología de Sistemas y Servicios Informáticos de la Universidad de Auckland). Esta plataforma la podemos encontrar en el FTP de la universidad de Auckland [18].

El medidor NeTraMet almacena las medidas de flujo de datos en la memoria, y suministra un agente SNMP. La plataforma NeTraMet incluye el paquete NeMaC, que es la combinación de un administrador o manager y colector que puede manejar un número arbitrario de medidores. Cada uno de estos medidores puede usar su propio conjunto de reglas (cargadas por el manager) con el que recolectara su propio flujo de datos en intervalos especificados. La distribución de NeTraMet además incluye varias aplicaciones de análisis rudimentarias, proporcionando a los usuarios graficas simples de los ficheros de flujo de datos generados por NeMaC (fd_filter y fd_extract) y un monitor en tiempo real de los flujos de un medidor remoto (nm_rc y nifty). Pero lo más importante y lo que ha hecho que elijamos esta plataforma es que a partir de la versión 4.2 se incorpora a la plataforma el medidor NetFlowMet que se utiliza para medir el flujo que nos puede exportar un router Cisco por medio de NetFlow [2] y que este flujo pueda ser tratado de la misma forma que con NeTraMet.

Desde la primera edición de NeTraMet, han estado ampliándolo y simplificándolo. Produciendo cambios significativos por lo que han incluido mejoras para especificar flujos de tráfico (mejoras en la estructura de control del PME) y computo de atributos.

NeTraMet es muy usado para entender exactamente el tráfico que esta corriendo en una red. Todo esto gracias a que los medidores realizan reducciones considerables del flujo de tráfico, ayudado por los conjuntos de reglas los cuales reducen significativamente el volumen de datos para ser leídos por el colector. Esta característica hace especialmente particular a NeTraMet en redes muy grandes, y con mucha carga de tráfico.

También suministra datos para un posterior análisis, por lo que puede ser usado para monitorización de tráfico de redes en tiempo real.

3.1.1 Historia y evolución de la plataforma NeTraMet.

Sobre 1992 la Universidad de Auckland necesitaba desarrollar un sistema para reducir los costes de su tráfico de Internet. En Marzo de ese año, Nevil acudió a una conferencia de Internet Accounting Working Group en San Diego. Este quedo muy sorprendido de la arquitectura de medida de trafico en tiempo real desarrollada por este grupo de trabajo.

Durante ese año, Nevil Brownlee desarrollo el primer prototipo de sistema de medida usando árboles binarios balanceados para el almacenamiento del flujo de tráfico. Este primer prototipo fue presentado en Washington IETF [1] en Noviembre de 1992. Este Prototipo tenía el problema que no recuperaba la memoria que utilizaban los viejos flujos, lo que lo haría poco estable y a esto había que añadir lo poco eficiente que era manejar árboles balanceados a la hora de hacer búsquedas en ellos.

Nevil Brownlee siguió en el desarrollo de nuevos sistemas de calidad, y en Octubre de 1993 sacó una nueva versión, llamada NeTraMet (Network Traffic Meter) en la que utilizaba tablas para almacenar la información de los flujos. Desde entonces ha seguido trabajando para mejorar NeTraMet.

Desde que apareció el primer prototipo de NeTraMet este ha ido mejorando y se le han ido añadiendo nuevas funcionalidades que a continuación se expondrá en orden cronológico:

V 1.0	**Nov 92**	• Primer prototipo del medidor usando el sistema de árbol binario balanceado. Es presentado al Washington IETF.
V 2.0	**20 Oct 93**	• Primera publicación de NeTraMet y NeMaC con sus respectivos manuales de usuarios y código abierto.
V 2.1	**14 Jan 94**	• Implementa subrutinas nuevas en las tablas de reglas que hace más fácil manejar largo número de redes. • Netramet comprende el protocolo CLNS (**snmpCLNSDomain** se usa cuando CLTS se ejecuta en servicios de red no orientados a conexión de OSI) [REF SNMP] • Los paquetes para protocolos que no entiende Netramet pueden ser contados como paquetes desconocido en el atributo PeerType • Ethernet II y SNAP encapsulados por IPX ahora son reconocidos. • Direcciones IPX de 10 bytes pueden ser usadas en vez de números de red de 4 bytes. • Las variables de entorno MIB son cambiadas a MIBTXT.

V 2.2	**19 Jul 94**	• Se incluye fd_filter y fd_extract en el directorio del manager como programas útiles para el post-filtrado de los ficheros de flujo. • Los puertos de Netramet y NeMaC para Solaris usando streams/dlpi en lugar de NIT (New Information Technology) para observar las interfaces Ethernet. • Binarios para Solaris y SUN suministrados por un ftp anónimo. • Implementa tasa de muestreo para variables MIB. Permitiendo procesar solo 1 de cada n paquetes. • Netramet ahora cuenta tanto paquetes enviados como recibidos.
V 2.3	**25 Nov 94**	• NeMaC ahora utiliza los nombres de los flujos tal como aparecen en el medidor MIB. • Gopher (puerto 70) y WWW (puerto 80) se han añadido a la lista de puertos IP de NeMaC. • Si NeMaC advierte que un medidor ha sido reinicializado, éste volverá a cargar automáticamente el fichero de reglas en el medidor. Este chequeo está hecho antes de la recolección del flujo de datos. • NeMaC ahora permite diferentes intervalos de recolección y mantenimiento de vida de cada medidor. Esto está implementado para permitir la opciones –c y –k que aparecen en el fichero de configuración de NeMaC y usando una cola de eventos para ordenar las actividades de los medidores. • Se ha implementado también un mecanismo para cerrar y reabrir fichero de flujo de datos. De esta forma podemos renombrar el fichero para realizar históricos. • Se han corregido reglas en la revisión sintáctica de los ficheros de reglas. También se han corregido errores en los ficheros de fd_filter y fd_extract. • El manejo de la memoria de netramet ha sido mejorada, se mejora la recolección de basura y esta actúa cuando un nuevo flujo necesita espacio.
V 3.1	**16 Feb 95**	• Rescribir y simplificar la MIB significa que los medidores anteriores no se ejecutaran con el NeMaC 3.1 y los medidores 3.1 no se ejecutaran con los antiguos NeMaC, es decir que los medidores y manager tienen que actualizarse a la misma versión 3.1. • Extiende y simplifica el emparejamiento de las reglas. Los valores de los atributos pueden ser sacado del paquete, por tanto agrega y cuenta flujos que no han sido largos. • Seis nuevos atributos son implementados. SourceClass, DestClass, FlowClass y SourceKind, DestKind, FlowKind permitiendo a un medidor pasar información durante el emparejamiento de paquetes introduce en el fichero de flujo. • NeMaC te permite incluir un fichero de reglas dentro de otro fichero de reglas. • Se implementa en el medidor un conjunto de reglas de emergencia que el medidor cambiara a este conjunto de reglas cuando el % del flujo activo obtenido supera HighWaterMark. • El tiempo de recolección es sincronizado por defecto. • Se han reparado un error en las tablas de reglas en las que tablas con más de 1350 reglas no trabajan correctamente en el medidor. • Un error en el código para optimizar testeando a un grupo de reglas podría a veces causar emparejamiento de paquetes que no deberían suceder. Esto ha sido corregido. • Los ficheros de reglas necesitaran ser convertidos de la forma antigua (de las versiones 2.x) a la nueva versión.
V 3.2	**8 Jun 95**	• El medidor de Netramet se pone al día para usar las librerías Libpcap para obtener las cabeceras de los paquetes. Estas librerías están disponibles en: ftp://ftp.ee.lbl.gov/libpcap-*.tar.Z • Libpcap soporta interfaces FDDI además de ethernet
V 3.3	**8 Nov 95**	• Se introduce en la plataforma el programa nm_rc que es una consola remota para NeTraMet. Este se encuentra en el directorio del manager combinado con fd_filter y NeMaC para suministrar una presentación del flujo de datos de un medidor colocado dentro de la red. • Aparecen nuevas opciones para NeMaC, las cuales detallaremos de forma mas precisa en siguientes apartados. • Se producen también cambien en las opciones de NeTraMet, por ejemplo que ya no se necesitan espacios para separar sus argumentos, -ile0, -ieth0. Otro cambie es que solo se puede especificar una comunidad de lectores.
V 3.4	**8 Aug 96**	• Aparece el programa Nifty que es un analizador de flujo, presentado por RTFM WG al IETF como 'NetFlow' renombrado para evitar concisión con 'Net Flor Switching'

- NeTraMet puede monitorizar hasta cuatro interfaces en lugar de una, especificando cada una con la opción –iXXX
- El medidor interpreta estadísticas que han sido implementadas por el medidor de unix. En particular, aps y mps dan la media y el máximo numero de paquetes por segundo, mientras que api y mpi dan la media y el mínimo porcentaje de tiempo que el procesador esta desocupado, en intervalos de un segundo.
- NeTraMet ha sido reestructurado de manera que simplifique el código para el emparejamiento de paquetes.
- Un problema conocido a partir de esta versión es que si ejecutas un NeMaC, debes asegurarte que no hay otro ejecutándose antes intentando escribir en el mismo medidor, ya que el medidor se confundiría.

V 4.1b4 **22 May 97**

- NeTraMet y su manager y colector empiezan a usar SNMPv2 en lugar de SNMPv1, los cuales implementaran el Meter MIB que se explica en el RFC2064.
- Los manager de la V4 trabajaran apropiadamente con los medidores de la V3, pero al contrario no funcionara. Para cambiar a la versión 4, primero deberías cambiar los managers y después los medidores.
- Hay dos cambios en el formato de los registros de los ficheros de flujo, e la fecha ahora se usan 4 dígitos para los años, y se utiliza un valor entero para el atributo PeerType.

V 4.1b10 **30 Jun 97**

- Se añade un atributo matchingStoD es configurado por el PME. Su valor es 1 si el paquete esta siendo emparejado con su atributo dirección en orden, y 0 si los paquetes están siendo emparejados con sus direcciones intercambiadas. Para mas detalle mirar [RFC2063].

V 4.1b13 **17 Jul 97**

- Un nuevo parámetro, 'owner name' ha sido añadido a este programa. Este es un identificador alfanumérico hasta 16 caracteres de largo. Este atributo es usado para identificar conjunto de reglas, esto es necesario cuando el medidor es arrancado con más de una regla.

V 4.1b14 **9 Sep 97**

- Se encuentran errores en nmc_c64.c que produce extremadamente grandes cuentas en el fichero de flujo de datos arrancando NeMaC en Linux.
- Estos cambios fueron implementados usando la constante WORDS_BIGENDIAN en el auto configuración. El método recomendado para instalar NeTraMet es usar la autoconfiguración.

V 4.1b15 **22 Sep 97**

- Usa WORDS_BIGENDIAN Y SIZEOF_LONG definidos para implementar código nativo para conseguir poner los contadores a 64 bits.

V 4.1 **24 Nov 97**

- Se documentan los ficheros ahora en formato PDF y se encuentran en la pagina http://www.aucland.ac.nz/Accounting

V 4.2b2 **11 May 98**

- Un nuevo medidor **NetFlowMet** ha sido añadido a la distribución, este captura flujos de un router cisco y los usa para construir las tablas de flujo.
- NetFlowMet suministra 5 nuevos atributos que pueden ser usado en las reglas:
    1. MeterID (8 bits, mascara 255)
    2. SourceAns, DestAns (16 bits, mascara 255.255): Numero de sistemas autónomos para origen y destino de redes. Estos pueden especificar cuando se empieza a exportar flujo desde el router.
    3. SourcePrefix, DestPrefix (8 bits, mascara 255): Longitud de la mascara para direcciones IP de origen y destino.
- Se producen cambios en NeMaC; cuando este es cerrado correctamente, éste cierra todas las tareas con todos los medidores.
- Se añade un registro #EndData al final de de cada fichero de flujo. Esto permite el procesado en tiempo real de flujos de datos, sin este atributo, uno tendría que esperar hasta la siguiente muestra.
- Cambios en el comportamiento del manager cuando un medidor falla, y el manager no recibe respuesta a los intentos de arrancarlos, y éste pasa a ignorar al medidor.

V 4.2b3	**22 May 98**	• Implementa un mejor algoritmo de encriptación para las tablas de flujo utilizando múltiples bytes de direcciones de transporte y de red.
V 4.2b4	**3 Jun 98**	• Usa el atributo LastTime en vez de sysUptime para obtener los tiempos de medición en NeMaC, nm_rc y nifty. • Se encuentran errores en las librerías SNMP que causó paradas en algunos paquetes de SNMP.
V 4.2	**5 Aug 98**	• Se añade a la instalación el programa SRL Compiler, este es programa compilador de SRL (Simple Ruleset Language). Este lenguaje esta documentado en la pagina de NeTraMet, y será explicado en detalle en el siguiente capitulo.
V 4.2.1	**2 Oct 98**	• Srl que arranca con una declaración especifica, ahora arranca con reglas GotoAct, Next. Sin esto no trabaja. • Fd_extraxt ahora maneja contadores de atributos de 64 bits. • Un error en la memoria ha sido arreglado en el fichero snmpapi.c de SNMP.
V 4.3b1	**23 Oct 98**	• Se producen mejoras en el medidor; El medidor asigna tanta memoria como sea posible cuando arranca, así como minimiza los costos operativos en referente a memoria. El espacio para las reglas es asignado desde el comienzo, con un máximo de 2000 reglas por defecto. Con la opción –u podemos cambiar este tamaño.
V 4.3b2	**12 Nov 98**	• Se le añade un soporte para FreeBSD, se cambian los ficheros de origen, así como aquellos que no utilizan malloc.h
V 4.3b5	**26 Nov 98**	• Se corrige un error en el compilador SRL, el cual no estaba distinguiendo entre "save sourcetransaddress y save sourcetransaddress = 0"
V 4.3b10	**26 May 99**	• Soporta IPv6 solamente cambiando en el fichero de configuración: AC_DEFINE(V6,0) a AC_DEFINE(V6,1) antes de arrancar la configuración. • El compilador SRL permite direcciones IPv6, esto esta especificado en el RFC2373.
V 4.3	**30 Sep 99**	• Añade soporte para la configuración en sistemas unix True64. Esto corrige varios errores introducidos desde la versión 4.2 • Se añade la posibilidad de utilizar NeTraMet y NeMaC como demonios utilizando la opción –D.

3.2 COMPONENTES BÁSICOS DE LA PLATAFORMA NETRAMET.

La plataforma NeTraMet [8] posee elementos básicos similares a los de la filosofía RTFM explicada en el capitulo anterior:

3.2.1 Administrador y colector:

Al igual que en la filosofía RTFM existe un administrador que gestione los medidores, y un colector que recogerá los flujos de trafico que el medidor envía. La diferencia reside en la

combinación de estos dos elementos en una sola aplicación. Esta aplicación se llama NeMaC y realiza las funciones de colector y de manager como hemos mencionado.

NeMaC: Este programa es una combinación de un colector y un manager. Este programa es necesario para suministrar el control al medidor NeTraMet así como hacer la configuración inicial y una monitorización útil y efectiva. Si NeMaC controla solo un medidor los valores que hay que pasarle a este medidor se le indica la aplicación NeMaC por la línea de comando mientras que si NeMaC controlara varios medidores los valores de configuración se les pasaría por la línea de comandos los parámetros comunes y el resto en un fichero de configuración.

Un ejemplo de fichero de configuración de varios medidores es el siguiente:

Línea de comando: ./NeMaC –f fichero_configuración

Fichero_configuración:

```
-c 900 – p –r id.rules
        212.128.24.41 escritores1
        212.128.24.50 escritores2
-c300 212.128.20.252 escritores3
```

Mediante este fichero de configuración podemos controlar con nuestro NeMaC tres medidores con direcciones 212.128.24.41, 212.128.24.50 y 212.128.20.252, cada uno de ellos pertenece a una comunidad de escritores diferentes escritores1, escritores2 y escritores3 respectivamente. El tiempo de recolección de flujo de datos de los medidores por el colector lo marcamos con la opción "-c", con la opción "-r" cargamos con el NeMaC el fichero de reglas al medidor. El conjunto de opciones de configuración de la plataforma NeTraMet se explicara con más detalle mas adelante en este capítulo.

- *Colector:* El colector recoge como hemos dicho en el tema anterior los flujos medidos por el medidor y los almacena en un fichero llamado "fichero de flujo". No es tan sencillo, primero el colector se debe de suscribir al medidor y este creara en la MIB [21] una nueva entrada para este colector donde almacenara el momento de comienzo y distintos atributos y parámetros que se irán actualizando cada vez que el colector consulte estos datos. El colector crea una tabla donde almacenara esos flujos interesantes leídos por el medidor los cuales actualizan mediante peticiones SNMP. La actualización de estos atributos la realiza solamente leyendo los atributos que han sido mas recientemente actualizados. Una vez que el colector ha hecho todo esto, crea un fichero llamado fichero de flujo como se muestra en la siguiente Figura 3-1.

```
##NeTraMet v4.3:  -c5 -r id.rules  127.0.0.1 eth0  10000 flows  starting at
14:34:01 Wed  8 May 2003

#Format: firsttime flowindex flowruleset sourcepeeraddress destpeeraddress
fromoctets  tooctets  sourceinterface  destinterface  sourcetransaddress
desttransaddress sourcetranstype desttranstype

#Time: 14:34:01 Wed  8 May 2003 127.0.0.1 Flows from 0 to 26873

#Ruleset: 14  2 id.rules  NeMaC

#EndData: 127.0.0.1

#Time: 14:34:05 Wed  8 May 2003 127.0.0.1 Flows from 26872 to 27278

#EndData: 127.0.0.1

#Time: 14:34:10 Wed  8 May 2003 127.0.0.1 Flows from 27277 to 27783

27361 65 14 212.128.20.252 192.168.1.20 0 269 1 1 53 1345 17 17

27426 66 14 130.216.191.125 192.168.1.20 0 368 1 1 80 1347 6 6

27747 68 14 62.81.133.153 192.168.1.20 0 77336 1 1 80 1351 6 6

27751 69 14 62.81.133.153 192.168.1.20 0 33430 1 1 80 1353 6 6

#EndData: 127.0.0.1
```

Figura 3-1.

Aquí se encuentra toda la información que ha pasado por el router que tiene instalado el medidor en los últimos 5 segundos y se estructura en columnas definidas en la segunda línea comentada "#Format:", todos los atributos que aparecen en esta línea se explicarán con mas detalle en el siguiente capitulo. En la primera línea comentada nos muestra la configuración del medidor indicándonos el tiempo de recolección del colector, las reglas que se le han cargado, dirección IP del medidor (127.0.0.1) y la interfaz en la que estamos midiendo el tráfico.

- *Manager:* Como hemos estado explicando se encarga de gestionar los medidores y actuar cuando uno de ellos se cae, volviéndole a cargar las reglas cuando se recupera la conexión con él. También elegirá que conjunto de reglas utilizara el medidor, alternando entre las que el medidor lleva por defecto y las que éste le carga.

3.2.2 Medidores:

Hasta la versión 4.4 han llegado dos clases de medidores NeTraMet y NetFlowMet:

- NeTraMet: Es un programa que implementa un medidor de trafico RTFM el cual almacena las medidas de flujo de datos en la memoria y suministra un agente SNMP, el cual facilita estas medidas al colector.
- NetFlowMet: Hace algún tiempo Cisco implementó un esquema de conmutadores llamado NetFlow [2]. NetFlow almacena los flujos de datos unidireccionales en el router y los usa para mejorar el funcionamiento del enrutado. Los datos NetFlow pueden ser exportados desde el router Cisco vía Internet mediante paquetes UDP. Cisco

suministra esta capacidad en sus routers de gama alta, pero en un futuro lo podrá incluir en el resto. Para más información sobre cisco NetFlow se puede consultar la referencia [2]. NetFlowMet es un programa muy parecido a NeTraMet, pero en vez de observar paquetes directamente de las interfaces, NetFlowMet coge los flujos de datos exportados por cisco y son usados para construir tablas de flujo RTFM. Esto da un numero de ventajas:

- Cada entrada de datos NetFlow se resume en muchos paquetes, y cada paquete contiene muchas entradas. Esto reduce la cantidad de trabajo de que NetFlowMet puede hacer para cada flujo.
- NetFlow puede recolectar datos desde algunos o todos los puertos de un router de alta velocidad. Esto hace posible medir interfaces serie de alta velocidad.

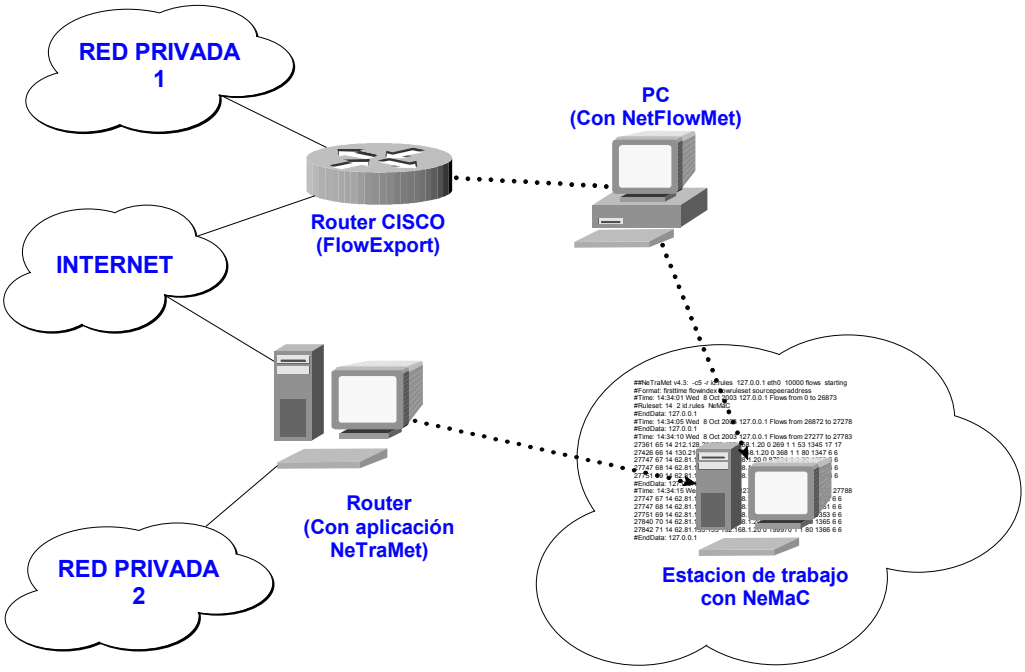


Figura 3-2.

3.3 CONFIGURACION DE NETFLOWMET, NETRAMET Y NEMAC.

Una vez decidido que vamos a utilizar la plataforma NeTraMet pasamos a instalarla, en nuestro caso será en el sistema operativo Unix. De la página de NeTraMet [8] encontramos todo lo necesario para instalarla en los sistemas operativos UNIX y DOS.

El fichero que debemos de descargar para instalar los componentes es **NeTraMet43.tar.gz,** este fichero lo descomprimimos mediante el comando 'tar' de la siguiente manera:

tar –zxvf NeTraMet43.tar.gz

En éste momento se nos crea un directorio con el mismo nombre, que va a contener las librerías y archivos necesarios para que pueda funcionar la aplicación, como los ficheros MIB y los ejecutables que se encuentran en el directorio src.

Una vez descomprimidos los ficheros básicos procedemos a la instalación de ellos. Para la instalación solamente ejecutamos los comandos:

./configure ➔ *Este comando prepara las librerías para enlazarlas y así optimizar la plataforma para nuestro hardware.*

make ➔ *Está instrucción es para compilar el software, ésta coge la información del makefile que esta incluido en todos los software para compilar todos los .c .h ...*

Ya tenemos la base de la plataforma NeTraMet instalada en nuestro equipo, ahora debemos instalar el contenido del fichero **Linux43.tar** que contiene los ejecutables necesarios para poder trabajar en Linux. Para descomprimir este fichero utilizamos la línea de comando:

tar –xvf Linux43.tar

En este momento se instalarán en el directorio NeTraMet43 los ejecutables para Linux, también podemos encontrar los ficheros para Solaris, iris y alpha. **Solaris43.tar.gz, Irix43.tar.gz y Alpha43.tar.gz.**

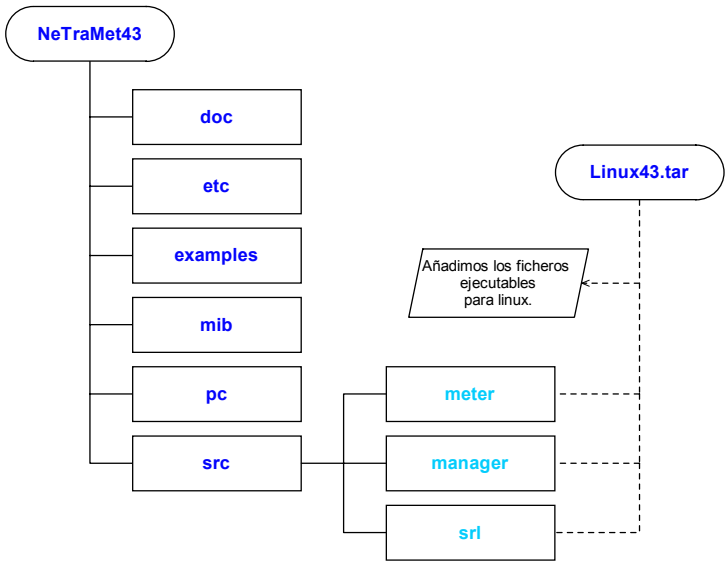


Figura 3-3.

Mediante la Figura 3-3 podemos entender la estructura de directorios que existirá en este momento en nuestro ordenador. A partir de este momento podemos empezar a utilizar la aplicación.

En el directorio "meter" encontramos los dos medidores que podemos utilizar; NetFlowMet y NeTraMet, en el directorio "manager" encontramos nuestro medidor y colector "NeMaC" y en el directorio "./src/srl" encontramos el compilador "srl" para generar ficheros de reglas .rules a partir de los ".srl" que hemos creado nosotros.

3.3.1 Configuración NeMaC

Ahora vamos a especificar las opciones que nos da NeMaC para su configuración. Como sabemos, NeMaC es tanto el administrador de los medidores como su colector a la vez, por lo que nos encontramos opciones tanto para una forma de trabajar como para la otra. Las opciones son las siguientes:

../manager/NeMaC [Opciones]... [IP o host-name del medidor] [Nombre de la comunidad]

- Opción "-b fichero_mib": Lee de un fichero mib alternativo indicado por el usuario. Para nuestro objetivo solamente utilizamos "mib.txt".

- Opción "-c nnn": Le pasamos al colector los intervalos de recolección. Si ponemos como intervalo de recolección cero, el administrador cargara el fichero de reglas y terminara la ejecución sin recolectar nada.
- Opción "-e fichero_reglas": Indicamos al medidor que lea las reglas de un fichero de reglas RLE.
- Opción "-f fichero_configuración": Lee el fichero de configuración. Este fichero se utiliza para la configuración de varios medidores, y así poder recolectar su flujo de datos.
- Opción "-g sss": Especifica los intervalos de recolección de basura.
- Opción "-i sss": Especificamos el tiempo de inactividad para los flujos asociados.
- Opción "-k nnn": Indicamos el tiempo al medidor para que haga comprobaciones si un medidor esta o no activo. Si el medidor se cae, y vuelve a ejecutarse, el administrador le volverá a cargar las reglas.
- Opción "-l": Muestra los ficheros .rules que el medidor esta procesando.
- Opción "-m ppp": Especificamos a NeMaC que puerto UDP de comunicaciones estamos utilizando para la comunicación.
- Opción "-p": Esta opción sirve para que cada vez q nos lleguen datos del medidor, abramos el fichero, añadimos los datos y cerramos el fichero. De esta forma podemos renombrar el fichero para hacer históricos de cada día.
- Opción "-r fichero_reglas": Con esta opción le indicamos al administrador el fichero de reglas a utilizar.
- Opción "-s": Chequea la sintaxis de las reglas y pero no carga las reglas en el medidor.
- Opción "-t": Sirve para suministrarnos mas información sobre el funcionamiento de NeMaC.
- Opción "-v": Mediante esta opción, el colector nos muestra por pantalla, información de los flujos que le llegan y si ha cargado bien el fichero de reglas.
- Opción "-x": Con esta opción el administrador no escribe nada al medidor.
- Opción "-D": Ejecuta NeMaC como un demonio.
- Opción "-F fichero_flujo": Nombre que le vamos a dar al fichero de flujo de datos.
- Opción "-L fichero_log": Nombre del fichero de log.
- Opción "-P": Abre, añade y cierra un fichero log.

3.3.2 NeTraMet y NetFlowMet.

Ahora nos vamos a centrar en la configuración de los medidores, la cual es similar en casi todas las opciones menos una, ésta es en la opción '-i' que explicaremos de forma mas detallada a continuación. Las opciones de configuración son las siguientes:

- Opción "-i" para NeTraMet: Esta opción especifica la interfaz a monitorizar con un máximo de 4 interfaces. Ejemplo "-i eth0"
- Opción "-i" para NetFlowMet: Esta vez, no se especifica la interfaz que se va a medir, sino el puerto UDP por donde Cisco nos exporta los flujos de datos. Ejemplo "-i 3001" donde 3001 es el puerto por donde nos llegan flujo de datos.
- Opción "-k": Desactiva la entrada de teclado para que no podamos introducir otras opciones que comentaremos mas adelante.
- Opción "-l": Especifica al medidor que debería usar el campo de la cabecera IP para saber el numero de bytes de un paquete IP. Por defecto el medidor usa el tamaño de paquete MAC.
- Opción "-m ppp": Especifica el puerto UDP para la comunicación con el colector. Por defecto este puerto es el 161 (establecido por SNMP).
- Opción "-n nnn": Configura la tasa de muestreo del medidor. El medidor solo procesara uno de cada 'nnn' paquetes. Por defecto es 1.
- Opción "-r rsc": Especifica una comunidad de lectores SNMP para el medidor NeTraMet. Solo se puede especificar una comunidad.
- Opción "-u nnn": Asigna espacio en el medidor para un máximo de 'nnn' reglas. Por defecto está a 2000 reglas.
- Opción "-w wsc": Especifica la comunidad de escritores SNMP del medidor. Siendo la comunidad de escritores distinta a la de lectores.
- Opción "-D": Establecemos el medidor como demonio en nuestra maquina.
- Para la asignación de memoria usamos las siguientes opciones:
    - Opción "-f fff": Asignación de memoria para flujos. Por defecto son 10000
    - Opción "-u rrr": Asignación de memoria para las reglas. Por defecto son 2000.
    - Opción "-b bbb": Asignación de memoria para flujos TCP. Por defecto son 500.
    - Opción "-t ttt": Asignación para corrientes TCP. Por defecto están a 2000.

Las opciones que podemos introducir por teclado si no hemos activado la opción –k son las siguientes:

- Tecla “a”: Esta nos muestra la tabla de información la cual nos dice que reglas se están ejecutando y cuales no.
- Tecla “b”: Nos muestra la cuenta de los paquetes perdidos.
- Tecla “e”: Nos muestra la tabla de información del colector.
- Tecla “f”: Nos muestra información de los flujos activos e inactivos, además de información sobre intervalos de recolección y direcciones IP de los colectores.
- Tecla “m”: Nos muestra la memoria usada por el medidor.
- Tecla “p”: Muestra los protocolos que el medidor puede reconocer. Esto dependerá de las tablas de reglas.
- Tecla “s”: Muestra estadísticas de rendimiento del medidor. Estas estadísticas las explicaremos al final de esta sección.
- Tecla “t”: Muestra en intervalos de tiempo desde que se arranco el medidor, estas son mostradas en centésimas de segundo.
- Tecla “u”: Muestra la tabla de información del conjunto de reglas.
- Tecla “z”: Pone a cero las estadísticas del medidor.
- Tecla “?”: Muestra la ayuda.

Las estadísticas anteriormente mencionadas son las siguientes:

Aps	→	promedio de paquetes por segundo.
Apb	→	promedio de paquetes atrasados.
Mps	→	máximo de paquetes por segundo.
Mpb	→	máximo de paquetes atrasados.
Lsp	→	número de paquetes perdidos.
Avi	→	promedio de procesador desocupado en %
Mni	→	mínimo procesador en %
Fiu	→	flujo activos.

Estas estadísticas son procesadas usando contadores que se actualizan cada segundo. Estos contadores pueden ser puestos a cero por el administrador, o presionando la tecla ‘z’. Las estadísticas son suministradas así como evalúan el rendimiento del medidor en varias configuraciones hardware, cargas de tráfico de red y tabla de reglas.

Capítulo 4

SRL (Simple Ruleset Language)

4.1 INTRODUCCION

Una vez estudiada la arquitectura RTFM (Realtime Traffic Flow Measurement) y en particular la arquitectura de la plataforma NeTraMet basada en la misma filosofía que RTFM, nos disponemos a la comprensión y completo estudio de un lenguaje de especificación de reglas de filtrado de tráfico. Este lenguaje se llama SRL (Simple Rules Language) [16] y es incluido dentro de lo que es la plataforma NeTraMet desde la versión 4.2.

Mediante este lenguaje podemos crear ficheros de configuración para el filtrado de tráfico que pueden ser cargados en cualquiera de los medidores que existen en esta plataforma "NeTraMet y NetFlowMet" por medio de un administrador, que en nuestro caso sería "NeMaC".

Podemos decir que un conjunto de reglas para un medidor RTFM no es más que un conjunto de instrucciones que se ejecutan en el PME (Packet Matching Engine) *ver capítulo 3* del medidor de forma secuencial, ya que el lenguaje SRL es un lenguaje estructurado y procedural. Este conjunto de reglas es ejecutado cada vez que llega un nuevo paquete al medidor, con las cuales el medidor decide si este flujo es de interés y los salva en su tabla y si el flujo no es de interés, éste es ignorado.

El compilador SRL está incluido en la plataforma NeTraMet [8], el cual mediante el fichero de reglas creado por el usuario con una extensión ".srl" para crear otro fichero de reglas con extensión ".rules" que es comprensible por el PME del medidor.

4.2 Opciones de compilación.

El programa "srl" es un compilador de fichero de reglas SRL, el cual esta documentado y se suministra tanto en la plataforma NeTraMet como en cualquier documentación sobre RTFM. Cuando este compilador es ejecutarlo, utiliza el fichero creado por el usuario '.srl', y si este esta libre de errores de sintaxis crea un fichero objeto '.rules',

que es utilizado por NeMaC. Por ejemplo: *./srl id.srl* donde id.srl es el fichero creado por el usuario.

A continuación se expondrán las opciones que nos suministra el compilador para optimizar nuestras reglas:

Línea de comando: ./src/srl [opciones] [fichero srl]

- Opción "-l": Muestra el programa srl por pantalla. Por defecto solo muestra las líneas con errores sintácticos.
- Opción "-s": Solamente chequea errores sintácticos sin crear ningún fichero objeto.
- Opción "-a nn": Determina el nivel de ensamblado de salida de los ficheros objetos.
    - nn = 0: Las reglas solo están en forma numérica.
    - nn = 1: Atributos y acciones dados como palabras.
    - nn = 2: Es como en la anterior pero no borra los ficheros inmediatamente.
- Opción "-O nn": Indica el nivel de optimización:
    - nn = 0: No hay optimización.
    - nn = 1: Optimiza borrando reglas redundantes.
    - nn = 2: Optimiza mirando las longitudes de las mascaras en las expresiones.

4.3 Sintaxis de los ficheros de reglas.

Un programa *SRL* debe seguir unas características. Primero el nombre del fichero debe tener la extensión '.srl', una vez compilado, se creara un fichero objeto, y después el fichero de reglas cuya extensión será '.rules'.

La estructura que debe seguir cualquier fichero de reglas será el siguiente:

- Definiciones de identificadores.
- Estructura del programa principal.
- Realización de subrutinas, que se pueden comparar con un procedimiento.
- Formato del fichero de flujo que obtendremos después de filtrar los datos.

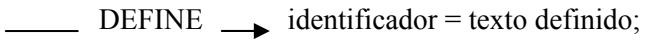
- Estadísticas.

Las palabras de un programa siguen las siguientes reglas:

- Pueden aparecer comentarios al igual que en cualquier lenguaje de programación; éstos deberán colocarse siempre después de una '#', así el compilador sabrá que es un comentario y lo tratará de tal forma.
- Los espacios en blanco separan los símbolos.
- Los identificadores deben empezar siempre con una letra y pueden contener números, letras y '_'
- Hay palabras reservadas que no se pueden utilizar como identificadores.

Para explicar el lenguaje SRL utilizaremos diagramas de raíles, los cuales nos ayudarán a comprender mejor el orden de ejecución de las sentencias. Para entender los diagramas de raíles siguientes, éstos se deben de mirar de izquierda a derecha y de forma descendente.

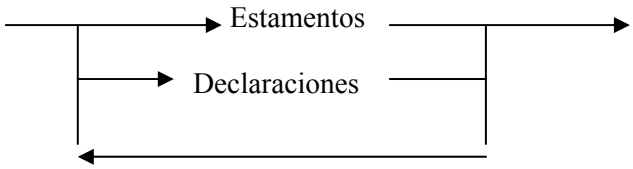
4.3.1 Define



Por ejemplo:

```
Define red_privada = 192.168.1/24,192.168.2/24;
Define red_telematica = 212.128.24/24;
Define labit401 = 212.128.24.50;
```

4.3.2 Programa



Estas declaraciones terminan en ";" o en paréntesis puestas correctamente.

4.3.3 Declaraciones

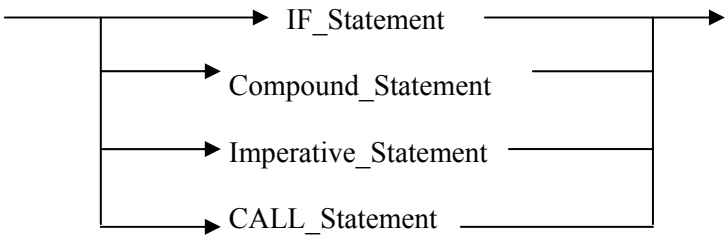
Es solo una declaración explicita de una subrutina, otras declaraciones implícitas llevan etiquetas y parámetros de estas subrutinas.

Por ejemplo:

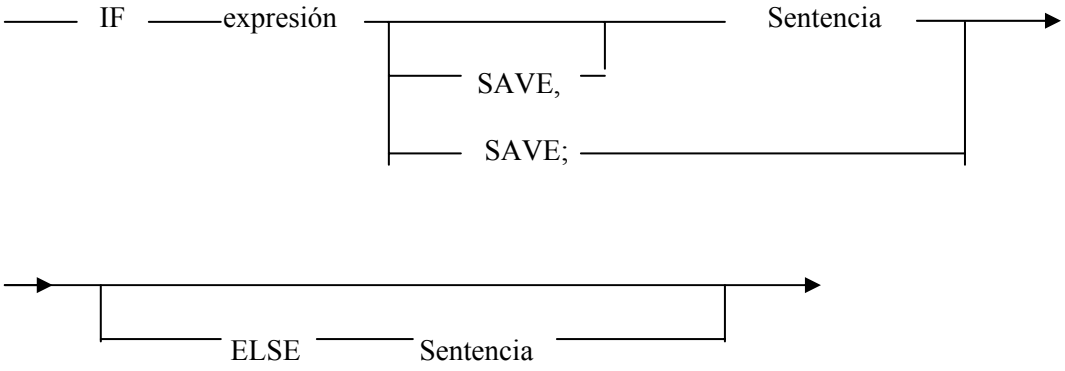
```
SAVE SourcePeerAddress/16;
IGNORE;
COUNT;
```

4.3.4 Sentencias

Hay cuatro clases:



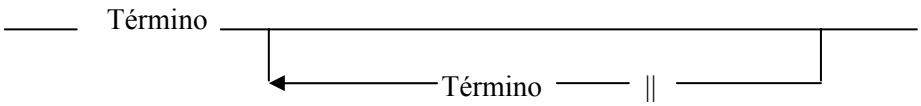
4.3.4.1 Sentencia IF



Por ejemplo:

```
IF (SourcePeerType == IP) || (DestPeerType == IP) SAVE;
ELSE IGNORE;
```

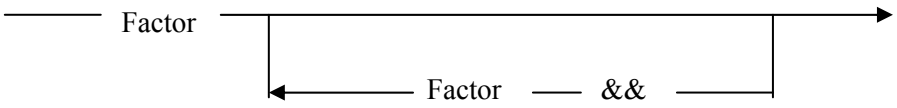
4.3.4.1.1 Expresión



|| ➔ Operador lógico OR

Siendo la expresión de la sentencia IF: `(SourcePeerType == IP) || (DestPeerType == IP)`.

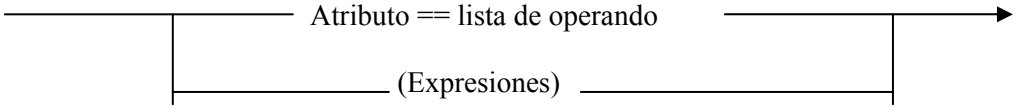
4.3.4.1.2 Término



&& ➔ operador lógico AND

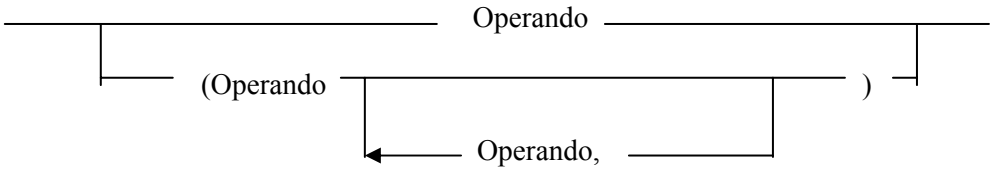
Siendo los factores de un término de la condición por ejemplo: `(SourcePeerType == IP)`

4.3.4.1.3 Factor



Podemos encontrar una gran variedad de atributos reconocidos por la MIB [13] [21], muchos de estos atributos más importantes se encuentran en las secciones 4.3.4.5 (Atributos Generales) y en la sección 4.4 (Atributos más característicos del protocolo de red IP).

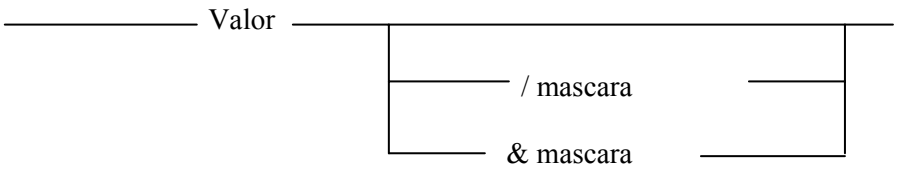
4.3.4.1.4 Lista de operando



Los operando mas utilizados en la creación de reglas, son los utilizados para salvar los atributos y contarlos, por ejemplo:

```
SAVE SourcePeerAddress/32;
COUNT;
IGNORE;
```

4.3.4.1.5 Operando



4.3.4.1.6 Test Part

Las sentencias ID evalúan expresiones lógicas, si el valor de la expresión es TRUE o FALSE, se ejecutan las acciones indicadas.

La forma más simple para estas expresiones es un test de igualdad (operador ==), por ejemplo: *SourcePeerAddress == 212.128 & 255.255*. Si el atributo *SourcePeerAddress* tiene como valor 212.128.24.41 y realizamos la operación con la mascara de red, obtenemos que esta dirección IP pertenece a la red, por lo que el test saldrá verdadero.

Las máscaras pueden especificarse como:

- & 255.255
- & FF-FF
- /16

En SRL un valor es siempre combinado con una máscara; esta combinación es referenciada como un operando. Por ejemplo, si nosotros estamos interesados en capturar flujos IP de la red 130.216, nosotros escribiremos:

- *IF SourcePeerAddress == 130.216.0.0/16 SAVE;*

  O también podría ponerse: *IF SourcePeerAddress == 130.216.0.0&255.255 SAVE;*

  También se puede especificar una lista de valores encerrados entre paréntesis:

- *IF SourcePeerAddress == (130.216.7/24, 130.216.34/24) SAVE;*

  Los bits que faltan a la derecha se rellenan con ceros.

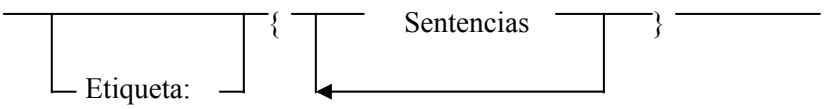
4.3.4.1.7 Acciones

Una acción SAVE salva atributos, máscaras y valores dados en la expresión. Si se testea la expresión más de un atributo las máscaras y valores son salvados para todos los atributos. Para cada lista de valores en la expresión el valor salvado es el primero de las parejas actuales. El resto de las acciones se describirán en la parte de declaraciones imperativas.

4.3.4.1.8 Cláusula ELSE

Una cláusula else en una declaración será ejecutada si el test del IF falla. Esta declaración se coloca siempre inmediatamente después de un IF.

4.3.4.2 Declaraciones compuestas

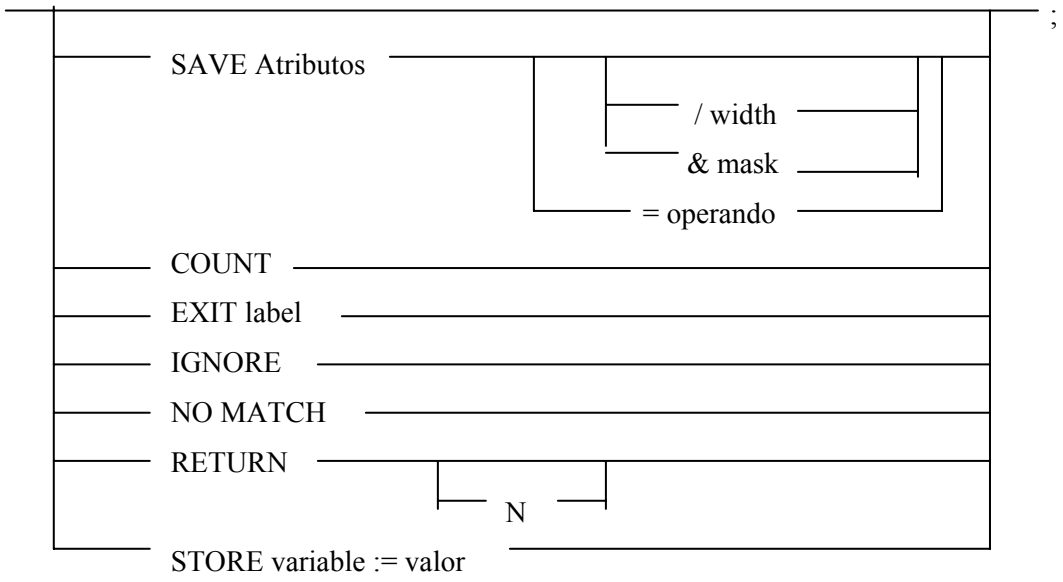


Una declaración compuesta es una secuencia de declaraciones encerradas entre llaves, cada declaración debería terminar en punto y coma (;) a menos que haya otra declaración compuesta.

Una declaración compuesta puede tener etiquetas, por ejemplo precedido por un identificador seguido por un punto y coma. Cada declaración dentro de las llaves es ejecutada secuenciálmente a menos que aparezca una sentencia EXIT, la cual explicaremos más tarde.

Las etiquetas bien definidas en su alcance (rango en el que la etiqueta es válida) deben ser únicas, las etiquetas definidas en una subrutina son locales a ésta y no son visibles fuera de esta subrutina.

4.3.4.3 Declaraciones Imperativas



4.3.4.3.1 Declaración SAVE

Salva la información la cual más tarde identificará el flujo en la tabla del flujo del medidor, pero no se guarda en ninguna de las tablas, esto se hace con la declaración COUNT.

SAVE tiene dos posibles formas:

- SAVE atributo = operando; salva los atributos máscara y valores dados en la declaración. Esta forma de salvar declaraciones es similar a la seguida en la declaración IF, excepto que después de la declaración imperativa no realiza un test sino, que se puede salvar un valor arbitrario:
- SAVE atributo;

SAVE atributo / width;

SAVE atributo & mask;

4.3.4.3.2 Declaración COUNT

La declaración COUNT aparece después de todo el procesado y produce un salvado completado del atributo; el PME construye el identificador de flujo para que los atributos puedan ser salvados, buscando los atributos en la tabla de flujo del medidor o (creando nuevas entradas si éste es el primer paquete observado) e incrementando sus contadores. Entonces el medidor pasa a observar el siguiente paquete.

4.3.4.3.3 Declaración EXIT

La declaración EXIT sale de una composición de declaraciones dentro de una etiqueta. Esto suministra una buena definición de caminos para saltar por el programa.

```
Etiqueta: {     ...
                IF SourcePeerAddress == 192.168/16 EXIT Etiqueta;
                …
                }
```

Rara vez se utilizarán las etiquetas porque el lenguaje tiene suficientes expresiones lógicas para no utilizar EXIT.

4.3.4.3.4 Declaración IGNORE

La declaración IGNORE termina la reexaminación de la corriente de paquetes sin salvar ninguna información. El medidor continúa examinando la siguiente entrada de paquetes, empezando otra vez desde la primera línea del programa de reglas.

4.3.4.3.5 Declaración NOMATCH

La declaración NOMTCH indica que la igualdad ha fallado para esta ejecución del programa, entonces el PME intercambia las direcciones origen y destino y vuelve a iniciar la regla.

4.3.4.3.6 Declaración STORE

La sentencia STORE asigna un valor a una variable SRL y la salva. Hay 6 variables SRL:

SourceClass, SourceKind, DestClass, DestKind, FlowClass, FlowKind.

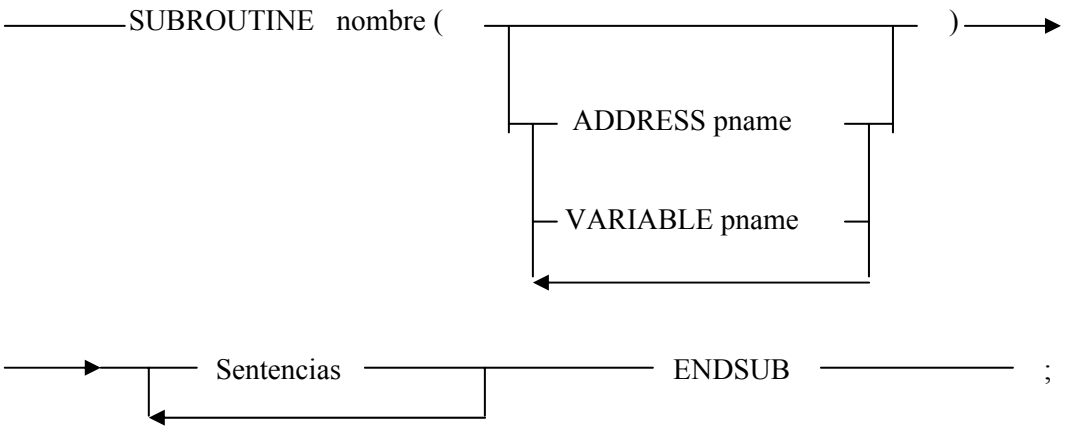
Sus nombres no tienen un significado particular, éstos se han escogido arbitrariamente como probables atributos RTFM, pero pueden ser usados para salvar algún valor entero o byte simple. Sus valores se inicializan a 0 cada vez que se examinan nuevos paquetes.

- STORE SourceClass := 3;
- STORE FlowKind := 'w';

4.3.4.3.7 Declaración RETURN

La sentencia RETURN es usada para devolver (salir) de las subrutinas, y sólo puede ser usada dentro del contexto de las subrutinas, esto es descrito con detalle dentro de la declaración CALL.

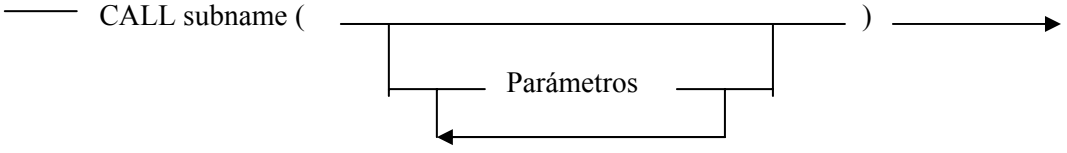
4.3.4.3.8 Subrutinas

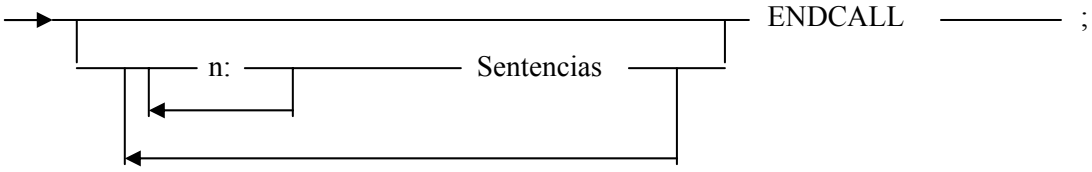


Una declaración de una subrutina tiene tres partes:

- nombre; es el identificador que se le dará a la subrutina.
- La lista de parámetros especifica los parámetros de la subrutina. Cada parámetro es precedido con una palabra que indica su tipo de variable indicando una variable SRL (mirar la declaración STORE), ADDRESS indica algún otro atributo RTFM. Un nombre de parámetro puede ser algún identificador, y su alcance está limitado por el cuerpo de la subrutina.
- El cuerpo indica qué procesamiento de la subrutina se realizará. Éste es simplemente una secuencia de sentencias, que termina con la palabra ENDSUB.

4.3.4.4 Declaración CALL





La declaración CALL invoca una subrutina SRL. Los parámetros son variables SRL u otros atributos RTFM, y sus tipos pueden asociarse a éstas en la declaración de la subrutina. Siguiendo a los parámetros hay una secuencia de declaraciones, cada una precedida de un número entero como etiqueta. Estas etiquetas serán normalmente 1:, 2:, 3:, etc., pero no tienen por qué ser continuas, no en algunos casos particulares. Éstas son referidas en la declaración RETURN con el cuerpo de la subrutina.

Por ejemplo: RETURN 2; debería devolver a la declaración de la etiqueta 2: en la declaración de CALL.

Si en la declaración RETURN no especificamos una etiqueta, la primera sentencia que se ejecuta después de RETURN será la sentencia inmediatamente después de ENDCALL.

4.3.4.5 Atributos generales

Los atributos generales son los que describen el flujo por el mismo y no por los datos extraídos de su cabecera, estos atributos son:

- SourceInterface, DestInterface; Es un numero entero que a partir del cual se indica la interfaz donde se ha capturado el flujo, NeTraMet especifica el entero (1, 2, 3,...) por orden en el que se han especificado las interfaces en la línea de comando (-i eth0 eth2 eth1).
- SourceClass, DestClass, FlowClass, SourceKind, DestKind, FlowKind.
    - 6 atributos no extraídos de los paquetes. En cambio ellos están puestos por el medidor durante el procesamiento del paquete por la ejecución de la acción PushRuleTo (acción que salva el nombre, mascara y valor del atributo en el PME) en las reglas.
- FlowIndex; Inserta el flujo en las tablas de flujo de Netramet.
- FlowRuleSet; Número de reglas del medidor que son usadas cuando el flujo es observado.
- ToOctets, FromOctets; Número de Bytes observados en cada dirección, en el ToOctets (de origen a destino), y FromOctets (de destino a origen) direcciones para estos flujos.

- ToPDUs y FromPDUs: Número de paquetes totales vistos en cada dirección.
- FirstTime: Tiempo en milisegundos que indica el momento que fue observado el flujo con respecto al momento de ejecución del medidor.
- LastTime: Indica el tiempo en milisegundos en el que el flujo fue observado por última vez.

4.4 Atributos IP

- Si SourcePeerType o DestPeerType es igual a 1 se refiero al protocolo Ipv4.
- SourcePeerAddress y DestPeerAddres indica la dirección de origen y destino respectivamente.
- SourcePeerMask y DestPeerMask son direcciones de máscaras, del tipo 255.255.255.0
- DSCodePoint. The Diferénciate Service Code Point (0..63) son los seis bits mas significativos dentro del campo ToS de los paquetes IPv4 y IPv6 que especifica la prioridad de cada paquete de datos. NeTraMet representa este atributo como un entero entre 0 y 63.
- SourceTransType y DestTransType Indica el tipo de protocolo de transporte que utiliza el paquete:
    - 1 == ICMP; si TransType es ICMP entonces SourceTransAddress y DestTransAddress pueden tomar estos valores:
        - 0 == echo reply
        - 3 == destination unreachable
        - 5 == redirect
        - 8 == echo request
    - 2 == IGMP
    - 3 == GGP
    - 6 == TCP
    - 8 == EGP
    - 17 == UDP
    - 18 == OSPF
    - 20 == FTP-DATA
    - 21 == FTP
    - 23 == TELNET
    - 25 == SMTP
    - 27 == RDP
    - 53 == DOMAIN
    - 70 == GOPHER
    - 80 == WWW
    - 94 == IPIP: Tuneling
    - 119 == NNTP
    - 123 == NTP
    - 161 == SNMP

- SourceTransMask y DestTransMack es como las direcciones de máscaras anteriormente explicadas, la diferencia es que los valores de SourcePeerType y DestPeerType se representan con 16 bits, entonces las máscaras serán del tipo 255.255 o 65535.

4.5 Resultados estadísticos.

Estos resultados estadísticos pertenecen al medidor, ya que es necesario saber cuál es la carga de su CPU para evitar que éste se sature y se pierdan. Es necesario que el ordenador que esté de medidor sólo se utilice para este fin, ya que si recibiera flujo de tráfico de varios routers podría saturarse. Para obtener los datos estadísticos del medidor se añaden al fichero de reglas la sentencia "`STATISTICS;`", lo que nos proporciona al final de cada flujo dentro del fichero una línea con las estadísticas. Los datos estadísticos son los siguientes:

Aps → promedio de paquetes por segundo que pasan por el medidor.
Apb → promedio de paquetes atrasados.
Mps → máximo de paquetes por segundo que pasan por el medidor.
Mpb → máximo de paquetes atrasados.
Lsp → número de paquetes perdidos.
Avi → promedio de procesador desocupado en %
Mni → mínimo procesador desocupado en %
Fiu → flujo en uso.
Frc → flujo recuperado.
Gci → basura recolectada en intervalo en segundos de la tabla de flujo del medidor.
Tpp → Cuenta parejas de paquetes que pasan por el medidor.
Cpt → Compara parejas de paquetes contados que pasan por el medidor.
Tts → Total de la cuenta de tablas asignadas.
Tsu → Cuenta tablas en uso.

4.6 Aplicación de las reglas a casos reales:

Ahora vamos a aplicar lo estudiado hasta el momento a casos reales de redes existentes en la universidad.

4.6.1 La red del laboratorio.

4.6.1.1 Descripción y componentes

Nuestra red de pruebas se compone de la red privada de un laboratorio de investigación y desarrollo (I+D) configurada con las direcciones 192.168.1.X, esta red privada estaba unida a la red troncal de la universidad por un router Cisco 2600 que realiza la función NAT (Network Address Translation). Entre la red privada y el router Cisco existía un router Teldat debido a un problema con las interfaces ethernet. Este router Teldat no repercute en ningún momento con la regla (Figura 4-2) realizada para esta red de la Figura 4-1.

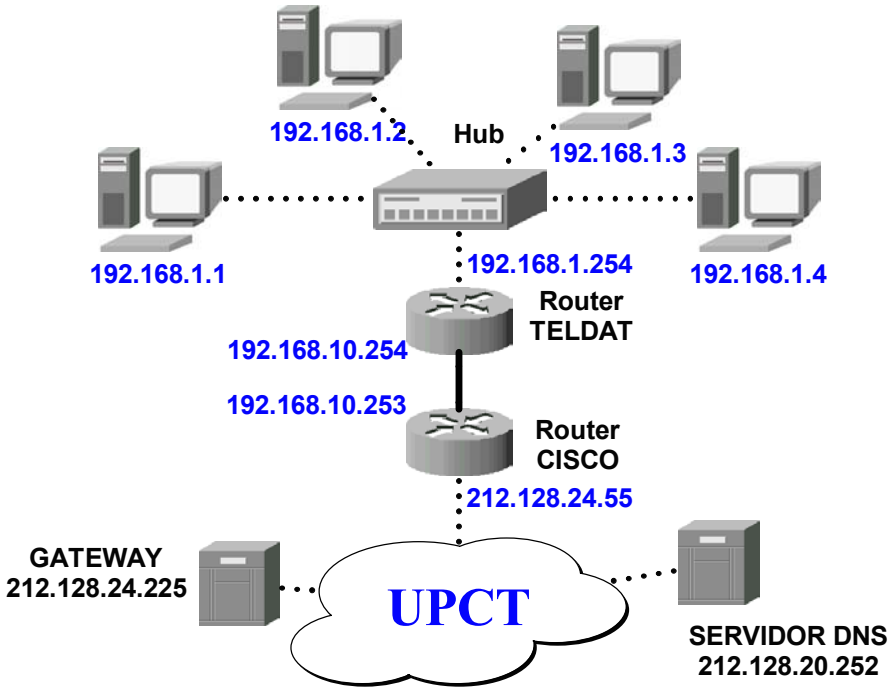


Figura 4-1.

4.6.1.2 Diseño de la regla para nuestra red de pruebas.

Ahora sería el momento de empezar a diseñar reglas para capturar nuestro propio tráfico y ver que realmente era lo que queríamos. Nuestro primer fichero de reglas para esta red es el que se muestra en la Figura 4-2.

```
1- DEFINE Red = (192.168.1/24); #Pool de direcciones de la red privada

2- DEFINE Cisco = (212.128.24.55/32,192.168.10.253/32, 192.168.10.254/32,192.168.1.254/32);
         #Direcciones del Cisco

3- #Pasamos a ignorar cualquier dirección que no sea IP.

4- IF SourcePeerType == IP SAVE;

5- ELSE IGNORE;

6- SUBROUTINE Clasificar (ADDRESS peer) #Pasamos como argumento de la subrutina una dirección.

7-        # Esta dirección se compara con la del Cisco si es igual la salvamos y salimos de la subrutina
          devolviendo el valor 1. De esta forma sabemos que la dirección era una de las interfaces del
          cisco. Si nos devuelve 2 es tráfico del teldat y 3 si es de la red privada.

8-        IF peer == Cisco {

9-                  SAVE peer/32;

10-                 RETURN 1;

11-       }

12-       ELSE IF peer == Red {

13-                 SAVE peer/32;

14-                 RETURN 2;

15-       }

16-       SAVE peer/32;

17- ENDSUB;

18- #miramos si la dirección ip de origen es del Teldat, Cisco o de la red...

19- CALL Clasificar (SourcePeerAddress)

20- #Al devolvernos 1 sabemos que es del Cisco.

21- 1:{   CALL Clasificar (DestPeerAddress)  #Miramos las direcciones de destino

22-       1:IGNORE;                    #Paquetes que tienen destino el cisco son ignorados

23-       2:COUNT;                     #El resto los contabilizamos

24-       ENDCALL;

25-       STORE FlowKind := 'C'; #Indicamos que es trafico de Internet a la red

26-       COUNT;

27-       }

28- #al devolvernos 2 sabemos que es del Teldat.

29- 2:{   CALL Clasificar (DestPeerAddress)  #Miramos las direcciones de destino

30-       1:COUNT;

31-       2:IGNORE; #los paquetes con direcciones de destino Teldat son ignorados el resto lo cogemos como
          validos

32-       ENDCALL;

33-       STORE FlowKind := 'R'; #Indicamos que el tráfico es de la red a Internet

34-       COUNT;

35-       }

36- ENDCALL;

37- FORMAT SourceSeerType  "@ origen:" SourcePeerAddress "- @dest :" DestPeerAddress "bytes q entran cisco-
>" ToOctets "bytes cisco internet ->" FromOctets
```

Figura 4-2.

En nuestra red de pruebas queremos estudiar solamente el tráfico generado por la red hacia Internet y el tráfico de Internet hacia nuestra red, por la arquitectura de nuestra red observamos que el trafico generado por nuestra red a Internet siempre pasa por el router cisco, por lo que podríamos clasificarlo como el trafico de origen la red y destino cisco y con el trafico entrante ocurre exactamente lo mismo. No nos interesa el tráfico interno de la red de pruebas ni los posibles procesos locales dentro del router Cisco que utilice el protocolo IP.

Para la creación de esta regla nos basamos en la recursividad (método muy usado en la programación estructurada). Para utilizar este método generamos una subrutina (*líneas 6-17*) que comprueba que dirección se le ha pasado como parámetro, si esta dirección pertenece al router CISCO la subrutina salva la dirección y devuelve 1, si en cambio se le ha pasado la dirección de la red de pruebas, devuelve un 2 y salva también la dirección.

Una vez generada la subrutina pasamos a generar la regla propiamente dicha (*líneas 19-36*). Para llamar a la subrutina utilizamos la sentencia CALL (*líneas 19,21 y 29*). Mediante la Figura 4-3 podremos comprender mejor el funcionamiento de las llamadas a la subrutina:

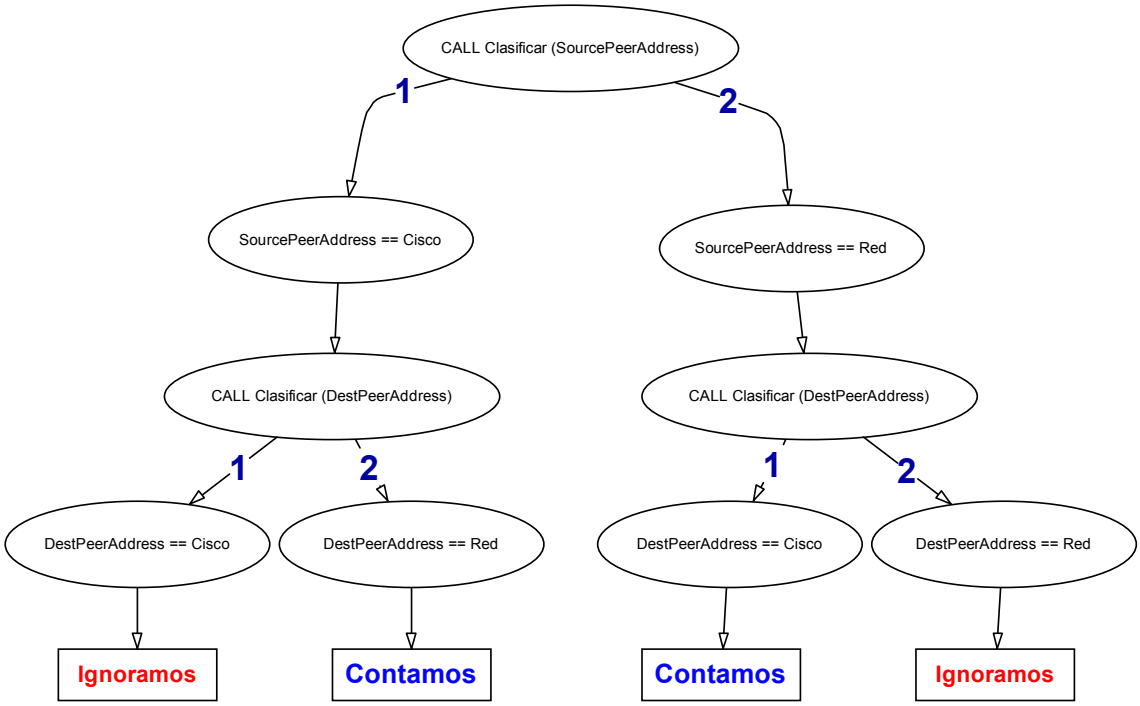


Figura 4-3.

En la primera llamada que se hace en la línea 19 obtenemos cual es la dirección origen, ya sea de cisco o de la red de pruebas. Si la red de origen es la de cisco volvemos a llamar a la subrutina, pero esta vez con la dirección destino y si la dirección destino es la red de pruebas salvamos los atributos y contamos el flujo, si no es así lo ignoramos. Para el caso que la dirección origen sea la red de pruebas es similar el funcionamiento.

4.6.2 Aplicación de las reglas a una red más compleja.

Después de haber realizado numerosas pruebas en el laboratorio, realizamos pruebas con una red más compleja (ver Figura 4-4). Un router cisco de una red institucional nos empezó a exportar flujo generado por el tráfico de su red.

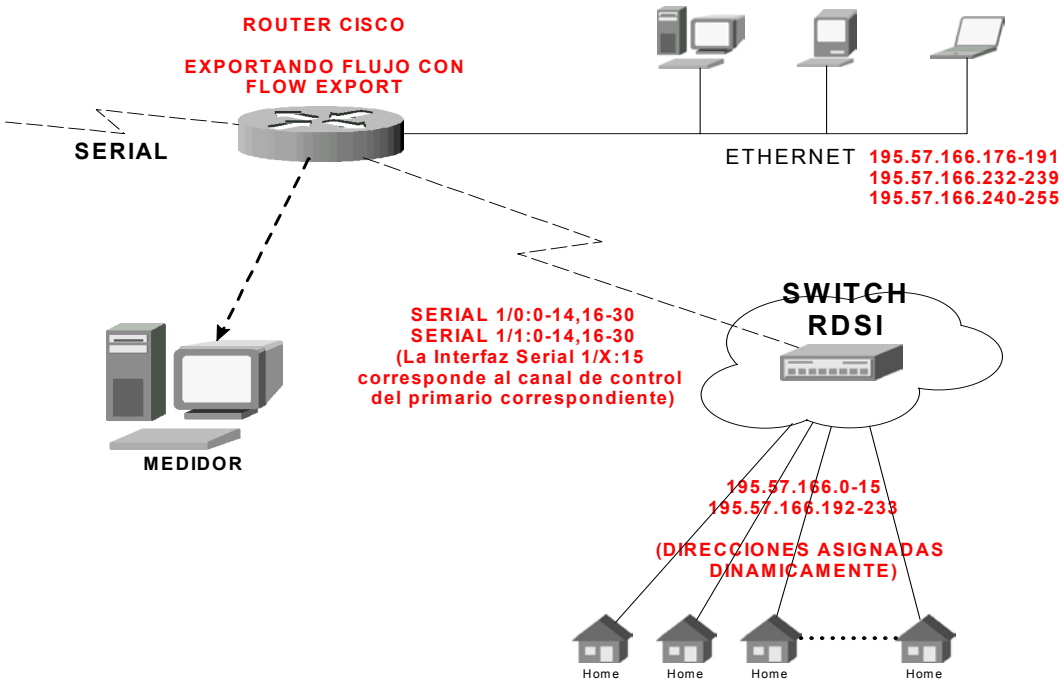


Figura 4-4.

Esta red posee un router Cisco con dos accesos primarios (30B + D) que se corresponden con las interfaces físicas Serial 1/0 y 1/1. Estas interfaces son las encargadas de dar acceso a Internet a los usuarios conectados por medio de RDSI de la red institucional.

Estas interfaces de red se configuran a través de un canal de control (que en Europa es el 15, y en EE.UU. el 23). Toda la configuración aquí especificada se aplicará a los 30 canales de este primario.

Para dar conexión a otros routers a través de los canales del primario, se ha habilitado el Dial-on-Demand-Routing (DDR). Como consecuencia de esto se han creado unas interfaces lógicas llamadas "Dialer" o "Perfiles de llamada" para separarlas la configuración de las interfaces físicas de la configuraciones lógicas requeridas para recibir una llamada DDR. Estas interfaces son las encargadas de asignar un destino según el perfil del llamante. Será en estas interfaces donde se detallará toda la configuración que se aplicará en cada canal del primario por el que se conecte el usuario correspondiente, como la IP o el Multilink.

Se han definido un total de 44 interfaces lógicas Dialer, tantas como direcciones IP se disponen para este enlace (un total de 44), y de esta forma dar conexión a 44 usuarios simultáneamente. Estas interfaces Dialers son las siguientes para cada primario:

- Serial 1/0: 39-69
- Serial 1/1: 101-131

Esta red puede observarse en la Figura 4-4.

Una vez comprendida la topología de esta red, nos dedicamos a crear las reglas para filtrar el tráfico de cada uno de los enlaces primarios utilizando las interfaces lógicas Dialer.

4.6.2.1 Filtrado de los enlaces primarios por medio de las interfaces lógicas

```
 1- DEFINE Primario_1 = (39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,
57,58,59,60,61,62,63,64,65,66,67,68,69);

 2- IF SourcePeerType == IP SAVE;

 3- ELSE IGNORE;

 4- IF ((SourceInterface == Primario_1) || (DestInterface == Primario_1))

 5- {

 6-       SAVE SourcePeerType;

 7-       SAVE SourcePeerAddress/32;

 8-       SAVE DestPeerAddress/32;

 9-       AVE SourceInterface;

10-       SAVE DestInterface;

11-       SAVE SourceTransAddress;

12-       SAVE DestTransAddress;

13-       SAVE SourceTransType;

14- }

15- COUNT;

16- SET 5;

17- FORMAT   FlowIndex FlowRuleSet  FirstTime  SourcePeerAddress  DestPeerAddress FromOctets ToOctets
SourceInterface DestInterface
```

Figura 4-5.

Mediante esta regla (Figura 4-5) filtramos tanto la carga de subida como de bajada del tráfico que pasa solamente por la interfaz serial 1/0. Ésto lo hacemos declarando una sola constante (línea 1) que haga referencia a los dialers correspondiente a este primario y salvamos tanto los flujos que lleven de origen o que lleven de destino esas interfaces (líneas de la 4 a la 15). Hacemos exactamente lo mismo para estudiar el trafico del segundo primario como se muestra en la Figura 4-6.

```
DEFINE Primario_2 =
(101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117,118,119,120,121,122,123,124,125,126,127,128,1
29,130,131);

IF SourcePeerType == IP SAVE;

ELSE IGNORE;

IF ((SourceInterface == Primario_2) || (DestInterface == Primario_2))

{

          SAVE SourcePeerType;

          SAVE SourcePeerAddress/32;

          SAVE DestPeerAddress/32;

          SAVE SourceInterface;

          SAVE DestInterface;

          SAVE SourceTransAddress;

          SAVE DestTransAddress;

          SAVE SourceTransType;

}

COUNT;

SET 5;

FORMAT   FlowIndex FlowRuleSet  FirstTime  SourcePeerAddress  DestPeerAddress FromOctets ToOctets SourceInterface
```

Figura 4-6.

Después de definir estas reglas e intentar filtrar el tráfico por medio de las interfaces, vimos que resultaba más cómodo y que obteníamos los mismos resultados que mirando solamente el pool de direcciones.

4.6.2.2 Filtrado de trafico por medio del pool de direcciones.

```
 1- DEFINE pool_1 = 195.57.166.192/27;
 2- DEFINE pool_2 = 195.57.166.0/28;
 3- IF SourcePeerType == IP SAVE;
 4- ELSE IGNORE;
 5- IF ((SourcePeerAddress == pool_1) || (DestPeerAddress == pool_1 ))
 6- {
 7-       SAVE SourcePeerAddress/32;
 8-       SAVE DestPeerAddress/32;
 9-       SAVE SourceTransAddress;
10-       SAVE DestTransAddress;
11-       SAVE SourceTransType;
12-       SAVE SourceInterface;
13-       SAVE DestInterface;
14-       COUNT;
15- }
16- IF ((SourcePeerAddress == pool_2)  || (DestPeerAddress == pool_2 ))
17- {
…         (Igual que en las líneas de la 7 a la 14)
26- }
27- SET 2;
```

Figura 4-7.

Para este fichero de reglas (ver Figura 4-7) utilizamos el pool de direcciones dinámicas que son asignadas dinámicamente a cada usuario cuando se conecta a Internet. En las dos primeras líneas del fichero de reglas observamos las constantes declaradas que identifican los dos pooles de direcciones y mediante las condiciones de las líneas 5 y 16 realizaremos el filtrado de los flujos.

Mediante este fichero de reglas obtenemos el siguiente fichero de capturas. En éste, hemos colocado trozos más significativos, ya que este fichero de capturas es bastante amplio.

```
##NeTraMet v4.3:  -c300 -r primarios.rules  212.128.24.41 udp-3000  10000 flows  starting at 17:36:57 Thu 15 Nov 2001

#Format: firsttime flowindex flowruleset sourcepeeraddress destpeeraddress fromoctets tooctets sourceinterface destinterface sourcetransaddress desttransaddress sourcetranstype desttranstype

#Time: 17:36:57 Thu 15 Nov 2001 212.128.24.41 Flows from 0 to 1390859

#Ruleset: 4

#EndData: 212.128.24.41

#Time: 17:40:00 Thu 15 Nov 2001 212.128.24.41 Flows from 1390858 to 1409067

845330052 2 4 63.95.137.150 195.57.166.194 0 41347 4 0 80 1563 6 6

3430621061 3 4 195.57.166.194 66.28.26.6 0 1252 166 4 1567 80 6 6

2588404614 4 4 195.57.166.194 66.28.26.6 0 1210 166 4 1569 80 6 6

11489926 5 4 195.57.166.194 66.28.26.6 0 1255 166 4 1570 80 6 6

3438996114 6 4 195.57.166.194 216.35.213.254 0 850 166 4 1566 80 6 6

1729463701 7 4 195.57.166.194 216.34.209.13 0 578 166 4 1571 80 6 6

3430608022 8 4 195.57.166.194 216.34.209.13 0 577 166 4 1572 80 6 6
```

Figura 4-8.

En este fichero de flujo de la Figura 4-8 podemos observar como se le asigna a la dirección IP 195.57.166.194 la interfaz lógica o Dialer 166, con cualquiera de las reglas descritas anteriormente podríamos obtener los mismos resultados.

```
1721035920 10 4 216.34.209.13 195.57.166.194 0 683 4 0 80 1572 6 6

3114387 11 4 195.57.166.194 212.106.193.75 0 510 166 4 1033 80 6 6

3114389 12 4 212.106.193.75 195.57.166.194 0 846 4 0 80 1033 6 6

2588444047 13 4 195.57.166.194 212.106.193.75 0 521 166 4 1037 80 6 6

1729450642 14 4 195.57.166.194 64.209.192.101 0 525 166 4 1575 80 6 6

853692292 15 4 195.57.166.194 63.95.137.150 0 1511 166 4 1563 80 6 6

...
```

Figura 4-9.

En este trozo del fichero (Figura 4-9) podemos observar que la interfaz lógica que se utiliza es la 166 y no corresponde con la dirección del ejemplo anterior, esto es debido a lo que comentamos anteriormente, que las interfaces lógicas se van asignando cuando un usuario necesita el canal cuando se conecta.

Toda esta información es tratada y usada para generar los gráficos en tiempo real de la ocupación del enlace primario.

Capítulo 5

Aplicación Para La Duplicación De Flujos En NeTraMet (DUFNE)

5.1 INTRODUCCION

En este capítulo se describe la realización de una aplicación en C para mejorar la gestión del flujo de tráfico exportado por un router CISCO.

El problema reside en el post filtrado de los ficheros de flujo creado por el colector. Si la red a estudiar es muy grande estos ficheros de flujos creados diariamente pueden salir de tamaños impensables, y unido a que un medidor no puede ser administrado por distintos manager (ya que se sobre escriben las reglas entre los administradores) y que no podemos poner varios medidores leyendo del mismo puerto por donde nos exporta el flujo el router. De aquí surgió la idea de intentar duplicar esta información que nos envía el router para poder utilizar varios medidores y poder crear varios ficheros de flujo con información distinta cada uno.

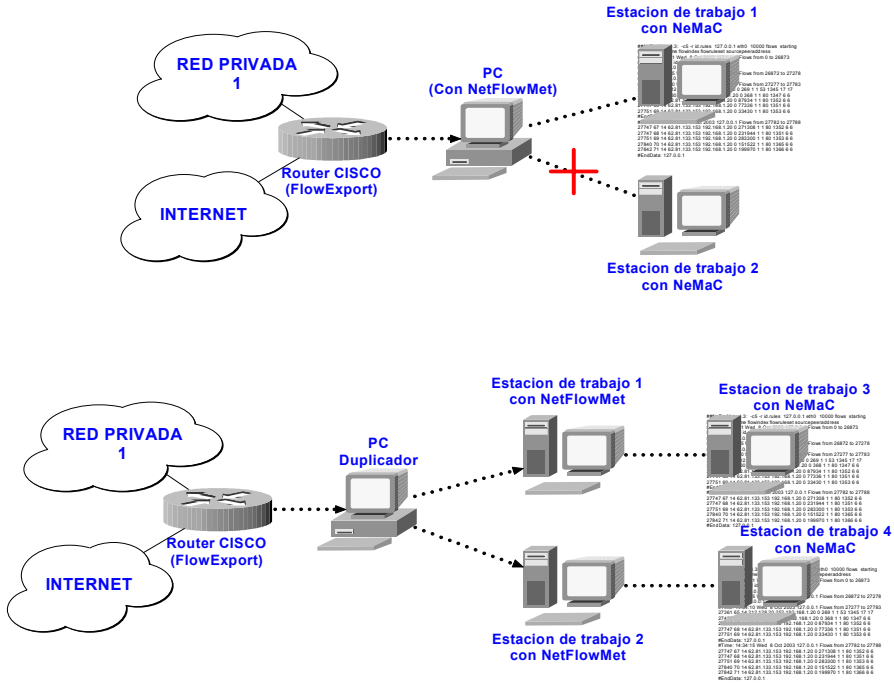


Figura 5-1

5.2 FASE DE DISEÑO DE LA APLICACIÓN

5.2.1 **Análisis del problema.**

El principal problema se resumía en leer el flujo de tráfico que nos exporta el router CISCO sin saber cual es su estructura, su tamaño, y su cabecera.

Para ello nos fijamos que la aplicación NetFlowMet puede leer este flujo, por lo que buscamos dentro de la plataforma NeTraMet un fichero que nos detallara la estructura del flujo. Encontramos el fichero "*flowdata.h localizado en el mismo directorio que el medidor*", una librería de la plataforma que es utilizada por NetFlowMet para reconocer la estructura de los flujos. Siendo la siguiente estructura para la versión 5 del sistema operativo del router CISCO, Figuras 5-1, 5-2 y 5-3.

Mensaje		
Header	Cabecera	
Records[30]	Cuerpo del mensaje	Array de 30 flujos.

Figura 5-2

Cabecera		
Versión	ushort	
Count	ushort	Numero de registros del PDU.
SysUptime	ulong	Tiempo en milisegundos desde que el router exporta flujo.
Unix_secs	ulong	Segundos desde 0000 UTC 1970.
Unix_nsecs	ulong	Nanosegundos desde 0000 UTC 1970.
Flow_secuence	ulong	Contador del total de flujos vistos.
Reserved	ulong	Reservado

Figura 5-3

Cuerpo del mensaje

Srcaddr	ipaddrtype	Dirección IP origen.
Staddr	ipaddrtype	Dirección IP destino.
NextHop	ipaddrtype	Dirección IP del router del siguiente salto.
Input	Ushort	Índice de la interfaz de entrada.
Output	ushort	Índice de la interfaz de salida.
Dpkts	ulong	Paquetes enviados.
DOctect	ulong	Octetos enviados.
First	ulong	SysUptime al empezar el flujo.
Last	ulong	Y el último paquete del flujo.
SrcPort	ushort	Número del puerto origen TCP/UDP o equivalente.
DstPort	ushort	Número del puerto destino TCP/UDP o equivalente.
Pad	uchar	
Tcp_flag	uchar	
Prot	uchar	Tipo de protocolo IP: 6 = TCP, 17 = UDP…
Pos	uchar	Tipo de servicio IP.
Src_as	ushort	
Dst_as	ushort	
Src_mask	uchar	Mascara para la dirección origen.
Dst_mask	uchar	Mascara para la dirección destino
Reserved	ushort	Reservado

Figura 5-4

Una vez analizado este fichero lo adaptamos a nuestro programa, creando un buffer con la estructura del flujo de datos, de tal forma que ahí asignaríamos el flujo que nos llega, para después poder enviárselo a los clientes. Un ejemplo simple de cómo se realizaría, se encuentra en la Figura 5-5.

En este ejemplo a la izquierda se encuentra la estructura del flujo en 'C' y a la derecha nuestro programa. Lo primero que tenemos que hacer es un buffer llamado flujo con la estructura del mensaje "IPStat5Msg flujo;", lo siguiente es establecer una constante con el tamaño de ese flujo mediante la sentencia "#define tam_flujo_v_5 (sizeof flujo);". Una vez hecho esto ya podemos utilizar la variable flujo para recibir mensajes de esas características y almacenarlos en ese buffer y mandarlos a los clientes subscritos.

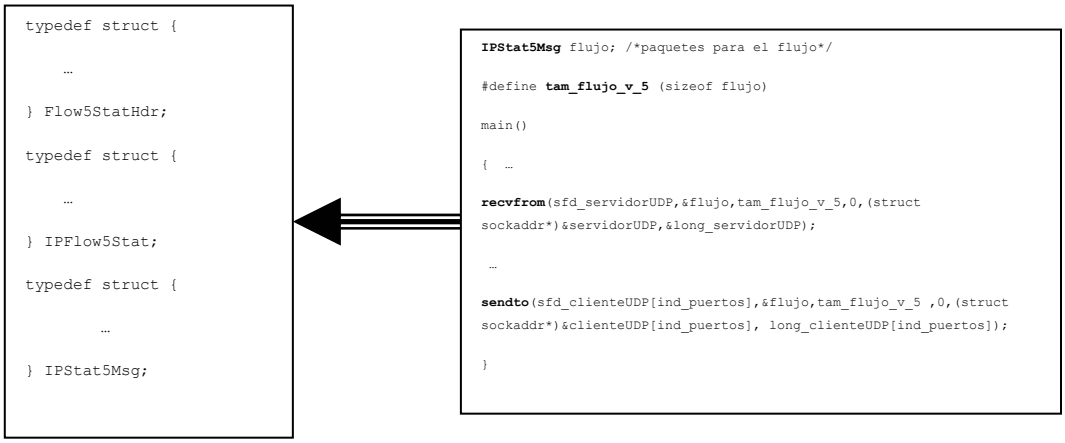


Figura 5-5.

5.2.2 **Problemas que se plantearon en la realización de la aplicación.**

5.2.2.1 Primera versión.

En la primera versión de la aplicación solamente estaba la aplicación del servidor, por lo que si había que añadir algún nuevo puerto para exportar flujos de datos, había que reiniciar el servidor y los medidores a los que se enviaban los datos. Pero ya, esta versión duplicaba los flujos de datos que le llegaba del router a los puertos (indicados por el usuario en la línea de comandos) dentro de la misma maquina. El problema es que varios medidores ejecutándose en una misma maquina consume muchos recursos.

Ejemplo: ./vp 3001 3002 3003 3004 Este ejemplo indica que los flujos que lleguen del router Cisco por el puerto 3001 se duplicarán a los puertos 3002, 3003 y 3004.

5.2.2.2 Segunda versión.

En esta segunda versión se modificó la forma de entrada de los parámetros, de tal forma que se podía poner o los puertos específicos en donde se querrá exportar el flujo, como en la versión anterior o bien poner un rango de valores añadiendo solo la opción “m” en la línea de comandos.

Ejemplo: ./vp2 3001 m 3002 3005 Mediante este comando el programa duplicaría la información que le llega por el puerto 3001 a los puertos 3002, 3003, 3004 y 3005.

5.2.2.3 Tercera versión.

En esta tercera versión podíamos exportar los flujos de datos que nos llegaban por un puerto a uno o varios puertos de una maquina remota, por lo que nos quitábamos el problema de tener que ejecutar varios medidores en la misma maquina local con el problema de que estos medidores consumieran la mayor parte de todos los recursos.

Ejemplo: ./vp3 3001 3002 3003 192.168.1.3 Mediante esta línea de comando el programa exportaba lo que le llegaba al puerto 3001 a los puertos 3002 y 3003 de la maquina remota 192.168.1.3.

5.2.2.4 Cuarta versión.

Esta cuarta versión se intenta mejorar añadiendo concurrencia, lo cual fue un intento fallido ya que entre los procesos padres e hijos se producía exclusión mutua (no podían compartir variables).

En esta misma versión se intentó mejorar también añadiendo la opción de que el servidor aceptara peticiones de un cliente para que le exportara el flujo por un puerto. De esta forma, no habría que reiniciar el servidor cuando se quisiera añadir un puerto, esto seria transparente a los medidores que seguirían leyendo flujos sin interrupciones. El problema que se producía con la concurrencia era que al crear un proceso hijo para cada cliente que realizaba una petición, este nuevo proceso se creaba con las mismas variables, pero independientes de las del padre. Por lo que el padre escribía los flujos en su variable pero los hijos no podían leerlas.

5.2.2.5 Quinta versión.

En esta última y definitiva versión se han implementado una aplicación servidor y otro cliente, de tal forma que desde cualquier máquina, el cliente pide al servidor que le mande flujos de datos. Para esta versión se utiliza la función “select” la cual nos permite escuchar un descriptor de un socket y cuando le llegue un mensaje por el descriptor ejecutar el código asociado. De esta forma estamos escuchando dos descriptores, uno para escuchar las peticiones del cliente y el otro para escuchar el flujo que nos exporta el router cisco. Esta versión se realizó sin concurrencia por lo que se podía plantear el problema de que se pudiera perder un paquete si llegaba justo cuando estaba aceptando la petición de un cliente. Pero que ocurra esto es remotamente difícil ya que Cisco envía una media de un paquete cada 30 segundos, dependiendo de la configuración del router. Haciendo una simulación

con un programa de pruebas, el duplicador de puerto ha llegado a reenviar 545 paquetes por segundo, sin perder ninguno.

5.2.3 Configuración del router para exportar flujo.

Para que el router nos pueda exportar el flujo a una maquina donde supuestamente tendremos la aplicación NetFlowMet, se hará mediante los comandos:

- ip route-cache flow ➔ Con este comando se arranca NetFlow en cada interfaz para ser medidas.
- ip flow-export <dirección IP de la maquina donde estará NetFlowMet> <Puerto UDP que usará NetFlowMet para recibir los datos>

5.2.4 Modo de funcionamiento de la aplicación DUFNE.

La aplicación DUFNE está compuesta por un servidor y un cliente. El servidor se encarga de recibir el flujo de datos exportados por el router y reenviarlos a los clientes que se han subscrito a su lista. Una vez que el servidor los ha metido en su lista, éste les manda vía TCP la dirección del puerto UDP por donde deben de recoger el flujo de datos.

5.2.4.1 Servidor.

Al servidor por medio de la línea de comandos le indicamos el puerto por donde el router le esta mandando el flujo de datos, una vez hecho esto se ejecuta el programa realizando éste un bucle infinito que describiremos a continuación (ver Figura 5-6.):

- El servidor se queda a la escucha de dos posibles eventos que puedan ocurrir:
  - Que un cliente haga una petición para que le exporte flujo de datos.
  - Que lleguen flujos de datos del router.
- Si recibe una conexión vía TCP de un cliente que quiere que le envíen flujo, el servidor lo añade a un array en el que almacena su dirección IP, y el puerto al cual le enviara el flujo de datos. Y por el mismo socket que el cliente le ha hecho la petición, el servidor informa a éste por que puerto UDP deberá escuchar para poder obtener el flujo de datos. Después de todo esto, el servidor cierra la comunicación con este cliente, y se queda a la escucha de otro posible cliente que le pida flujo.

- Si recibe por el puerto UDP un nuevo flujo de datos del router, este lee el flujo de datos del socket y lo almacena en un buffer que posee la misma estructura del paquete que le acaba de llegar, después el servidor recorre el array de los clientes que se han subscrito y les reenvía este mismo flujo a cada uno de ellos. Una vez hecho esto, se queda a la escucha de nuevos posibles flujos.

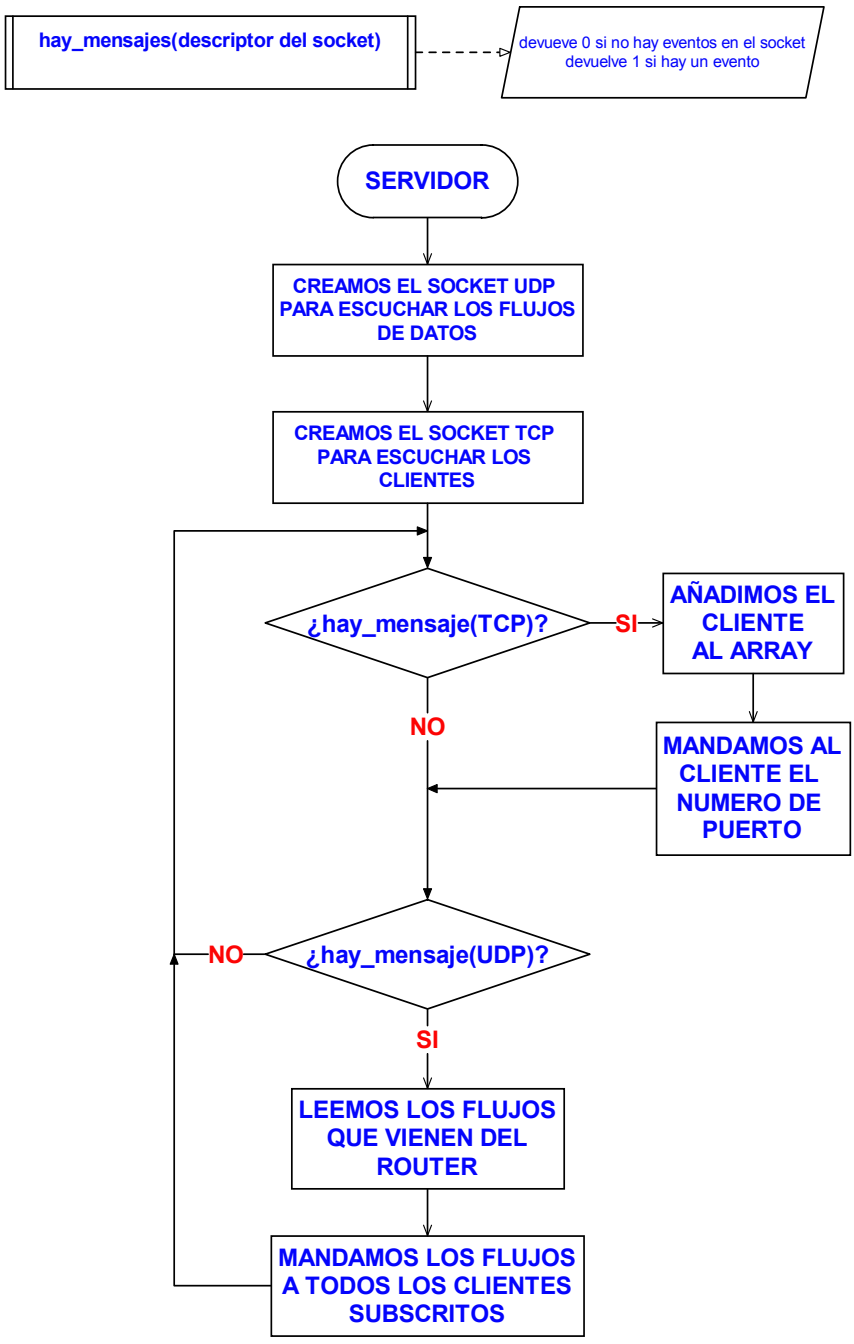


Figura 5-6.

5.2.4.2 Cliente.

El cliente se encarga de establecer una comunicación vía TCP con el servidor, una vez hecho esto el cliente lee el número de puerto por el que recibirá el flujo de datos que ha duplicado el servidor. A partir de ese momento el usuario puede poner un medidor en esa maquina y empezar a medir tráfico. Se puede observar el diagrama de flujo en la Figura 5-3.

Figura 5-7.

5.3 Ejemplo de utilización.

Un ejemplo de utilización es suponer una red como la de la Figura 5-8. Supongamos que un router Cisco conecta una red privada (por ejemplo edificio de oficinas que genera gran cantidad de tráfico) a la red de Internet. Todo este tráfico de Internet generado por el edificio pasa por el router Cisco en el cual esta ejecutándose NetFlow. Debido a esa gran cantidad de tráfico el router generará mucha información a exportar.

Toda esta información generada por el router podría ser filtrada por un solo medidor con una única regla. El problema se crearía a la hora de tratar el gran fichero de flujo que generará el manager NeMaC.

Para evitar este problema utilizamos nuestra aplicación y configuramos tantos medidores como necesitemos. Para nuestro ejemplo del edificio de oficinas generamos dos reglas de filtrado, uno para estudiar la carga de salida y otro para la carga de entrada. Estas dos reglas

estarán cargadas en dos medidores que podrán estar controlados por un solo manager tal y como se explicó en el fichero de configuración de la sección 3.2.1 Administrador y colector:

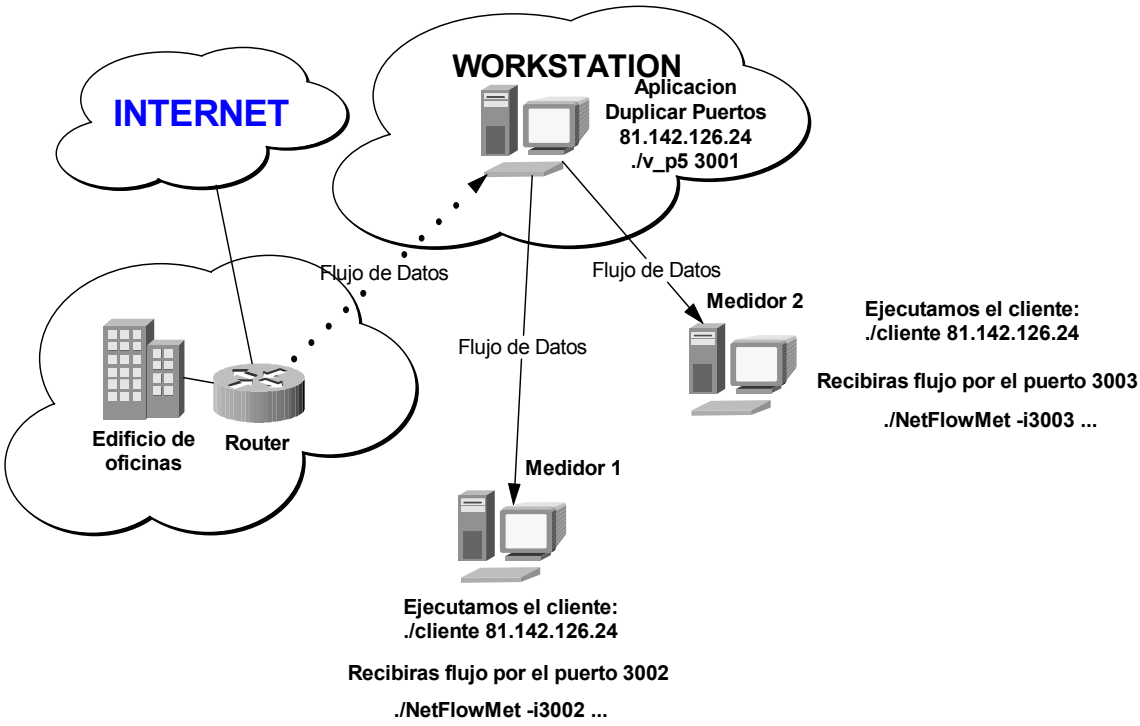


Figura 5-8.

Una vez introducido el problema, pasamos a explicar como se ejecutaría nuestras aplicaciones cliente y servidor.

El router Cisco anteriormente mencionado exporta el flujo a un ordenador (con dirección IP 81.142.126.24) donde teóricamente habrá instalado un medidor NetFlowMet, pero en vez de esto, ejecutamos la aplicación DUFNE con la línea de comando "./v_p5 3001" siendo 3001 el puerto por donde Cisco nos envía los flujos. Ya tenemos el servidor ejecutándose en nuestra estación de trabajo, ahora es el momento de empezar a medir con nuestras reglas la carga de subida y de bajada. En el equipo medidor 1 de la Figura 5-8. queremos medir la carga de subida, por lo que ejecutamos la aplicación cliente "./cliente 81.142.126.24", en ese momento el servidor recibe la petición del medidor 1 y le asigna un puerto "3002" y le envía esta información por el socket TCP. El medidor 1 recibe el numero de puerto por donde el servidor el enviara los flujos duplicados. En este momento ya se puede ejecutar la aplicación NetFlowMet en el medidor 1 y capturar flujos en el puerto 3002. El medidor 2 funciona exactamente igual, la diferencia es que a este medidor se le asignaría un nuevo puerto "3003" y así sucesivamente con el resto de clientes que se subscriban al servidor.

Capítulo 6

Aplicación Para La Gestión Automática De Reglas En NeTraMet (GARNE)

6.1 INTRODUCCIÓN

En este capítulo describimos el proceso de realización de una aplicación hecha en lenguaje JAVA [5] [6] [7] para la realización de reglas, de tal forma que un usuario que utilice por primera vez la plataforma NeTraMet pueda realizar de forma guiada un conjunto de reglas sin necesidad de conocer el lenguaje SRL (Simple Ruleset Language) [16]. Solamente se necesitan unas nociones básicas de redes para entender los términos que describe la aplicación. Estos términos son:

- SourcePeerAddress, DestPeerAddress: Dirección de red origen y destino
- SourcePeerType, DestPeerType: Tipo de protocolo de red origen y destino.
- SourceTransAddress, DestTransAddess: Protocolo de transporte origen y destino.
- SourceInterface, DestInterface: Interfaz origen y destino.
- …

Estos son algunos de los atributos que puede poseer un flujo entre otros muchos, pero estos son los más importantes a la hora de generar reglas.

6.2 DISEÑO DE LA APLICACIÓN.

6.2.1 **Análisis del problema.**

Como hemos comentado anteriormente, el mayor problema a la hora de estudiar el tráfico es la realización de las reglas para disminuir la cantidad de información, de tal forma

que los ficheros de flujo salgan lo más minimizados posibles para agilizar su posterior estudio.

Antes de realizar la aplicación un usuario tenía que leerse y comprender la sintaxis SRL para realizar unas simples reglas, tan solo para capturar trafico IP. A parte tenia que estudiarse como funcionaba la plataforma y como tenía que configurar los medidores (NeTraMet o NetFlowMet) y el manager y colector (NeMaC). Ahora esta persona desde que instale la aplicación y coloque los medidores en los puntos de red necesarios podrá enseguida realizar unas reglas para el filtrado del tráfico.

La aplicación para el diseño de reglas debía proporcionar un entorno sencillo, con unos pasos a seguir, ayuda instantánea explicando la funcionalidad de los atributos y como utilizarlos, y facilidad para guardar las reglas y compilarlas.

6.2.2 **Tareas que debe desempeñar nuestra aplicación.**

La estructura de una regla simple consta de seis partes importantes:

- Establecimiento de constantes que corresponderán a valor o rango de valores, de tal forma que el usuario le sea más fácil de recordar, y no tener que memorizar o escribir el rango de valores anteriores.
- Establecimiento de condiciones de emparejamiento de atributos, salvando o ignorando estos atributos si se cumplen las condiciones.
- Establecimiento del formato del fichero de flujo, indicando que atributos consideramos importantes para el posterior estudio de este fichero.
- Indicamos si queremos que en el fichero de reglas nos aparezcan estadísticas sobre paquetes perdidos, mandados, utilización de la CPU, ...
- Una vez que hemos realizado los pasos anteriores procedemos a guardar los datos en un fichero con extensión '.srl', en el que se reflejaran todos los pasos en lenguaje SRL.
- Compilar el fichero '.srl' creando un fichero '.rules' para que lo entienda el PME del medidor explicado en la sección 2.3.1.

6.2.3 **Problemas que se plantearon al inicio de la realización de la aplicación.**

Al comienzo de la realización de la aplicación se nos plantearon una serie de problemas típicos en la realización de cualquier proyecto software. Estas soluciones deberían responder a las siguientes cuestiones:

- ¿Qué y cuántos pasos deberíamos seguir a la hora de realizar una regla?
- ¿Cómo realizaríamos dichos pasos?
- ¿Qué forma es la más intuitiva para el usuario?
- ¿Cómo organizaríamos la estructura de clases de nuestra aplicación?
- ¿Qué mensajes se pasarían entre ellas?
- ¿Qué lenguaje programación deberíamos usar? ¿Por qué?

Para que nuestra aplicación grafica fuera portable a cualquier sistema hardware y software utilizamos el lenguaje de programación JAVA (del cual podemos encontrar información en las referencias [5], [6] y [7]), este nos daba la oportunidad de crear ficheros de reglas tanto en sistemas operativos Unix y Windows, esto nos lleva a otro problema que es que las reglas solo se pueden compilar en unix, esta será la única opción que en Windows no funcionará.

Para que nuestra aplicación gráfica fuera más intuitiva y sencilla para el usuario deberíamos colocar tantos pasos como partes tiene un fichero de reglas ".srl" explicadas en el apartado anterior. Los tres primeros pasos generarán el código srl que se ira añadiendo a un fichero temporal, el código generado por cada paso se añadirá al finalizar éste, de tal manera que no se podrá modificar desde dentro de la aplicación. Los siguientes tres pasos descritos anteriormente se añaden en forma de botones que se pondrán activos una vez realizados los tres primeros pasos.

Para que la organización de la aplicación sea lo suficientemente intuitiva para el usuario, se creo de la siguiente forma:

- Se crea un paso llamado "Define" que se encargara de generar constantes relacionadas con la red a estudiar.
- Se crea un paso llamado "Condiciones" que genera las condiciones o reglas propiamente dichas para el filtrado del flujo de datos.
- Se crea un ultimo paso denominado "Formato" que sirve para indicar que atributos estarán presentes en el fichero de flujo.

6.2.3.1 Defines

La representación gráfica final para este paso es la de la Figura 6-1. Esta ventana tiene dos botones principales, el botón "Nuevo" se utiliza para añadir una nueva constante a la lista que se encuentra en el área de texto. En el ejemplo que nos proporciona la figura el ultimo añadido es "Resto" que simboliza las redes cuatro, cinco y seis. El otro botón es "Aceptar" que

se encarga de añadir todas las constantes definidas en el área de texto en el fichero temporal, y activa el siguiente paso.

El JCheckBox "Solo tráfico IP" se utiliza para limitar el trafico filtrado solamente a tráfico IP, descartando otros protocolos de red. Mientras que el segundo JCheckBox nos habilita el área para poder modificar las constantes en caso de error.

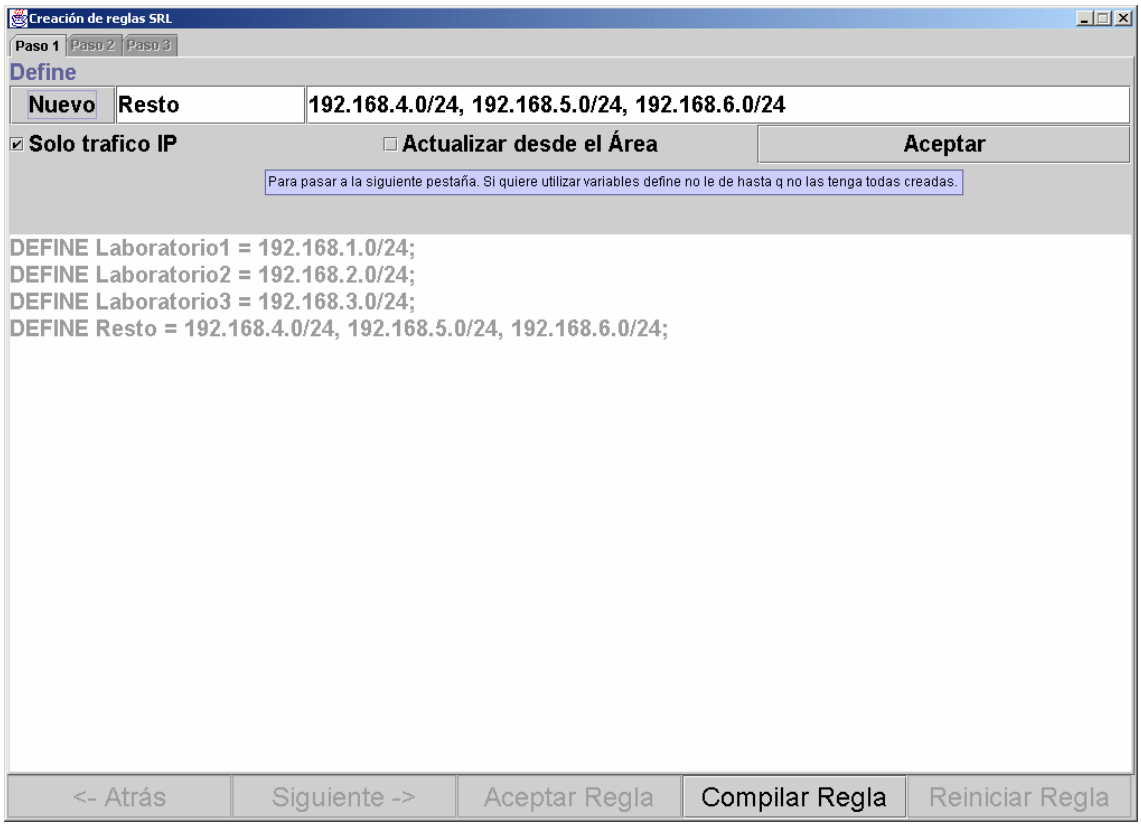


Figura 6-1.

6.2.3.2 Condiciones

En este segundo paso queda representado como vemos en la Figura 6-2. donde se realizaran las condiciones de filtrado. Este paso es el más importante y más complicado de entender por el usuario ya que es necesario tener nociones básicas sobre temas de redes y así poder comprender los atributos de los flujos con los que estamos tratando. En esta ventana encontramos las siguientes partes en orden descendente:

En una primera parte nos encontramos las condiciones propiamente dichas, y mediante los objetos "JComboBox" elegimos los atributos por los que queremos filtrar (SourcePeerAddress, SourceTransAddress, ...) y con los de al lado podemos escoger las constantes definidas en el paso anterior (Laboratorio1, Laboratorio2, …). En la parte siguiente encontramos "JCheckBox" (SAVE …) con los atributos que queremos salvar de los flujos que cumplen la condición, estos atributos son los mas importantes y mas utilizados.

Una vez realizado estos pasos aceptamos la condición con el botón "Aceptar" colocado en la tercera parte de esta ventana, y ya podemos pasar al siguiente paso.

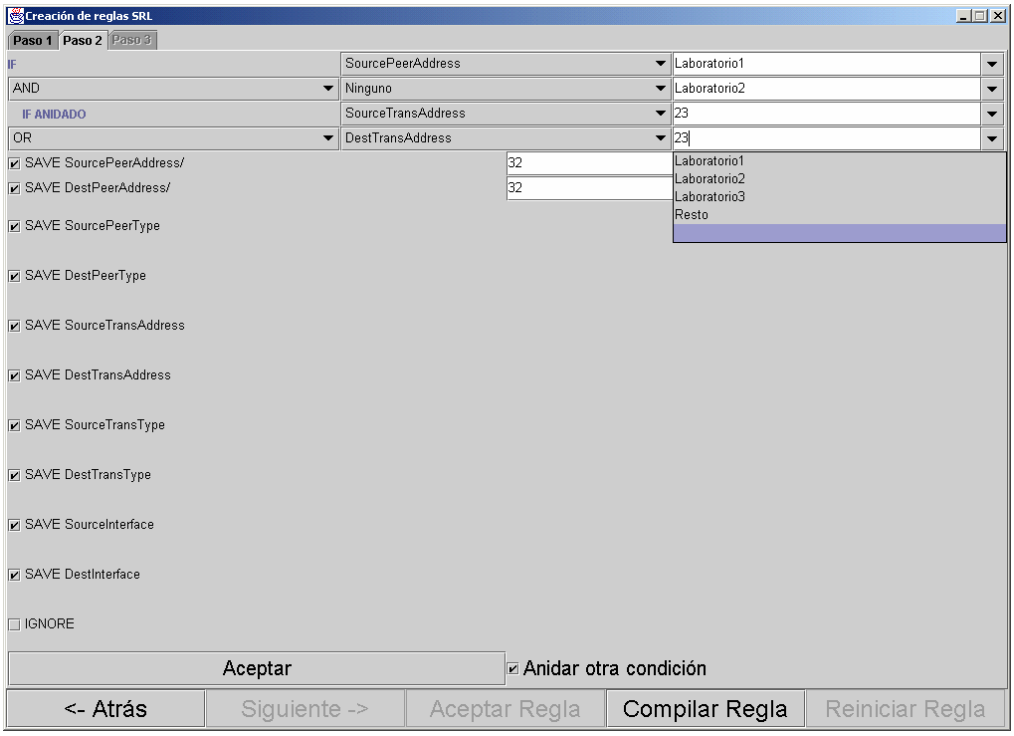


Figura 6-2.

6.2.3.3 Formato

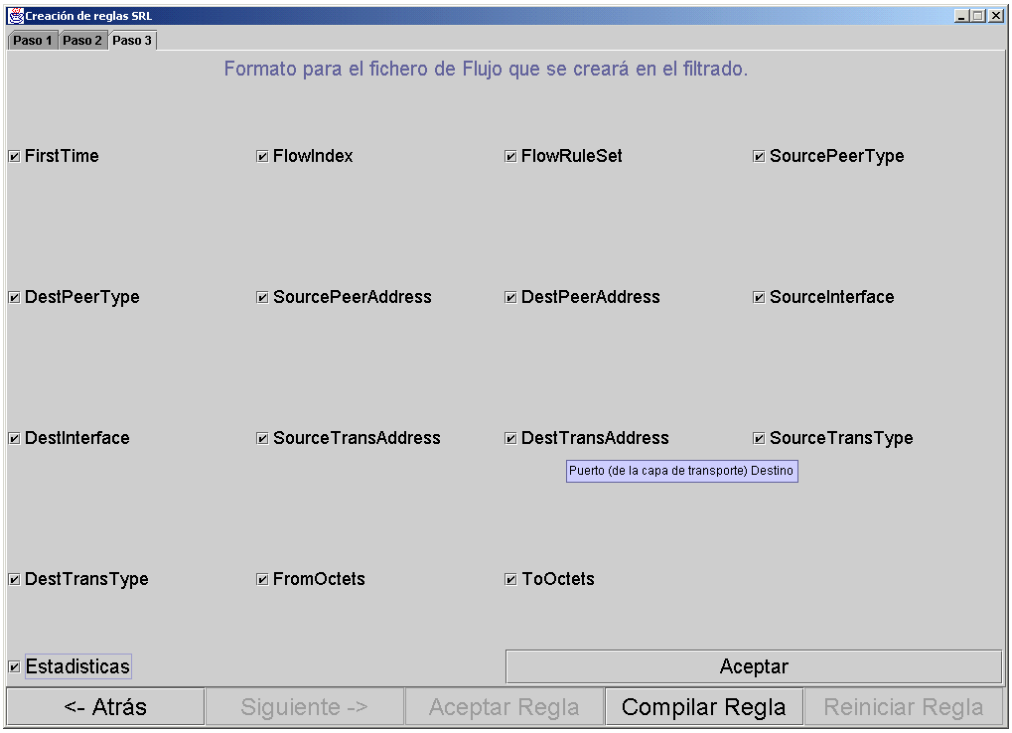


Figura 6-3.

En este último paso solamente tenemos que elegir mediante los "JCheckBox" los atributos que queremos que nos salga en el fichero de flujo, y hacer clic en el botón "Aceptar". Una vez hecho este último paso nuestra regla estará completamente creada, ahora solo tendremos que guardarla. Para guardar la regla hacemos clic en el botón "Aceptar Regla".

6.2.4 **Clases de la aplicación.**

Una vez que sabemos lo que tiene que hacer nuestra aplicación la descomponemos en cinco clases cada ellas en un fichero distinto ".java", estas clases se comunican entre ellas por medio de métodos que explicaremos mas adelante. Las clases son las siguientes:

- *Reglas.java:* Esta es la clase principal de nuestra aplicación. En ella se crearan referencias al resto de las clases para el envío de mensajes entre ellas. Esta clase se encarga de crear un JFrame en el cual se añadirá las otras clases en forma de paneles dentro de un "JTabbedPane" como se puede ver en la Figura 6-4.. Utilizando este sistema de clases mejoramos la legibilidad, la extensibilidad y posible reutilización del código. Esta clase también es la encargada de salvar el fichero de reglas con la ruta y el nombre que el usuario cree oportuno, y encargada también de la opción de compilar dicha regla ejecutando el compilador "srl" y creando el fichero ".rules" necesario para cargarlo en el medidor.

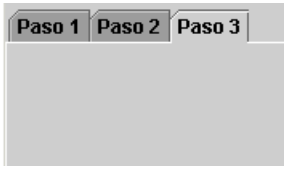


Figura 6-4.

- *Defines.java:* Esta clase que extiende de un JPanel (al igual que las dos siguientes *Condiciones.java y Formato.java*), se añade como un nuevo panel dentro del JTabbedPane de la clase principal. Esta clase es la encargada de generar las constantes que identifican la red y las guardará en un fichero de reglas temporal.
- *Condiciones.java:* Esta clase se utiliza para ir creando una a una las condiciones de los ficheros de reglas, y conforme se están creando se van añadiendo al fichero temporal. Una vez que cada condición ha sido escrita en el fichero de reglas temporal, ésta ya no se puede modificar mediante la aplicación y hay que hacerlo directamente a código.

- *Formato.java:* La clase formato se utiliza para generar el formato del fichero como hemos dicho anteriormente, este se añade al final del fichero. Además incluye la opción de meter estadísticas correspondientes al medidor en donde están cargadas las reglas.
- *GestionArchivos.java:* Mediante esta clase podemos gestionar los archivos para la generación de las reglas. Esta clase es de código reutilizado de un proyecto anterior presentado por Sergio Almagro Carrión (Ingeniero Técnico de Telecomunicación, especialidad Telemática).

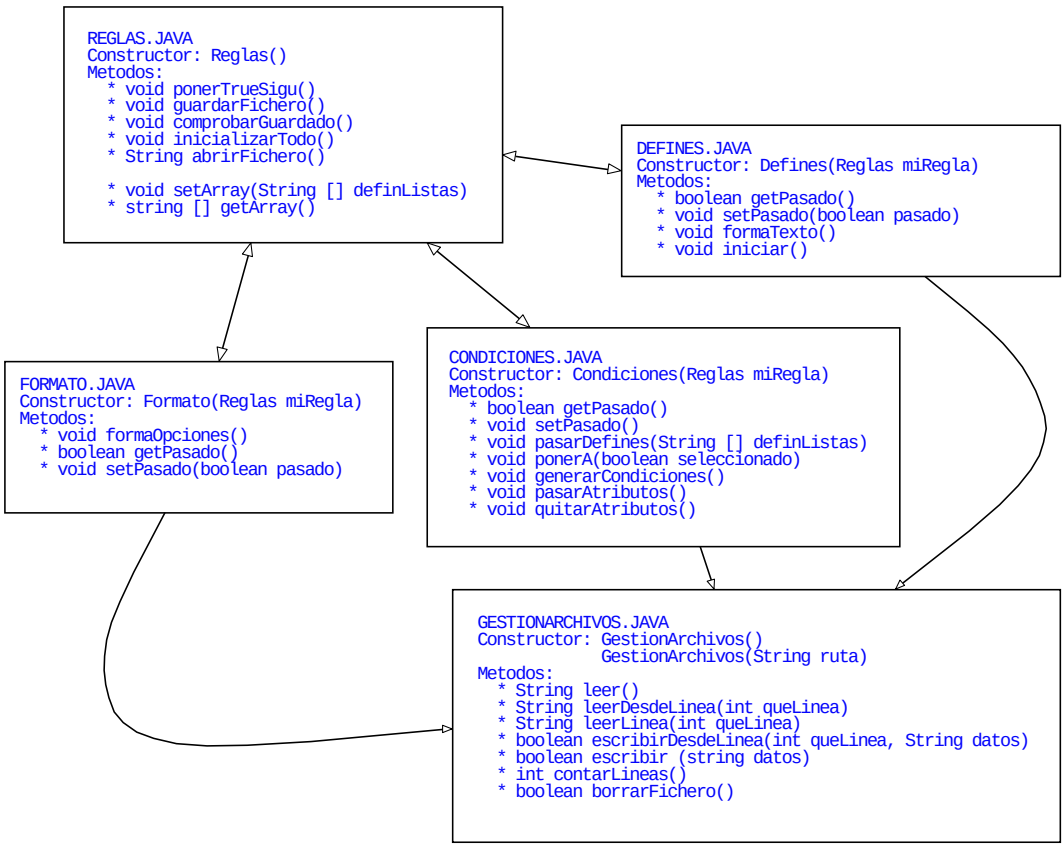


Figura 6-5: Estructura de las clases

En la Figura 6-5 se muestra la estructura de clases que posee la aplicación, y todos los métodos de cada clase que detallaremos en la sección 5.3. Como podemos observar la clase principal es Reglas.java, esta clase crea tres objetos a los cuales les pasa en el constructor su propia clase. Estos tres objetos son declarados como:

private Defines define = new Defines(this);

private Condiciones cond = new Condiciones(this);

private Formato forma = new Formato(this);

De esta forma podemos intercambiar mensajes entre los objetos utilizando la clase principal.

Una última clase llamada GestiónArchivos.java es referenciada por las clases Defines, Condiciones y Formato para que estas puedan trabajar con ficheros. Esta clase ha sido recuperada de otro proyecto, y se le han realizado modificaciones para adaptarla a nuestra aplicación. Estos cambios realizados se han basado en la eliminación de mensajes de error que lanzaba la clase cuando se producían excepciones. Ahora estos errores son tratados pos la clase que cree un objeto GestionArchivos.

6.3 ORGANIZACIÓN DE LAS CLASES Y FUNCIONAMIENTO.

6.3.1 Clase Reglas.java.

En esta clase podemos encontrar dos tipos de métodos, unos locales para la misma clase y otros que son utilizados por las clases Define y Condiciones. Las clases son las siguientes:

6.3.1.1 public ponerTrueSigu()

Esta clase la utilizamos en el resto de las clases cuando pulsamos el botón aceptar de cada pestaña, y lo que hace es poner activa la siguiente pestaña y el botón siguiente. De esta forma limitamos lo errores y hacemos que la creación de reglas se haga siempre por pasos, lo que hace que sea mas intuitiva para el usuario. También modifica los botones aceptar regla y siguiente si la pestaña es la última, inhabilitando el botón siguiente y habilitando el de aceptar regla.

6.3.1.2 private guardarFichero()

Este método lo utilizamos cuando hacemos click en el botón aceptar regla, en este momento se abre una ventana para indicar donde queremos guardar el fichero, y establecemos la ruta. Una vez hecho esto y teniendo la ruta, lo único que hacemos es renombrar el fichero temporal donde estamos creando la regla, poniéndole la extensión '.srl' y mandándolo a la ruta indicada. Este botón solo se activa cuando hemos empezado la creación de un fichero de reglas.

6.3.1.3 private comprobarGuardado()

Este método comprueba que se han guardado el conjunto de reglas en un fichero simplemente observando si existe el fichero temporal, si este fichero existe nos pregunta si deseamos guardar las reglas. Si este fichero no existe es porque ya se guardaron o bien que no hemos empezado a trabajar con la aplicación.

6.3.1.4 private inicializarTodo()

Este método se utiliza para reiniciar la aplicación, de tal modo que en caso de error, se pueda empezar de nuevo paso a paso sin necesidad de reiniciar la aplicación.

6.3.1.5 private String abrirFichero()

El método abrir fichero lo único que hace es devolver la ruta donde esta el fichero con el que queramos trabajar.

6.3.1.6 public setArray(String [] defineListas)

Este método lo utiliza la clase Defines para pasar las constantes que se han definido, de esta forma la clase Condiciones puede acceder a ellas.

6.3.1.7 public String [] getArray()

Este método lo utiliza la clase Condiciones para acceder a las constantes que se han definido antes en la clase Defines.

6.3.1.8 Eventos.

En la clase principal es en la que más tratamiento de eventos se hace, los botones a los que añadimos un escuchador de eventos "ActionListener" son los siguientes:

6.3.1.8.1 Botón "Siguiente":

Este botón lo utilizamos para pasar a la siguiente carta (siguiente paso), esto también se puede realizar directamente desde la pestaña de la carta. Cuando hacemos clic en este botón, comprueba en que paso se encuentra y pasa al siguiente paso, habilitando los botones necesarios para moverte solo entre los pasos ya realizados.

6.3.1.8.2 Botón “Atrás”:

Este botón al igual que el de siguiente tiene el mismo cometido, de esta forma es más intuitivo moverte entre los distintos pasos de la aplicación.

6.3.1.8.3 Botón “Aceptar Regla”:

Mediante este botón justificamos que la regla, se ha completado y pasamos a guardarla, abriendo una ventana con la estructura de directorios mediante la cual indicamos a la aplicación donde queremos salvar el fichero de reglas.

6.3.1.8.4 Botón “Compilar Regla”:

Utilizando este botón compilamos el fichero de reglas solamente si estamos en sistemas operativos Linux, ya que solo esta preparado para esta plataforma.

6.3.1.8.5 Botón “Reiniciar Regla”:

Este botón se utiliza para volver a generar una nueva regla, activando a desactivando los botones para dejarlo como si acabásemos de ejecutar la aplicación.

6.3.1.8.6 Botón “Cerrar Ventana”:

Al hacer click en el botón típico para cerrar la ventana, el programa comprueba si has guardado correctamente el fichero de reglas que estabas creando, si no lo habías guardado la aplicación te avisa con un mensaje, dándote la opción de guardar la regla. Si la regla había sido guardada, la aplicación se cierra normalmente.

También añadimos un escuchador de eventos del tipo “ChangeListener” este lo utilizaremos para el panel de cartas, para activar o desactivar los botones de Siguiente y Atrás.

6.3.2 **Clase Defines.java**

6.3.2.1 public boolean getPasado()

Utilizamos este método para ver si ya hemos aceptado los defines, lo que quiere decir que ya están escritos en el fichero temporal, si no están escritos esta variable booleana estará a false por lo que las escribiremos en el fichero.

6.3.2.2 public void setPasado(bolean pasado)

Este método se utiliza para cambiar la variable de instancia a la que nos hemos referido en el apartado anterior.

6.3.2.3 public void formaTexto()

En este método lo único que hacemos es copiar en el fichero temporal el comienzo del fichero de reglas y las constantes declaradas en el área de escritura.

6.3.2.4 public void iniciar()

El método iniciar es llamado desde la clase principal Reglas para volver a empezar de cero. Borra todas las constantes y pone todas las opciones como si acabásemos de ejecutar la aplicación, para volver a crear la regla paso por paso.

6.3.2.5 Eventos.

En esta clase solo encontramos del tipo ActionListener, para manejar los posibles eventos en los siguientes botones:

6.3.2.5.1 Botón “Aceptar”.

Mediante este botón aceptamos todas las constantes declaradas, este debe de hacerse click cuando estamos seguros que ya hemos terminado con las constantes, en este momento se las constantes añaden al fichero temporal y ya no se pueden modificar. Una vez hecho esto le decimos a la clase principal que ponga activo el botón siguiente, y la siguiente pestaña para pasar al otro paso.

6.3.2.5.2 Botón “Nuevo”.

Este botón añade la constante creada en las barras de texto al área de texto, presentándolas tal y como aparecerán en el fichero de reglas, de esta forma se podrán modificar en caso de error antes de añadirlas al fichero.

6.3.2.5.3 JCheckBox “Actualizar desde el Área”.

Mediante esta opción podemos hacer que el área de texto sea editable o no, para evitar que se añadan errores que puedan perjudicar la compilación de estas reglas.

6.3.3 **Clase Condiciones.java**

6.3.3.1 public boolean getPasado()

Utilizamos este método para la misma utilidad que en la clase anterior. La clase regla comprueba con este método si ya puede pasar a la pestaña de Formato.

6.3.3.2 public void setPasado(boolean pasado)

Este método lo utilizamos para inicializar la aplicación y poner la variable de instancia a false. Este método se utiliza desde la clase en el evento del botón de reiniciar reglas.

6.3.3.3 public void ponerA(boolean seleccionado)

Este método local se utiliza para poner todos los 'save' de forma que no se puedan editar, imponiendo la regla de ignorar los paquetes con esa condición.

6.3.3.4 public void generarCondiciones()

Este método se llama desde el evento del botón aceptar, y lo que hace escribe la condición en el fichero temporal según las opciones que se han impuesto.

6.3.3.5 public void pasarAtributos()

Este método coge los atributos que la clase Define ha pasado a la clase principal, y los coloca en los atributos a elegir en el 'JComboBox'.

6.3.3.6 public void quitarAtributos()

Este método solo lo utiliza la clase principal cuando queremos reiniciar las reglas, de forma que borra todos los atributos del 'JComboBox'.

6.3.3.7 Eventos.

En esta clase nos encontramos dos tipos de escuchadores de eventos; ActionListener y MouseListener. El primero se utiliza para lo mismo que en las clases anteriores mientras que el segundo lo utilizamos para controlar los eventos del ratón.

6.3.3.7.1 Botón "Aceptar"

Mediante este botón aceptamos la nueva condición que hemos creado y la escribimos en el fichero de reglas. Una vez hecho esto le decimos a la clase principal que ponga activo el botón siguiente, y la sucesiva pestaña de las cartas.

6.3.3.7.2 JCheckBox “Ignore”

Mediante esta opción lo único que hacemos es que ignore los atributos seleccionados para ser salvados e ignore los paquetes que contengan esas condiciones.

6.3.3.7.3 JComboBox “Operación Lógica”

Eligiendo la opción de solo un atributo en la operación lógica, solo se utilizara una condición, sin embargo las operaciones lógicas AND u OR habilitan el segundo atributo para poder ser usado en la condición.

6.3.3.7.4 JComboBox “Atributo 1” o “Atributo 2”

Este evento se utiliza para evitar que se pongan los mismos atributos en los dos JComboBox, sacando un mensaje de error por pantalla en el caso de que esto sucediera.

6.3.4 **Clase Formato.java**

6.3.4.1 public void formaOpciones()

Mediante este método creamos el formato del fichero de flujo dependiendo de los atributos marcados y lo añadimos al fichero de reglas.

6.3.4.2 public boolean getPasado()

Este método al igual que los anteriores es para asegurarnos que ya se ha escrito la línea de formato en el fichero de reglas, y se llama desde la clase principal.

6.3.4.3 public void setPasado(boolean pasado)

Al igual que en las clases anteriores este método se utiliza para reiniciar las reglas.

6.3.4.4 Eventos.

Esta última clase solo implementa el escuchador de eventos ActionListener. Con el que solamente se escucha el evento del botón ‘Aceptar’, al cual cuando se le hace click escribe la línea de formato (y las estadísticas si se han seleccionado) en el fichero de reglas.

6.4 Ejemplo de generación de un fichero de reglas.

Ahora nos disponemos a realizar una demostración de la utilización de la aplicación, para la creación de una regla. Para hacerlo más real suponemos el laboratorio de ingeniería Telemática IT5 como se muestra en la Figura 6-6, con dirección IP privada 192.168.5.X y con salida a Internet por medio de un router con dirección IP pública 212.128.24.34 haciendo NAT (Network Address Translation).

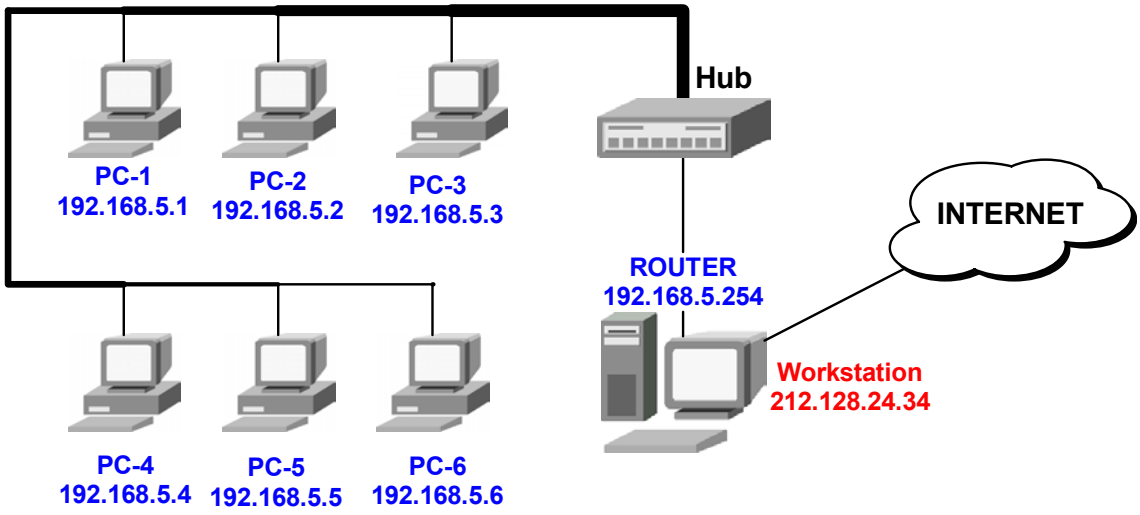


Figura 6-6.

Una vez descrita nuestra red, nos disponemos a generar nuestra regla. Queremos hacer una sola regla, que nos mida el tráfico FTP interno de la red privada, es decir paquetes con origen y destino 192.168.5.0/24 con destino el puerto FTP, queremos medir paquetes HTTP (puerto 80) que entren a la red privada y TELNET (puerto 23) que salgan o entren al servidor que hace de router. El medidor estará observando la interfaz con dirección privada 192.168.5.254.

Al ejecutar la aplicación GARNE obtenemos el primer paso para generar nuestra regla. En este primer paso definimos las constantes necesarias que nos describa nuestra red. Las constantes definidas deben siempre empezar por una letra seguida de cualquier carácter alfanumérico, esto debe ser así ya que sino el compilador detectaría un error en la línea. Si en una constante se utiliza alguna palabra reservada como FTP o Telnet el compilador avisa con un mensaje “Warning”.

La descripción de la red por medio de las constantes, la podemos observar en la Figura 6-7.

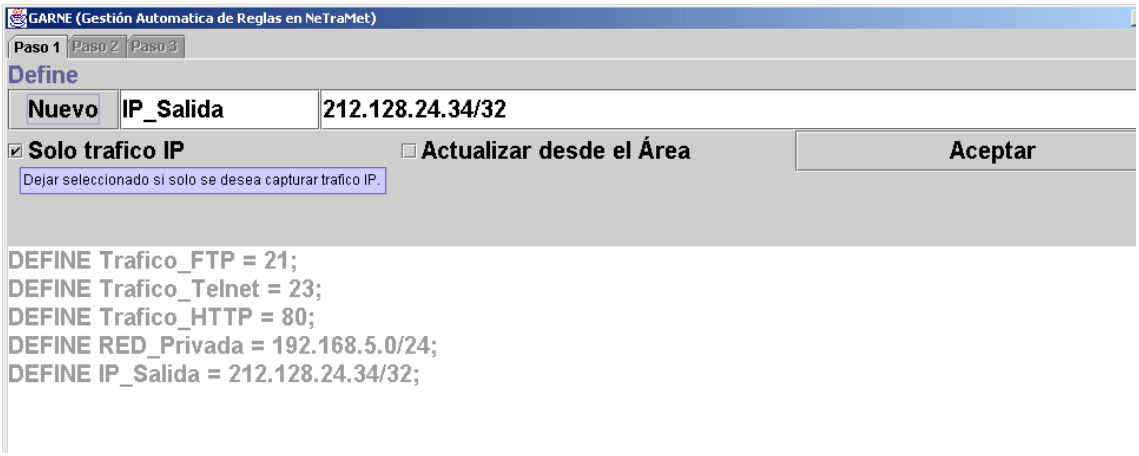


Figura 6-7.

Para ir añadiendo una a una las constantes, primero escribimos los valores en los cuadros de textos superiores al lado del botón "Nuevo", y una vez que hemos terminado hacemos click en el botón "Nuevo" y este nuevo define se almacena en el Área de Texto (que en un principio se encuentra inhabilitada para escribir en ella). Si al terminar nos encontramos un error en las constantes que hemos definido podemos corregirlo activando el Área de Texto, esta área se activa seleccionando el "JCheckBox Actualizar desde Área". Una vez que ya estamos seguros que todos los defines están correctamente escritos (nos referimos a que describan nuestra red) pulsamos el botón aceptar. Al pulsar el botón aceptar nos pregunta la confirmación de guardar las constantes en el fichero ya que una vez guardadas no de pueden modificar. Figura 6-8.

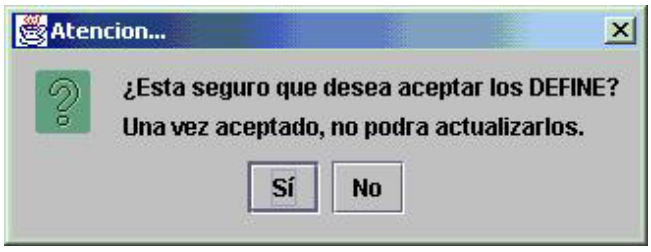


Figura 6-8.

Una vez hecho esto se nos activa la siguiente pestaña y el botón siguiente que se encuentra en la parte inferior de la aplicación. Ahora podemos acceder a la siguiente pestaña para la realización de las condiciones de la regla (nos disponemos a realizar las reglas propiamente dichas).

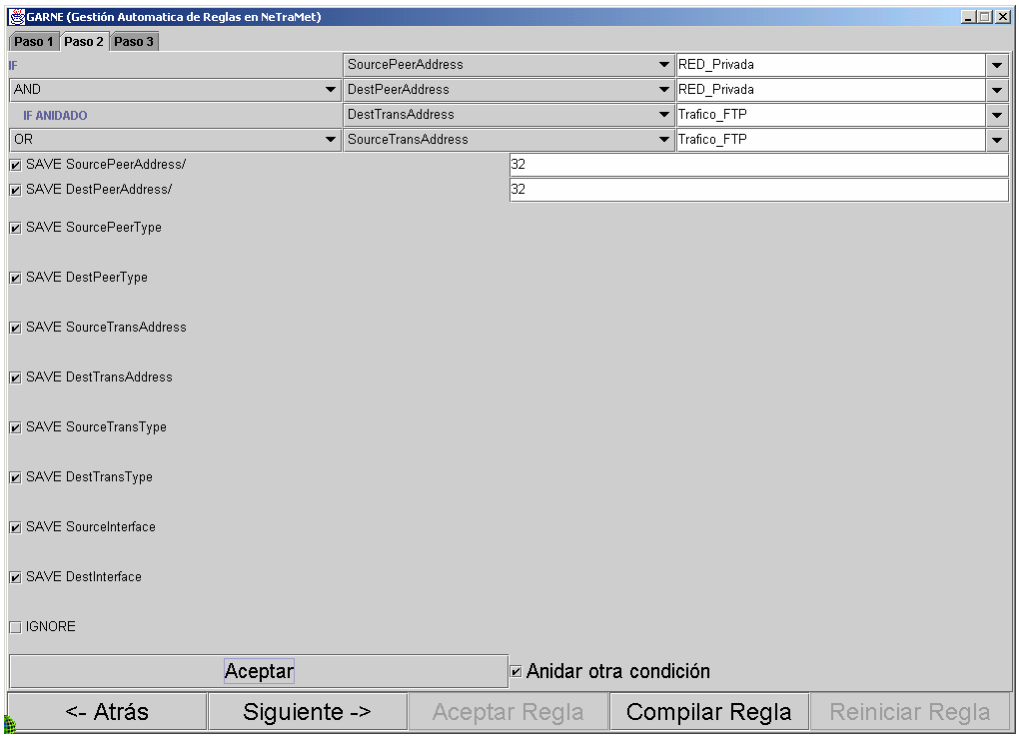


Figura 6-9.

Desde un principio marcaremos todos los JCheckBox "SAVE …" ya que nos puede interesar salvar todos los atributos, como podemos observar en la Figura 6-9. A continuación vamos creando las condiciones y si es necesario podemos anidar dos condiciones como en la figura anterior. Podemos observar que en los JCheckBox que se encuentran editables, podemos seleccionar las constantes declaradas en el paso anterior, como se muestra en la Figura 6-10 y 6-11.

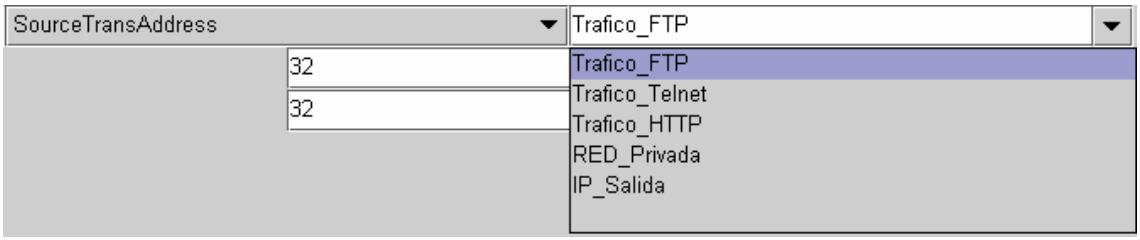


Figura 6-10.

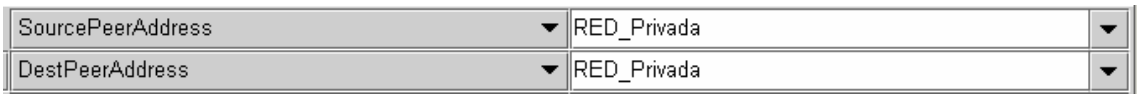


Figura 6-11.

De la misma forma vamos completando nuestra regla con el resto de las condiciones de filtrado, como la regla para HTTP y Telnet.

Con cada una de las reglas debemos de ir aceptándolas para que se añadan al fichero temporal que se va creando. Una vez realizadas las reglas ya podemos pasar a la siguiente pestaña que será la del formato del fichero de flujo. Esta se puede ver en la Figura 6-12 con las opciones ya marcadas.

Figura 6-12.

Una vez que le damos a aceptar se nos activa el botón para poder guardar la regla en un fichero srl, este botón es "Aceptar Regla", al pinchar nos sale un menú para guardar el fichero en el directorio que nosotros deseemos tal y como se nos muestra en la Figura 6-13.

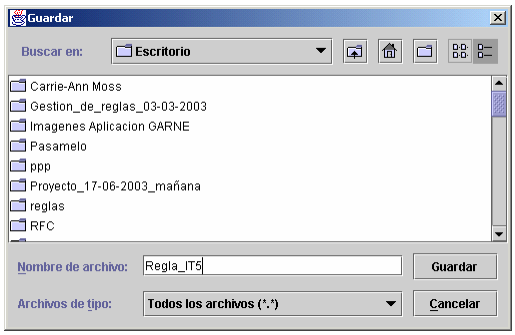


Figura 6-13.

Una vez guardado ya podemos compilar el fichero, para ello utilizamos el botón "Compilar Regla", esto nos abrirá una ventana como la de la figura 6-13 y desde ahí podremos elegir el fichero ".srl" que queremos compilar. Esto nos generara un fichero ".rules" que ya podrá ser cargado al medidor y dejara el fichero srl por si se desea hacer una modificación manual del mismo.

NOTA: Para poder compilar una regla es necesario esta ejecutando la aplicación bajo sistema operativo Linux, ya que en windows no está el compilador.

El fichero de reglas '.srl' generado es el siguiente:

```
DEFINE Trafico_FTP = 21;
DEFINE Trafico_Telnet = 23;
DEFINE Trafico_HTTP = 80;
DEFINE RED_Privada = 192.168.5.0/24;
DEFINE IP_Salida = 212.128.24.34/32;

IF (SourcePeerType == IP) || (DestPeerType == IP) SAVE;
ELSE IGNORE;

IF ((SourcePeerAddress == RED_Privada) && (DestPeerAddress == RED_Privada))
          IF ((DestTransAddress == Trafico_FTP) && (SourceTransAddress == Trafico_FTP))
          {
                    SAVE SourcePeerAddress/32;
                    SAVE DestPeerAddress/32;
                    SAVE SourcePeerType;
                    SAVE DestPeerType;
                    SAVE SourceTransAddress/16;
                    SAVE DestTransAddress/16;
                    SAVE SourceTransType;
                    SAVE DestTransType;
                    SAVE SourceInterface;
                    SAVE DestInterface;
                    COUNT;
          }
IF (SourceTransAddress == Trafico_HTTP)
          {
                    SAVE SourcePeerAddress/32;
                    SAVE DestPeerAddress/32;
                    SAVE SourcePeerType;
                    SAVE DestPeerType;
                    SAVE SourceTransAddress/16;
                    SAVE DestTransAddress/16;
                    SAVE SourceTransType;
                    SAVE DestTransType;
                    SAVE SourceInterface;
                    SAVE DestInterface;
                    COUNT;
          }
IF ((SourceTransAddress == Trafico_Telnet) || (DestTransAddress == Trafico_Telnet))
          {
                    SAVE SourcePeerAddress/32;
                    SAVE DestPeerAddress/32;
                    SAVE SourcePeerType;
                    SAVE DestPeerType;
                    SAVE SourceTransAddress/16;
                    SAVE DestTransAddress/16;
                    SAVE SourceTransType;
                    SAVE DestTransType;
                    SAVE SourceInterface;
                    SAVE DestInterface;
                    COUNT;
          }
```

```
SET 2;
FORMAT FlowIndex FlowRuleSet FirstTime SourcePeerType DestPeerType SourcePeerAddress DestPeerAddress SourceInterface
DestInterface SourceTransAddress DestTransAddress SourceTransType DestTransType FromOctets ToOctets;
```

Una vez copilado este fichero de reglas mediante la aplicación obtenemos el siguiente fichero ".rules" que ya podrá ser cargado en el medidor.

```
#Source file: Regla_IT5.srl
#Compiled by: SRL compiler, version 4.3
#Time:        12:56:17 Sat  1 May 2004
  sourcepeertype & 255.0 = 1.0: pushto, g1;
  destpeertype & 255.0 = 1.0: pushto, g1;
  null & 0 = 0: gotoact, n1;
n1:
  null & 0 = 0: ignore, 0;
g1:
  sourcepeeraddress & 255.255.255 = 192.168.5: goto, s1;
  null & 0 = 0: goto, n2;
s1:
  destpeeraddress & 255.255.255 = 192.168.5: goto, a1;
n2:
g2:
  sourcetransaddress & 255.255 = 0.80: gotoact, a3;
g4:
  sourcetransaddress & 255.255 = 0.23: gotoact, a4;
  desttransaddress & 255.255 = 0.23: gotoact, a4;
  null & 0 = 0: gotoact, n5;
n5:
g5:
  null & 0 = 0: nomatch, 0;
a1:
  desttransaddress & 255.255 = 0.21: goto, s2;
  null & 0 = 0: goto, n3;
s2:
  sourcetransaddress & 255.255 = 0.21: gotoact, a2;
  null & 0 = 0: gotoact, n3;
n3:
g3:
  null & 0 = 0: goto, g2;
a3:
  sourcepeeraddress & 255.255.255.255 = 0.0: pushpkttoact, next;
  destpeeraddress & 255.255.255.255 = 0.0: pushpkttoact, next;
  sourcepeertype & 255.0 = 0.0: pushpkttoact, next;
  destpeertype & 255.0 = 0.0: pushpkttoact, next;
  sourcetransaddress & 255.255 = 0.0: pushpkttoact, next;
  desttransaddress & 255.255 = 0.0: pushpkttoact, next;
  sourcetranstype & 255.0 = 0.0: pushpkttoact, next;
  desttranstype & 255.0 = 0.0: pushpkttoact, next;
  sourceinterface & 255.0 = 0.0: pushpkttoact, next;
  destinterface & 255.0 = 0.0: pushpkttoact, next;
  null & 0 = 0: count, 0;
a4:
  sourcepeeraddress & 255.255.255.255 = 0.0: pushpkttoact, next;
  destpeeraddress & 255.255.255.255 = 0.0: pushpkttoact, next;
  sourcepeertype & 255.0 = 0.0: pushpkttoact, next;
  destpeertype & 255.0 = 0.0: pushpkttoact, next;
  sourcetransaddress & 255.255 = 0.0: pushpkttoact, next;
  desttransaddress & 255.255 = 0.0: pushpkttoact, next;
  sourcetranstype & 255.0 = 0.0: pushpkttoact, next;
  desttranstype & 255.0 = 0.0: pushpkttoact, next;
  sourceinterface & 255.0 = 0.0: pushpkttoact, next;
  destinterface & 255.0 = 0.0: pushpkttoact, next;
  null & 0 = 0: count, 0;
a2:
  sourcepeeraddress & 255.255.255.255 = 0.0: pushpkttoact, next;
  destpeeraddress & 255.255.255.255 = 0.0: pushpkttoact, next;
  sourcepeertype & 255.0 = 0.0: pushpkttoact, next;
  destpeertype & 255.0 = 0.0: pushpkttoact, next;
  sourcetransaddress & 255.255 = 0.0: pushpkttoact, next;
  desttransaddress & 255.255 = 0.0: pushpkttoact, next;
  sourcetranstype & 255.0 = 0.0: pushpkttoact, next;
  desttranstype & 255.0 = 0.0: pushpkttoact, next;
  sourceinterface & 255.0 = 0.0: pushpkttoact, next;
  destinterface & 255.0 = 0.0: pushpkttoact, next;
  null & 0 = 0: count, 0;
set 2;
format
  flowindex flowruleset firsttime sourcepeertype destpeertype
  sourcepeeraddress destpeeraddress sourceinterface destinterface
  sourcetransaddress desttransaddress sourcetranstype desttranstype fromoctets
  tooctets;
```

Capítulo 7

Conclusiones

Ante la realización de este proyecto se habían marcado como principales objetivos el estudio del funcionamiento de aplicaciones RTFM, y en especial el aprovechamiento de la plataforma NeTraMet que sigue esta filosofía RTFM. La explicación de RTFM se encuentra en el segundo capitulo, donde hacemos un resumen de sobre todo lo aparecido en los documentos RFC's publicados por la IETF.

Una adaptación de esta filosofía a una aplicación real se puede observar en la plataforma **NeTraMet**. Una vez planteado el problema existente con respecto a la gestión de redes, realizamos las primeras pruebas de estudio de tráfico con la plataforma NeTraMet, una aplicación útil, fiable y económica (debido a que es de libre distribución).

NeTraMet fue la primera implementación de un medidor con esas características (útil, fiable y económico), que desde su primera versión 2.0 disponible en Internet ya podíamos hacer mediciones completamente fiables y en tiempo real. Esta plataforma sigue una arquitectura similar (medidor, colector, manager y aplicación de análisis), salvo con una sutil diferencia, esta diferencia es que el colector y el manager están implementados en una única aplicación llamada NeMaC. La plataforma NeTraMet cuenta además con un nuevo medidor llamado NetFlowMet utilizado para capturar flujos de tráfico exportados por medio de la tecnología NetFlow de Cisco.

Una vez comprendida la filosofía RTFM y mas detalladamente la plataforma NeTraMet pasamos a estudiar el lenguaje SRL. Esta es una parte importante para el filtrado de flujo de tráfico que llega a cualquiera de nuestros medidores (NeTraMet o NetFlowMet). Un fichero ".srl" no es más que un programa que se ejecuta de forma secuencial en el PME del medidor cada vez que a este le llega un nuevo paquete. Mediante este programa el medidor decide que debe hacer con la información del paquete (guardarla en sus contadores o ignorarlo ya que no es de interés). Si el paquete es guardado el medidor almacena los atributos de la cabecera en una tabla de flujo.

Un problema que observamos durante el proyecto sobre la plataforma NeTraMet es la imposibilidad de controlar un mismo medidor (NetFlowMet) con varios managers para un mejor filtrado de tráfico utilizando distintas reglas. En ocasiones resultaría útil aplicar dos reglas

de filtrado distintas para obtener dos ficheros de flujos mas reducidos, lo que facilitaría su posterior estudio con otras aplicaciones. De aquí nos surgió la idea de realizar una aplicación que duplicara estos flujos para poder filtrarlos con medidores y reglas distintas.

Para la realización de la aplicación DUFNE nos basamos en una librería que nos facilitaba la plataforma NeTraMet, esta librería nos proporciona la estructura de los flujos que se exporta con NetFlow. La última versión de nuestra aplicación DUFNE se compone de dos aplicaciones, un servidor que se encarga de duplicar los flujos a los clientes que se han subscrito a el, y un cliente que se encarga de hacer las peticiones al servidor para que exporte el flujo.

Otro problema que observamos al analizar la plataforma NeTraMet es a la hora de realizar una regla de filtrado, esto es debido a que antes de poder hacer una regla es necesario comprender el lenguaje SRL, entender la sintaxis y como funciona cada uno de los atributos de los flujos. Esto nos dio la idea de realizar una aplicación que nos gestionara la realización de reglas de forma guiada y sencilla, añadiendo un manual que explicase el funcionamiento de la aplicación. El resultado ha sido el diseño de la aplicación GARNE.

Apéndice A:

Manual de Instalación y Utilización de la aplicación GARNE

1.Instalación de la Aplicación GARNE

Uno de los requisitos para la instalación de la aplicación GARNE es la existencia en el sistema operativo (ya sea Windows o Linux) de la maquina virtual java con JDK2 [6] también llamadas como **Java 2 Platform, Standard Edition (J2SE)**. Si esta versión no se encuentra instalada en nuestro sistema operativo podemos encontrarla junto con los ficheros de la aplicación, en dos directorios llamados "JKD-Windows y JDK-Linux". Para la instalación de la JDK2 podemos visitar la pagina http://search.java.sun.com/search/java/?qt=install en la que se nos detalla como debemos instalar los fichero, y que variables de entorno debemos modificar.

La aplicación GARNE no necesita ninguna instalación, solamente copiando el directorio GARNEv1 a nuestro disco duro podemos trabajar directamente con ella.

Este directorio se compone de un fichero "GARNE.zip" que contiene la aplicación ya compilada. Dentro de este fichero "zip" se adjunta también el código fuente de la aplicación por si se le desea hacer alguna modificación. Dentro de este directorio se encuentran los ficheros para ejecutar la aplicación tanto en Linux (GARNE) como en Windows (GARNE.bat).

Otro fichero que no nos puede faltar dentro del directorio GARNEv1 es el fichero "srl", ya que éste es el compilador del lenguaje SRL para Linux y sin el no podríamos compilar los ficheros ".srl". En el Sistema Operativo Windows este fichero no es necesario, ya que no se puede ejecutar.

2.Presentación de la Aplicación GARNE.

La aplicación se compone de dos partes principales, la primera parte son los botones de control de la aplicación, que se encuentran en la parte inferior de la ventana como se puede observar en la Figura 14.

Figura 14.

Estos botones gestionan la aplicación, de forma que nos podemos mover entre los distintos pasos mediante los botones "<-Atrás y Siguiente->", podemos aceptar la regla que hemos creado y guardarla en un fichero ".srl" mediante el botón "Aceptar Regla", compilar la regla que anteriormente habremos guardado mediante el botón "Compilar Regla" o reiniciar al estado inicial la aplicación para volver a crear un nuevo fichero de reglas evitando de este modo tener que reiniciar la aplicación, esto se hace mediante el botón "Reiniciar Regla".

NOTA: *Para la compilación de la regla es necesario que la aplicación se ejecute bajo un sistema operativo Linux, ya que el compilador que utiliza es de esta plataforma.*

Una segunda parte de la aplicación que se encuentra en el centro nos encontramos los pasos a seguir en el proceso de elaboración de un fichero de reglas. Este proceso se encuentra dividido en tres pasos indicados por unas pestañas como las de la Figura 15.

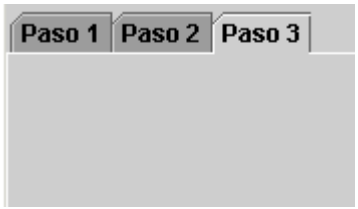


Figura 15.

- El primer paso se refiere a la elaboración de constantes que describen nuestra red y nos ayuda a trabajar con estas constantes en vez de direcciones IP, números de puertos (TCP o UDP), etc.
- El segundo paso se utiliza para la realización de las condiciones de filtrado del flujo, utilizando los atributos de las cabeceras y comparándolos con las constantes definidas en el paso anterior. Si se cumplen las condiciones salvara una serie de atributos indicados por el usuario.

- En el tercer paso diseñamos el formato del fichero de flujo que se creará después del filtrado con nuestra regla.

3.Pasos a seguir para la realización de un fichero de regla

Para la realización de un fichero de reglas tenemos que realizar los tres pasos descritos en el apartado anterior. Además iremos explicando todos los componentes que hay en cada paso.

3.1. Defines.

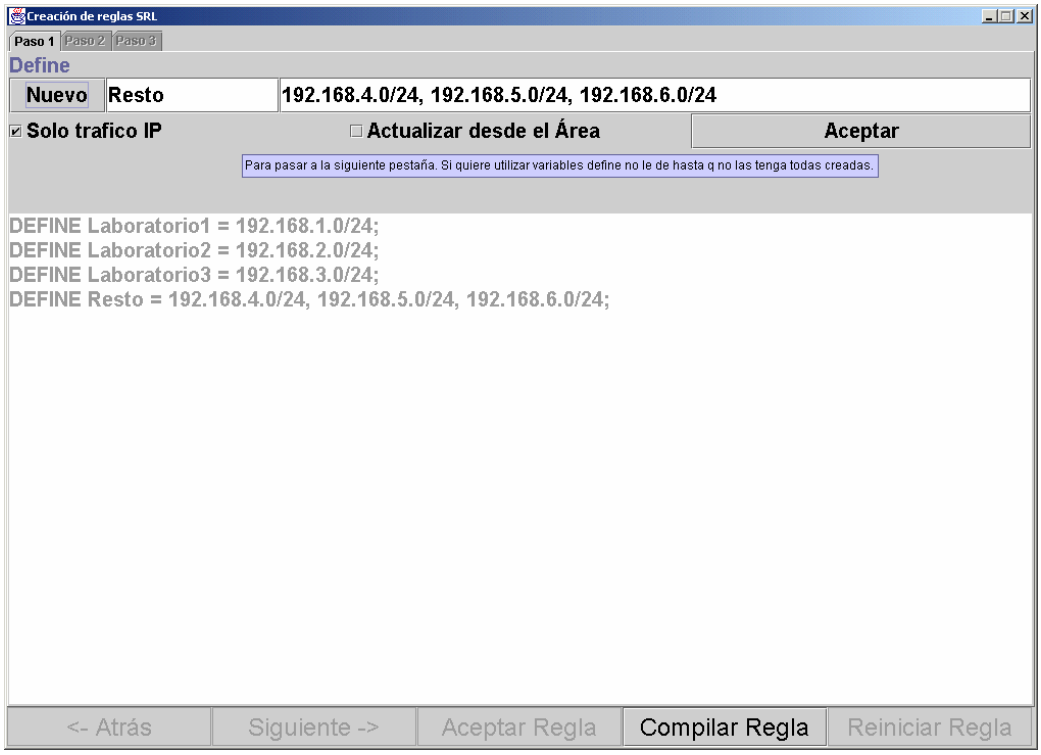


Figura 16

En la Figura 16 podemos observar este paso. En la parte superior de la ventana podemos encontrar dos pequeñas áreas de texto donde introduciremos el nombre de la constante y lo que define esa constante. Podemos ver más claramente estos elementos en la Figura 17.

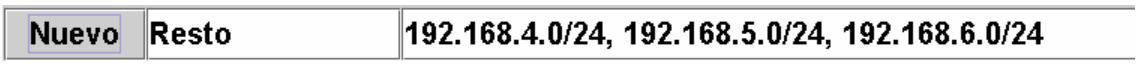


Figura 17

Cada vez que queramos añadir una nueva constante, primero escribiremos el nombre de dicha constante y los elementos que definen, una vez hecho esto, pinchamos en el botón "Nuevo" para añadirlo al área de Texto que contiene todas las constantes que hemos definido. Esta segunda área de texto podemos observarla en la Figura 18. En esta área de texto no se puede escribir por defecto, pero si queremos modificar algún valor de las constantes podemos hacerlo seleccionando el JCheckBox "Actualizar desde Área" que se encuentra en la Figura 19.

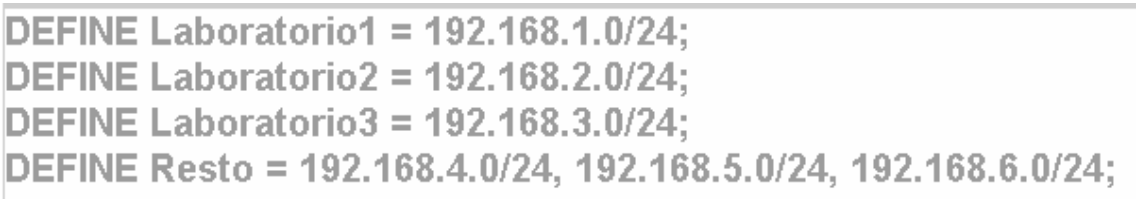


Figura 18

Como podemos observar tenemos añadidas cuatro constantes, las cuales se añadirán al fichero de regla cuando hagamos clic en el botón "Aceptar" que se muestra en la Figura 19.

Figura 19

En la Figura 19 podemos encontrar dos opciones, en la primera y que se encuentra marcada "Solo trafico IP" se utiliza para descartar todo el tráfico que no sea del tipo de protocolo de red IP, y la segunda opción se utiliza para modificar el área de texto como hemos comentado anteriormente.

Una vez que hemos aceptado nuestras constantes mediante el botón "Aceptar", nos aparece una ventana de alerta avisándonos que si las escribimos en el fichero ya no se podrán modificar mediante la aplicación. Esta ventana se muestra en la Figura 20.

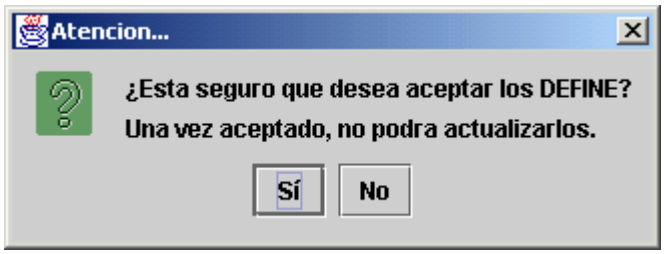


Figura 20

3.2. Condiciones.

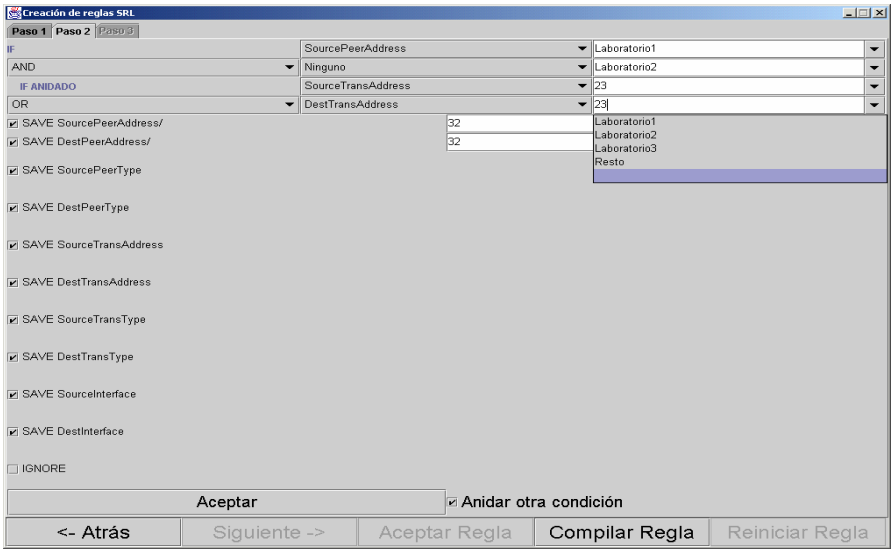


Figura 21

En la Figura 21 podemos observar el segundo paso llamado **Condiciones**. Este paso se utiliza para la creación de las reglas propiamente dicha, es decir las condiciones de filtrado de los flujos que pasan por el medidor.

En la parte superior de la ventana podemos encontrar las líneas donde desarrollaremos las condiciones de filtrado, los dos "IF" son idénticos salvo que el segundo puede esta anidado dentro del primero. Explicaremos el primer IF ya que el segundo es análogo.

En la Figura 22 tenemos el primer "IF". Mediante el JComboBox (ver Figura 23) podemos elegir una de las constantes definidas anteriormente en el primer paso o escribir el valor que necesitemos para realizar la condición de filtrado. Los JComboBox que queda justo al lado se utiliza para elegir el atributo de la cabecera del flujo que nos interesa y compararlo con el del la Figura 23. El JComboBox que esta en la esquina inferior izquierda de la Figura 22 se

utiliza para definir el operador lógico de la condición, de esta forma obligamos a que se cumplan las dos condiciones o solamente una de las dos definidas.

Figura 22

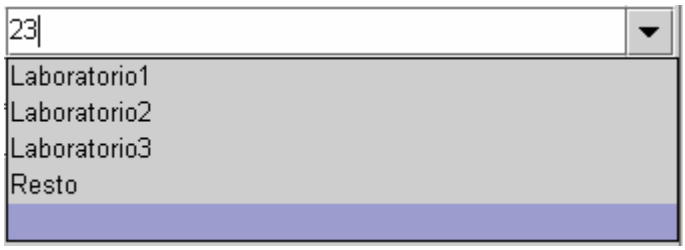


Figura 23

Seguidamente después de las dos condiciones podemos observar en la Figura 24 dos de todos los atributos que podemos salvar, nos hemos fijado en estos dos porque son los únicos que poseen un campo de texto a su derecha. Este campo de texto indica la mascara de red por un "1", de tal forma que salvará los bits de las direcciones origen y destino que se le indique por la mascara de red.

Figura 24

Por ejemplo: Si la dirección de origen del flujo es 192.168.1.20 pasados a bits sería **11000000.10101000.00000001.00010100**. Y si la mascara de red es 24 que se podría expresar también como 255.255.255.0 y en bits como **11111111.11111111 .11111111.00000000** el resultado de hacer la operación XOR sería **11000000.10101000.00000001.00000000.**

Cada vez que creemos una condición debemos pinchar el botón "Aceptar" de la Figura 25 para que esa condición se escriba en el fichero de reglas. Una vez que hemos pinchado en Aceptar la condición es imposible modificarla desde la aplicación, por lo que antes de aceptar la condición deberemos estar seguros de que esta bien realizada.

Figura 25

3.3. Formato.

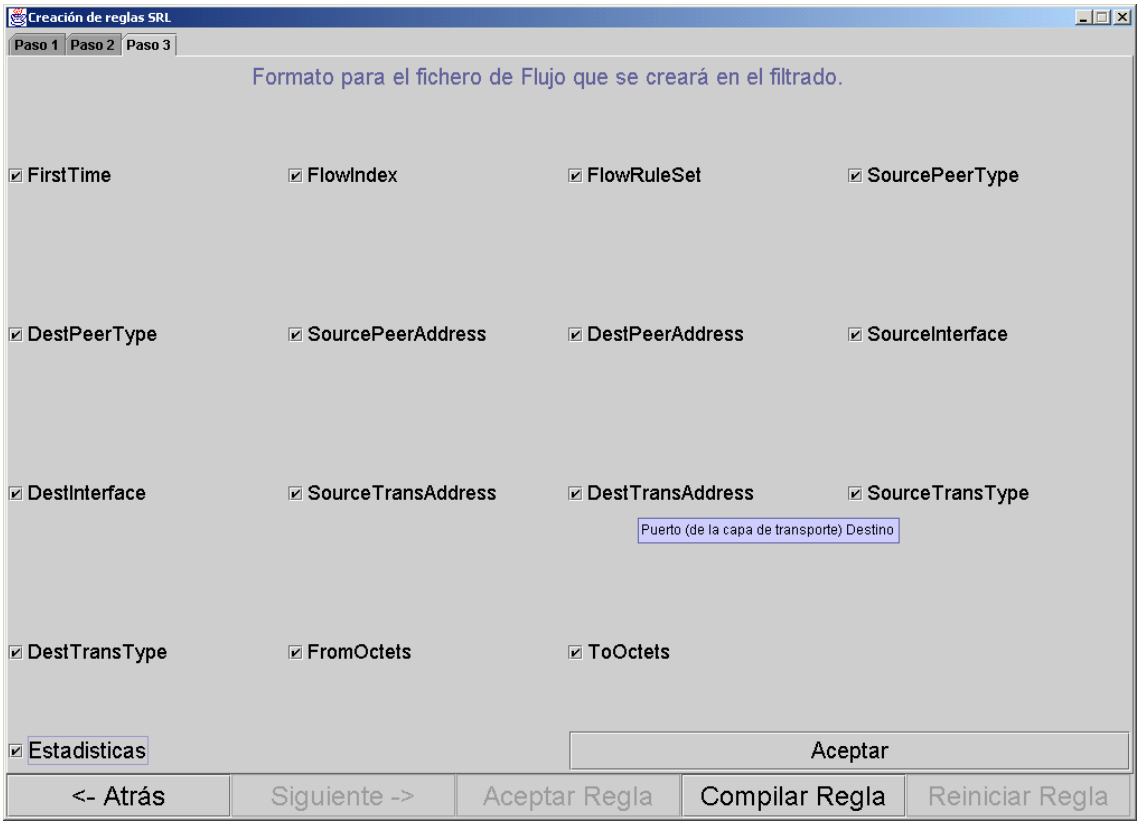


Figura 26

En la Figura 26 podemos observar el tercer paso para la realización de la reglas de filtrado. En este paso generaremos el formato del fichero de flujo que será creado por el colector para su posterior estudio con otras aplicaciones de análisis. A parte de definir el formato del fichero de flujo, podremos añadirle estadísticas sobre el medidor (paquetes q pasan por segundo, paquetes perdidos, …)

Una vez creadas las reglas pulsamos el botón “Aceptar Regla” que se ha mostrado en la Figura 14., y se nos abrirá una ventana con la estructura de directorios para poder elegir donde queremos salvar la regla como se muestra en la Figura 27. El nombre de la regla siempre se nos guardara por defecto con la extensión “.srl”, por lo que no es necesario escribir la extensión del fichero.

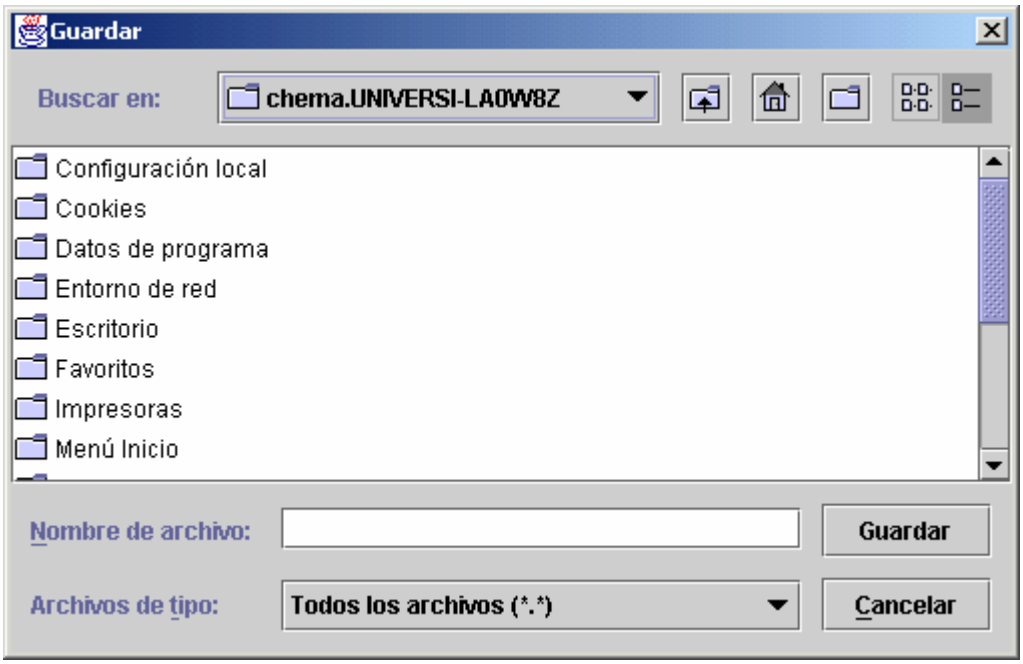


Figura 27

Apéndice B:

Código Fuente de la Aplicación GARNE

Este es el código JAVA de la aplicación GARNE, en el que detallaremos y explicaremos mediante comentarios cada una de sus clases y partes más importantes de la aplicación.

Reglas.java

```
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.lang.Runtime;

public class Reglas extends JPanel implements ActionListener, ChangeListener
{
  private final int tamanoLetra = 24;
//Les pasamos nuestra propia clase para que ellos puedan poner a true las cartas
  private Defines define = new Defines(this);
  private Condiciones cond = new Condiciones(this);
  private Formato forma = new Formato(this);
// Creamos el panel de cartas, los botones y paneles necesarios para la interfaz grafica
  private JTabbedPane cartas = new JTabbedPane();
  private JPanel botones = new JPanel();
  private JButton sigu = new JButton("Siguiente ->");
  private JButton atras = new JButton("<- Atrás");
  private JButton aceptarRegla = new JButton("Aceptar Regla");
  private JButton compilarSRL = new JButton("Compilar Regla");
  private JButton iniciarRegla = new JButton("Reiniciar Regla");

  private static final String ruta = "temp.srl";
// Nombre del fichero temporal donde se almacena la regla
  private JOptionPane optionPane; // Para las ventanas de errores
  private String [] defineListas;
// ################## Constructores ######################
  public Reglas()
  {
        aceptarRegla.setFont(new Font("aceptarRegla",4,tamanoLetra));
        sigu.setFont(new Font("sigu",4,tamanoLetra));
        atras.setFont(new Font("atras",4,tamanoLetra));
        compilarSRL.setFont(new Font("compilar",4,tamanoLetra));
```

```
        iniciarRegla.setFont(new Font("iniciar",4,tamanoLetra));

        cartas.add("Paso 1",define);
        cartas.add("Paso 2",cond);
        cartas.setEnabledAt(1,false); //Para desactivar la carta de condicion
        cartas.add("Paso 3",forma);
        cartas.setEnabledAt(2,false); //Para desactivar la carta de formato
// Insertamos los botones en el panel en 3 filas y 1 columna
        botones.setLayout(new GridLayout (1,4));
                atras.setEnabled(false);
                botones.add(atras);
                sigu.setEnabled(false);
                botones.add(sigu);
                aceptarRegla.setEnabled(false);
                botones.add(aceptarRegla);
                botones.add(compilarSRL);
                iniciarRegla.setEnabled(false);
                botones.add(iniciarRegla);
// insertamos las cartas y los botones en el panel principal
        setLayout(new BorderLayout());
                add(cartas,BorderLayout.CENTER);
                add(botones,BorderLayout.SOUTH);
// Añadimos los escuchadores de eventos de los botones
        sigu.addActionListener(this);
        atras.addActionListener(this);
        aceptarRegla.addActionListener(this);
        compilarSRL.addActionListener(this);
        iniciarRegla.addActionListener(this);
        cartas.addChangeListener(this);
}
// ################ Método main principal ####################
public static void main(String[] args)
{
        JFrame creaReglas = new JFrame ("GARNE (Gestión Automática de Reglas en
NeTraMet)");
        final Reglas misReglas = new Reglas();
        creaReglas.addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent e) {
                                misReglas.comprobarGuardado();
                                System.exit(0);
                        }
                }); // fin del creaReglas.addWindowListener
        // Ponemos el panel misReglas
        creaReglas.getContentPane().add(misReglas);
        creaReglas.pack();
        //Para que el JFrame sea visible
        creaReglas.setVisible(true);
```

```
        creaReglas.setResizable(false);
} //fin del main


// ############## Métodos de las implementaciones ##########################

//En nuestro caso es el método ActionPerformed de la interfaz ActionListener

public void actionPerformed(ActionEvent e)
{
//Estas condiciones son para no salirnos de las cartas y seguir un orden
//en la generación de las reglas
Object source = e.getSource();
if (source == sigu)
{
// Para que surja efecto el hacer click en el botón "siguiente"
//se tiene que haber aceptado con el botón aceptar de cada panel en las cartas
//el cual modificara una variable privada de cada clase y si estas es true
//podremos pasar a la siguiente carta
    if ((cartas.getSelectedIndex() == 0) && (define.getPasado()))
    {
        cartas.setEnabledAt(cartas.getSelectedIndex() + 1,true);
        cartas.setSelectedIndex(cartas.getSelectedIndex() + 1);
        atras.setEnabled(true);
        sigu.setEnabled(false);
        iniciarRegla.setEnabled(true);
    }
    if ((cartas.getSelectedIndex() == 1) && (cond.getPasado()))
    {
        cartas.setEnabledAt(cartas.getSelectedIndex() + 1,true);
        cartas.setSelectedIndex(cartas.getSelectedIndex() + 1);
        sigu.setEnabled(false);
    }
    if ((cartas.getSelectedIndex() == 2) && (forma.getPasado()))
    {
        aceptarRegla.setEnabled(true);
        sigu.setEnabled(false);
    }
} // fin del if (source == sigu)

if (source == atras)
{
// Mediante este tratamiento en el evento del botón atras lo que queremos
// es movernos por las cartas de la ultima a la primera poniendo a false
// la carta siguiente si nos encontramos en la ultima y a false atras si
// nos encontramos en la primera
```

```
            cartas.setSelectedIndex(cartas.getSelectedIndex() - 1);
            if (cartas.getSelectedIndex() == 0)
            {
                atras.setEnabled(false);
            }
            if (cartas.getSelectedIndex() != 2)
            {
                sigu.setEnabled(true);
            }
        } // fin del if (source == atras)
        if (source == aceptarRegla)
        // Solamente llamamos al método guardarFichero
        {
                guardarFichero();
        } // fin del if (source == aceptarRegla)
        if (source == iniciarRegla)
        {
                File antiguoFichero = new File(ruta);
                inicializarTodo();
                iniciarRegla.setEnabled(false);
                antiguoFichero.delete();
                define.setPasado(false);
                cond.setPasado(false);
                forma.setPasado(false);
        } // fin del if(source == iniciarRegla)
        if (source == compilarSRL)
        {
            Runtime     compilando = Runtime.getRuntime();
            File comprobacion = new File("srl");
            String aux = abrirFichero();
            String comando = "./srl " + aux + " . ";
            if (aux != null)
            {
                if (comprobacion.exists())
                {
                        try{
                        compilando.exec(comando);
                        }catch(IOException ioe){
                           JOptionPane.showMessageDialog(new JFrame(),"No se ha podido
compilar el fichero: " +  aux + "\n Revise los permisos que dispone en el directorio.");
                        }
                }else
                {
                   JOptionPane.showMessageDialog(new JFrame(),"No se ha encontrado el
fichero: " + comprobacion.getAbsolutePath() + "\nÉste es un fichero de UNIX del paquete
NeTraMet43." );
                }
                try{
```

```
                compilando.exec(comando);
            }catch(IOException ioe)
            {
            }
        } //fin del if (aux != null)
    } // fin del if compilar SRL
} //fin del action performed

public void stateChanged(ChangeEvent e)
{
    if (cartas.getSelectedIndex() == 0)
    {
            atras.setEnabled(false);
            if (cartas.isEnabledAt(cartas.getSelectedIndex() + 1) == true)
            {sigu.setEnabled(true);}
    }
    if (cartas.getSelectedIndex() == 1)
    {
            atras.setEnabled(true);
            if (cartas.isEnabledAt(2) == true) sigu.setEnabled(true);
            else sigu.setEnabled(false);
    }
    if (cartas.getSelectedIndex() == 2)
    {
            sigu.setEnabled(false);
            if (cartas.isEnabledAt(0) == true)
            {
                    atras.setEnabled(true);
            }
    }
} //Fin del stateChanged

//#################### Métodos Auxiliares ##########################
// Este método lo utilizamos para que los objetos condicionales, defines y formato
// puedan habilitar la siguiente carta una vez que se ha dado al botón aceptar
    public void ponerTrueSigu()
    {
            if (cartas.getSelectedIndex() < cartas.getTabCount() - 1)
            {
                    cartas.setEnabledAt(cartas.getSelectedIndex() + 1,true);
                    sigu.setEnabled(true);
            }
            if (cartas.getSelectedIndex() == cartas.getTabCount() - 1)
            {
                    // Si la carta es la ultima modificamos solamente los botones
                    // aceptar regla y siguiente.
```

```
                        aceptarRegla.setEnabled(true);
                        sigu.setEnabled(false);
                        iniciarRegla.setEnabled(true);
                }
        } //fin del método ponerTrueSigu()
// Este método se utiliza para guardar la regla en el fichero que el usuario desee
//y poniendo el nombre al fichero
        public void guardarFichero()
        {
        // La clase JFileChosser nos abre una ventana con la estructura de directorios
        // para guardar el fichero en el directorio que el usuario desee
                JFileChooser guardar = new JFileChooser();
                File antiguoFichero = new File(ruta);// Abrimos el fichero temporal
                //guardar.addChoosableFileFilter(new FiltroSRL());
                // JFileChooser tiene la opción para guardar o para abrir ficheros,
                // nosotros elegimos la de salvar
                int returnVal = guardar.showSaveDialog(this);
               if (returnVal == JFileChooser.APPROVE_OPTION) {
                // Creamos un nuevo fichero con el nombre y ruta que desea el usuario
                        String temporal = guardar.getSelectedFile().getAbsolutePath();
                        if (!temporal.substring(temporal.length() - 3,temporal.length())
.equalsIgnoreCase("srl"))
                        {
                                temporal = temporal + ".srl";
                        }
                        File ficheroNuevo = new File(temporal);
                        if (antiguoFichero.renameTo(ficheroNuevo) == false)
                        {
                                JOptionPane.showMessageDialog(new JFrame(),"Error en la
creación del fichero.\n" + "Probablemente el fichero ya exista o no tiene permisos de
escritura.\n");
                        }
                        compilarSRL.setEnabled(true);
                // Ponemos activo este botón solo cuando se haya creado el fichero srl.
                        aceptarRegla.setEnabled(false);
                // Ponemos a false aceptar la regla porque ya no hay fichero temporal.
                        inicializarTodo(); // Inicializamos el programa

               } else if (returnVal == JFileChooser.CANCEL_OPTION)
                System.out.println("Operación cancelada: " + guardar.getSelectedFile());
        } //if (returnVal == JFileChooser.APPROVE_OPTION)

        // Este método lo utilizamos para comprobar que el fichero ha sido guardado
        // antes de cerrar la aplicación.
        public void comprobarGuardado()
        {
                File fichero = new File(ruta);
                int respuesta;
```

```
                // Almacenamos la respuesta de la ventana de información JOptionPane
                if (fichero.isFile())
                {
                        respuesta = optionPane.showConfirmDialog(new JFrame(),"La regla no
se ha guardado.\n" + "¿Desea guardarla
ahora?\n","Atención...",JOptionPane.ERROR_MESSAGE);
                        if (respuesta == 0) guardarFichero();
                        else fichero.delete();
                }
        }
        //Mediante este método, la clase Define nos pasa los atributos
        public void setArray(String [] defineListas)
        {
                this.defineListas = defineListas;
        }
        //Mediante este método, le pasamos a la clase Condiciones los atributos
        public String[] getArray()
        {
                return this.defineListas;
        }
      // Este método se utiliza para reiniciar la aplicación, de tal modo que en caso de
error, se pueda empezar de nuevo paso a paso sin necesidad de reiniciar la aplicación.
        public void inicializarTodo()
        {
                cartas.setEnabledAt(1,false);
                cartas.setEnabledAt(2,false);
                cartas.setSelectedIndex(0);
                atras.setEnabled(false);
                sigu.setEnabled(false);
                aceptarRegla.setEnabled(false);
                define.iniciar(); // llamamos al iniciar de la clase Define
                cond.quitarAtributos();

        } //fin del método inicializarTodo
        public String abrirFichero()
        {
                JFileChooser guardar = new JFileChooser();
                int returnVal = guardar.showOpenDialog(this);
                if (returnVal == JFileChooser.APPROVE_OPTION)
                {
                        return guardar.getSelectedFile().getAbsolutePath();
                }else
                {
                        return null;
                }
        }
      } //Fin de la clase
```

Defines.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;

public class Defines extends JPanel implements ActionListener
{
 private JTextField    identificadorPool1;
 private JTextField    poolDirecciones1;
 private JLabel define1;
 private final int total = 20; //constante del array histórico
 private JLabel [] histórico = new JLabel[total];
 private JTextArea areaDef;
 private JPanel panelDefine, panelText, parteSur, todo, etiqHis;
 private boolean pasado = false;
 private JButton aceptar, nuevoDefine;
 private JCheckBox siTrafIP,siEscribir;
 private final String ruta = "temp.srl";
 private final int tamanoLetra = 20;
 private String [] todosDefines = new String[total];
 private Reglas miRegla;
 private int cont = 0;
 private int respuesta;

 //############ Constructores ####################
 public Defines(Reglas miRegla)
 {
        this.miRegla = miRegla;
        //Inicialización de los JTextField, JLabel
        define1 = new JLabel("Define");
          define1.setFont(new Font("Define",1,tamanoLetra));
          define1.setToolTipText("Se utiliza para generar etiquetas que
simbolizan rangos de cualquier atributo.");
        identificadorPool1 = new JTextField(10);
          identificadorPool1.setFont(new Font("Nuevo",1,tamanoLetra));
          identificadorPool1.setToolTipText("Ejemplo de identificador
\"Primario1\".");
        poolDirecciones1 = new JTextField(20);
          poolDirecciones1.setFont(new Font("Nuevo",1,tamanoLetra));
          poolDirecciones1.setToolTipText("Ejemplo de pool de direcciones
\"192.168.1.0/24 siendo /24 la mascara de subred\".");
        nuevoDefine = new JButton("Nuevo");
          nuevoDefine.setFont(new Font("Nuevo",1,tamanoLetra));
          nuevoDefine.setToolTipText("Se utiliza para crear un nuevo DEFINE.");
```

```
                siTrafIP = new JCheckBox("Solo trafico IP", true);
                  siTrafIP.setFont(new Font("siTrafIP",1,tamanoLetra));
                  siTrafIP.setToolTipText("Dejar seleccionado si solo se desea capturar
trafico IP.");
                siEscribir = new JCheckBox("Actualizar desde el Área",false);
                  siEscribir.setFont(new Font("siEscribir",1,tamanoLetra));
                  siEscribir.setToolTipText("Si lo selecciona podrá escribir en el
área.");
                panelDefine = new JPanel();
                todo = new JPanel();
                panelText = new JPanel();
                areaDef = new JTextArea(20,20);
                  areaDef.setEnabled(false);
                  areaDef.setFont(new Font("Área",1,tamanoLetra));
                  areaDef.setToolTipText("Puede modificar los atributos directamente.
NOTA: Si no se hace de forma correcta no compilara la regla.");
                panelText.setLayout(new BorderLayout());
                  panelText.add(identificadorPool1,BorderLayout.WEST);
                  panelText.add(poolDirecciones1,BorderLayout.CENTER);

                aceptar = new JButton("Aceptar");
                  aceptar.setFont(new Font("Aceptar",2,tamanoLetra));
                  aceptar.setToolTipText("Para pasar a la siguiente pestaña. Si quiere
utilizar variables define no le de hasta q no las tenga todas creadas.");

                parteSur = new JPanel();
                parteSur.setLayout(new GridLayout(1,3));
                  parteSur.add(siTrafIP);
                  parteSur.add(siEscribir);
                  parteSur.add(aceptar);

                //AÑADIMOS AL PANEL TOTAL

                todo.setLayout(new BorderLayout());
                  todo.add(define1,BorderLayout.NORTH);
                  todo.add(nuevoDefine,BorderLayout.WEST);
                  todo.add(panelText,BorderLayout.CENTER);
                  todo.add(parteSur,BorderLayout.SOUTH);

                setLayout(new BorderLayout());
                  add(todo,BorderLayout.NORTH);
                  add(areaDef,BorderLayout.SOUTH);
                //Añadimos el escuchador de eventos
                aceptar.addActionListener(this);
                nuevoDefine.addActionListener(this);
                siEscribir.addActionListener(this);

        } //Fin del constructor
```

```
        //################# Métodos auxiliares ###################
        public boolean getPasado()
       // Utilizamos este método para ver si los defines ya han sido aceptados, lo que
       //quiere decir que ya están escritos en el fichero de reglas.
        {
                return this.pasado;
        }
        public void setPasado(boolean pasado)
       // Esta variable sirve para cambiar la variable de instancia que se refiere a si
       //hemos aceptado los defines.
        {
                this.pasado = pasado;
        }
        public void formaTexto()
       // Este método añade al fichero temporal las constantes definidas por la
       //aplicación.
        {
                GestionArchivos miGestor = new GestionArchivos(ruta);
                miGestor.escribir(areaDef.getText());
                if (siTrafIP.isSelected() == true)
                {
                   miGestor.escribir(" ");
                   miGestor.escribir(" ");
                   miGestor.escribir("IF (SourcePeerType == IP) || (DestPeerType == IP)
SAVE;");
                   miGestor.escribir("ELSE IGNORE;");
                   miGestor.escribir(" ");
                }
                nuevoDefine.setEnabled(false);
                identificadorPool1.setEditable(false);
                poolDirecciones1.setEditable(false);
                areaDef.setEditable(false);
        }
        //Este método lo utilizamos para pasarle a condicionales los defines
        public String[] getDefines()
        {
                return todosDefines;
        }
        public void iniciar()
       // Método que se llama cuando queremos reiniciar para crear otra regla
        {
                nuevoDefine.setEnabled(true);
                identificadorPool1.setEditable(true);
                identificadorPool1.setText(null);
                poolDirecciones1.setEditable(true);
                poolDirecciones1.setText(null);
                areaDef.setEnabled(false);
```

```
                areaDef.setText(null);
                siEscribir.setSelected(false);
                this.pasado = false;
        }
        //################ Métodos de las interfaces ######################
        public void actionPerformed(ActionEvent e)
        {
                Object source = e.getSource();
                if ((source == aceptar) && (!this.pasado))
      // Mediante el tratamiento de este evento añadimos todas las constantes declaradas
      //en el área de texto las añadimos al fichero temporal.
                {
                   respuesta = JOptionPane.showConfirmDialog(new JFrame(),"¿Esta seguro
que desea aceptar los DEFINE?\n" + "Una vez aceptado, no podrá actualizarlos.",
"Atención...",JOptionPane.ERROR_MESSAGE);
                   if (respuesta == 0)
                   {
                        if (!this.pasado) formaTexto();
                        //Miramos si no la hemos añadido antes
                        this.pasado = true;
                        miRegla.ponerTrueSigu();
                        miRegla.setArray(todosDefines);
                   }
                }
                if (source == nuevoDefine)
      // Este botón añade cada constante declarada en el ares de texto. Y las presenta
      //tal y como estarán en el fichero de reglas
                {
                   if ((0!=identificadorPool1.getText().compareTo(""))&&(0!=
poolDirecciones1.getText().compareTo("")))
                //Comprobamos que no haya ningún campo vacío
                   {
                        areaDef.setText(areaDef.getText() + "DEFINE " +
identificadorPool1.getText() + " = " + poolDirecciones1.getText() + ";\n");
                        if (cont < 20) todosDefines[cont++] =
identificadorPool1.getText();
                        else System.out.println("No se admiten mas constantes.");
                   }else
                   {
                        JOptionPane.showMessageDialog(new Frame(),"Debe poner antes un
atributo o la dirección y la mascara...","Error",JOptionPane.ERROR_MESSAGE);
                   }
                }
                if (source == siEscribir)
                //Habilita el área de texto para escribir en ella
                {
                        areaDef.setEnabled(siEscribir.isSelected());
                }
        } //Fin del ActionPerformed
      }
```

Condiciones.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;

public class Condiciones extends JPanel implements ActionListener, MouseListener
{
  private JCheckBox saveSourcePeerAddress, saveDestPeerAddress, saveSourcePeerType,
saveDestPeerType;
  private JCheckBox saveSourceInterface, saveDestInterface, ignore;
  private JCheckBox saveSourceTransAddress, saveDestTransAddress;
  private JCheckBox saveSourceTransType, saveDestTransType;
  private JComboBox atrib1,atrib2,operLogica;
  private JComboBox relAtrib1, relAtrib2;
  private JCheckBox ifConsecutivo;
  private JComboBox atrib3,atrib4,operLogica2;
  private JComboBox relAtrib3, relAtrib4;
  private JTextField relSaveSource, relSaveDest;
  private JLabel sentIf, sentIf2, operLog; //Etiquetas
  private JPanel panelSDPeerAddress, panelRestoSave, panelSentCond;
  private JPanel panelBotones, panelOrigDest, panelSaves;
  private JPanel panelSentCond2;
  private JButton aceptar,verCondiciones;
  private boolean pasado = false;
  private boolean relAtrib = false;
  private String texto,texto2;
  private String [] defineListas = {""};
  private final String ruta = "temp.srl";
  private final int tamanoLetra = 13;
  private Reglas miRegla;

// ################# Constructor ###################
  public Condiciones(Reglas miRegla)
  {
        this.miRegla = miRegla;
        saveSourcePeerAddress = new JCheckBox("SAVE SourcePeerAddress/");
           saveSourcePeerAddress.setFont(new Font("saveSourcePeerAddress",4
,tamanoLetra));
           saveSourcePeerAddress.setToolTipText("Salvamos las Direcciones IP
Origen");
        saveDestPeerAddress = new JCheckBox("SAVE DestPeerAddress/");
           saveDestPeerAddress.setFont(new Font("saveDestPeerAddress",4
,tamanoLetra));
           saveDestPeerAddress.setToolTipText("Salvamos las Direcciones IP
Destino");
```

```
                relSaveSource = new JTextField("32",3);
                   relSaveSource.setFont(new Font("relSaveSource",4,tamanoLetra));
                   relSaveSource.setToolTipText("Podemos modificar la mascara de
subred.");

                relSaveDest = new JTextField("32",3);
                   relSaveDest.setFont(new Font("relSaveDest",4,tamanoLetra));
                   relSaveDest.setToolTipText("Podemos modificar la mascara de subred.");

                saveSourcePeerType = new JCheckBox("SAVE SourcePeerType");
                   saveSourcePeerType.setFont(new Font("saveSourcePeerType",4
,tamanoLetra));
                   saveSourcePeerType.setToolTipText("Salvamos el tipo de dirección
Origen. Ejemplo IP.");

                saveDestPeerType = new JCheckBox("SAVE DestPeerType");
                   saveDestPeerType.setFont(new Font("saveDestPeerType",4,tamanoLetra));
                   saveDestPeerType.setToolTipText("Salvamos el tipo de dirección
Destino. Ejemplo IP.");

                saveSourceInterface = new JCheckBox("SAVE SourceInterface");
                   saveSourceInterface.setFont(new Font("saveSourceInterface"
,4,tamanoLetra));
                   saveSourceInterface.setToolTipText("Salvamos la interfaz de Origen.");

                saveDestInterface = new JCheckBox("SAVE DestInterface");
                   saveDestInterface.setFont(new Font("saveDestInterface"
,4,tamanoLetra));
                   saveDestInterface.setToolTipText("Salvamos la interfaz de Origen.");

                saveSourceTransAddress = new JCheckBox("SAVE SourceTransAddress");
                   saveSourceTransAddress.setFont(new Font("saveSourceTransAddress"
,4,tamanoLetra));
                   saveSourceTransAddress.setToolTipText("Salvamos la dirección del
puerto de origen (Telnet, FTP, ...).");

                saveDestTransAddress = new JCheckBox("SAVE DestTransAddress");
                   saveDestTransAddress.setFont(new Font("saveDestTransAddress"
,4,tamanoLetra));
                   saveDestTransAddress.setToolTipText("Salvamos la dirección del puerto
de destino (Telnet, FTP, ...).");

                saveSourceTransType = new JCheckBox("SAVE SourceTransType");
                   saveSourceTransType.setFont(new Font("saveSourceTransType"
,4,tamanoLetra));
                   saveSourceTransType.setToolTipText("Salvamos el tipo de Protocolo
Origen UDP o TCP .");

                saveDestTransType = new JCheckBox("SAVE DestTransType");
                   saveDestTransType.setFont(new Font("saveDestTransType"
,4,tamanoLetra));
```

```
                saveDestTransType.setToolTipText("Salvamos el tipo de Protocolo
Destino UDP o TCP.");


            ignore = new JCheckBox("IGNORE");
               ignore.setFont(new Font("ignore",4,tamanoLetra));
               ignore.setToolTipText("Ignoramos todos los atributos.");


            panelSDPeerAddress = new JPanel();
            panelSDPeerAddress.setLayout(new GridLayout (2,2));
               panelSDPeerAddress.add(saveSourcePeerAddress);
               panelSDPeerAddress.add(relSaveSource);
               panelSDPeerAddress.add(saveDestPeerAddress);
               panelSDPeerAddress.add(relSaveDest);


            //Añadimos la sentencia condicional
            sentIf = new JLabel("IF");
            sentIf2 = new JLabel("     IF ANIDADO");
            String [] operaciones = {"Solo un atributo","OR","AND"};
            String [] operaciones2 = {"Solo un atributo","OR","AND"};
            operLogica = new JComboBox(operaciones);
               operLogica.setFont(new Font("operLogica",4,tamanoLetra));
               operLogica.setToolTipText("Si necesitamos dos condiciones, elegimos la
operación lógica que las rigen.");
            operLogica2 = new JComboBox(operaciones2);
               operLogica2.setFont(new Font("operLogica",4,tamanoLetra));
               operLogica2.setToolTipText("Si necesitamos dos condiciones, elegimos
la operación lógica que las rigen.");
            String [] elegir1 = {"SourcePeerAddress","DestPeerAddress"
,"SourcePeerType","DestPeerType","SourceInterface","DestInterface","SourceTransType",
"DestTransType","SourceTransAddress","DestTransAddress"};
            String [] elegir2 = {"Ninguno","SourcePeerAddress","DestPeerAddress"
,"SourcePeerType","DestPeerType","SourceInterface","DestInterface","SourceTransType"
,"DestTransType","SourceTransAddress","DestTransAddress"};
            String [] elegir3 = {"SourcePeerAddress","DestPeerAddress"
,"SourcePeerType","DestPeerType","SourceInterface","DestInterface","SourceTransType"
,"DestTransType","SourceTransAddress","DestTransAddress"};
            String [] elegir4 = {"Ninguno","SourcePeerAddress","DestPeerAddress"
,"SourcePeerType","DestPeerType","SourceInterface","DestInterface","SourceTransType"
,"DestTransType","SourceTransAddress","DestTransAddress"};


            atrib1 = new JComboBox (elegir1);
                   atrib1.setFont(new Font("atrib1",4,tamanoLetra));
                   atrib1.setToolTipText("Atributos básicos.");
            atrib2 = new JComboBox (elegir2);
                   atrib2.setFont(new Font("atrib2",4,tamanoLetra));
                   atrib2.setToolTipText("Atributos básicos.");
                   atrib2.setEnabled(false);


            relAtrib1 = new JComboBox(defineListas);//si no va quita la variable
                   relAtrib1.setEditable(true);
                   relAtrib1.setFont(new Font("relAtrib1",4,tamanoLetra));
```

```
            relAtrib2 = new JComboBox(defineListas);
               relAtrib2.setEditable(false);
               relAtrib2.setFont(new Font("relAtrib2",4,tamanoLetra));
               relAtrib2.setEnabled(false);
            atrib3 = new JComboBox (elegir3);
               atrib3.setFont(new Font("atrib3",4,tamanoLetra));
               atrib3.setToolTipText("Atributos básicos.");
            atrib4 = new JComboBox (elegir4);
               atrib4.setFont(new Font("atrib4",4,tamanoLetra));
               atrib4.setToolTipText("Atributos básicos.");
               atrib4.setEnabled(false);
            relAtrib3 = new JComboBox(defineListas);//si no va quita la variable
               relAtrib3.setEditable(true);
               relAtrib3.setFont(new Font("relAtrib3",4,tamanoLetra));
            relAtrib4 = new JComboBox(defineListas);
               relAtrib4.setEditable(false);
               relAtrib4.setFont(new Font("relAtrib2",4,tamanoLetra));
               relAtrib4.setEnabled(false);
         // Creamos los paneles con la línea de condiciones
            panelSentCond = new JPanel();
            panelSentCond.setLayout(new GridLayout (2,3));
               panelSentCond.add(sentIf);
               panelSentCond.add(atrib1);
               panelSentCond.add(relAtrib1);
               panelSentCond.add(operLogica);
               panelSentCond.add(atrib2);
               panelSentCond.add(relAtrib2);
            panelSentCond2 = new JPanel();
            panelSentCond2.setLayout(new GridLayout (2,3));
               panelSentCond2.add(sentIf2);
               panelSentCond2.add(atrib3);
               panelSentCond2.add(relAtrib3);
               panelSentCond2.add(operLogica2);
               panelSentCond2.add(atrib4);
               panelSentCond2.add(relAtrib4);
               panelSentCond2.setVisible(false);
         //Añadimos a un panel todos los SAVE de los atributos y la sentencia Condicional.
            panelRestoSave = new JPanel();
            panelRestoSave.setLayout(new GridLayout (12,1));
               panelRestoSave.add(panelSentCond);
               panelRestoSave.add(panelSentCond2);
               panelRestoSave.add(panelSDPeerAddress);
               panelRestoSave.add(saveSourcePeerType);
               panelRestoSave.add(saveDestPeerType);
               panelRestoSave.add(saveSourceTransAddress);
               panelRestoSave.add(saveDestTransAddress);
```

```
                panelRestoSave.add(saveSourceTransType);
                panelRestoSave.add(saveDestTransType);
                panelRestoSave.add(saveSourceInterface);
                panelRestoSave.add(saveDestInterface);
                panelRestoSave.add(ignore);

            aceptar = new JButton("Aceptar");
                aceptar.setFont(new Font("aceptar",4,20));
                aceptar.setToolTipText("Podemos Aceptar tantas condiciones como
deseemos para formar la regla.");
            verCondiciones = new JButton("Ver Condiciones");
                verCondiciones.setFont(new Font("ver",4,20));
                verCondiciones.setToolTipText("Podemos ver las condiciones que
llevamos realizadas.");
            ifConsecutivo = new JCheckBox("Anidar otra condición");
                ifConsecutivo.setFont(new Font("Anidar",4,20));
                ifConsecutivo.setToolTipText("Lo utilizamos para anidar una condición
dentro de otra.");
        //En este panel añadiremos los botones de aceptar, ver regla y anidar if
            panelBotones = new JPanel();
            panelBotones.setLayout(new GridLayout (1,2));
                panelBotones.add(aceptar);
                panelBotones.add(ifConsecutivo);
        //Añadimos todo al panel principal
            setLayout(new BorderLayout());

            add(panelRestoSave,BorderLayout.CENTER);
            add(panelBotones,BorderLayout.SOUTH);
        //Añadimos el escuchador de eventos
            aceptar.addActionListener(this);
            verCondiciones.addActionListener(this);
            ifConsecutivo.addActionListener(this);
            ignore.addActionListener(this);
            operLogica.addActionListener(this);
            operLogica2.addActionListener(this);
            atrib4.addActionListener(this);
            atrib3.addActionListener(this);
            atrib2.addActionListener(this);
            atrib1.addActionListener(this);
            relAtrib1.addMouseListener(this);
            relAtrib2.addMouseListener(this);
            relAtrib3.addMouseListener(this);
            relAtrib4.addMouseListener(this);
            addMouseListener(this);
        }
```

```
      //############### Métodos de las interfaces ####################
      public void actionPerformed(ActionEvent e)
      {
        Object source = e.getSource();
        if (source == aceptar)
        {
           if ((atrib2.getSelectedIndex() == 0) && (operLogica.getSelectedIndex() != 0))
           {
               JOptionPane.showMessageDialog(new Frame(),"Debe colocar un segundo
Atributo.","Error",JOptionPane.ERROR_MESSAGE);
           }else
           {
               this.pasado = true;
               generarCondiciones();
               miRegla.ponerTrueSigu();
           }
        } //fin del if aceptar

       if (source == ignore) //para poner a no editable los otros jCheckBox cuando se
selecciona ignorar
        {
           if (ignore.isSelected())
           {
               ponerA(false);
           }else{
               ponerA(true);
          }
        } //fin del if ignore

        if (source == operLogica)
        {
           if (operLogica.getSelectedIndex() == 0)
           {
               relAtrib2.setEditable(false);
               relAtrib2.setEnabled(false);
               atrib2.setSelectedIndex(0);
               atrib2.setEnabled(false);
           }else{
               relAtrib2.setEditable(true);
               relAtrib2.setEnabled(true);
               atrib2.setEnabled(true);
          }
        } //fin del if operLogica

        if (source == operLogica2)
        {
           if (operLogica2.getSelectedIndex() == 0)
```

```
            {
                relAtrib4.setEditable(false);
                relAtrib4.setEnabled(false);
                atrib4.setSelectedIndex(0);
                atrib4.setEnabled(false);
            }else{
                relAtrib4.setEditable(true);
                relAtrib4.setEnabled(true);
                atrib4.setEnabled(true);
            }
        } //fin del if operLogica2

        if (source == atrib2) //Para no poner el mismo atributo
        {
            if (atrib1.getSelectedItem() == atrib2.getSelectedItem())
            {
                JOptionPane.showMessageDialog(new Frame(),"No se pueden poner dos
atributos iguales...","Error",JOptionPane.ERROR_MESSAGE);
                atrib2.setSelectedIndex(0);
            }
        } //fin if (source == atrib2)

        if (source == atrib4) //Para no poner el mismo atributo
        {
            if (atrib3.getSelectedItem() == atrib4.getSelectedItem())
            {
                JOptionPane.showMessageDialog(new Frame(),"No se pueden poner dos
atributos iguales...","Error",JOptionPane.ERROR_MESSAGE);
                atrib4.setSelectedIndex(0);
            }
        } //fin if (source == atrib4)

        if (source == atrib1) //Para no poner el mismo atributo
        {
            if (atrib1.getSelectedItem() == atrib2.getSelectedItem())
            {
                JOptionPane.showMessageDialog(new Frame(),"No se pueden poner dos
atributos iguales...","Error",JOptionPane.ERROR_MESSAGE);
                atrib1.setSelectedIndex(0);
            }
        } //fin if (source == atrib1)

        if (source == atrib3) //Para no poner el mismo atributo
        {
            if (atrib3.getSelectedItem() == atrib4.getSelectedItem())
            {
                JOptionPane.showMessageDialog(new Frame(),"No se pueden poner dos
atributos iguales...","Error",JOptionPane.ERROR_MESSAGE);
```

```
            atrib3.setSelectedIndex(0);
        }
    } //fin if (source == atrib1)

    if (source == ifConsecutivo) //Para no poner el mismo atributo
    {
       panelSentCond2.setVisible(ifConsecutivo.isSelected());
    } //fin if (source == ifConsecutivo)
}  //Fin del ActionPerformed

// ################ EVENTOS DE RATON ####################
 public void mouseClicked(MouseEvent e)
 {pasarAtributos();}
 public void mouseEntered(MouseEvent e)
 {pasarAtributos();}
 public void mouseExited(MouseEvent e)
 {pasarAtributos();}
 public void mousePressed(MouseEvent e)
 {}
 public void mouseReleased(MouseEvent e)
 {}
 //#######################################################

 //################ Metodos Auxiliares ####################
 //Este método se ejecuta cuando se pulsa el botón aceptar, indica que podemos
 //pasar a la siguiente carta
 public boolean getPasado()
 {return this.pasado;}

 public void setPasado(boolean pasado)
 {this.pasado = pasado;}
 //Metodo para poner todos los save a false o a true
 public void ponerA(boolean seleccionado)
 {
        saveSourcePeerAddress.setEnabled(seleccionado);
        saveDestPeerAddress.setEnabled(seleccionado);
        saveSourcePeerType.setEnabled(seleccionado);
        saveDestPeerType.setEnabled(seleccionado);
        saveSourceInterface.setEnabled(seleccionado);
        saveDestInterface.setEnabled(seleccionado);
        saveSourceTransAddress.setEnabled(seleccionado);
        saveDestTransAddress.setEnabled(seleccionado);
        saveSourceTransType.setEnabled(seleccionado);
        saveDestTransType.setEnabled(seleccionado);
 }
```

```
        // Este método genera las condiciones y las escribe en el fichero, este método
        //se llama desde el evento del botón aceptar

        public void generarCondiciones()

        {

           GestionArchivos miGestor = new GestionArchivos(ruta);

           if (ifConsecutivo.isSelected())

           {

                if (operLogica.getSelectedIndex() == 0)

                {

                   texto = "IF (" + atrib1.getSelectedItem() + " == " +
relAtrib1.getSelectedItem() + ")";

                }else{

                   texto = "IF ((" + atrib1.getSelectedItem() + " == " +
relAtrib1.getSelectedItem() + ") " + ((operLogica.getSelectedItem() == "OR") ? "||" :
"&&") + " (" + atrib2.getSelectedItem() + " == " + relAtrib2.getSelectedItem() + "))";

                }

                if (operLogica2.getSelectedIndex() == 0)

                {

                   texto2 = "IF (" + atrib3.getSelectedItem() + " == " +
relAtrib3.getSelectedItem() + ")";

                }else{

                   texto2 = "IF ((" + atrib3.getSelectedItem() + " == " +
relAtrib3.getSelectedItem() + ") " + ((operLogica.getSelectedItem() == "OR") ? "||" :
"&&") + " (" + atrib4.getSelectedItem() + " == " + relAtrib4.getSelectedItem() + "))";

                }

           }else{

                if (operLogica.getSelectedIndex() == 0)

                {

                   texto = "IF (" + atrib1.getSelectedItem() + " == " +
relAtrib1.getSelectedItem() + ")";

                }else{

                   texto = "IF ((" + atrib1.getSelectedItem() + " == " +
relAtrib1.getSelectedItem() + ") " + ((operLogica.getSelectedItem() == "OR") ? "||" :
"&&") + " (" + atrib2.getSelectedItem() + " == " + relAtrib2.getSelectedItem() + "))";

                }

           }

           miGestor.escribir(texto);

           if (ifConsecutivo.isSelected()) miGestor.escribir("\t" + texto2);

           miGestor.escribir("\t{");

           if (ignore.isSelected() == true) {

                miGestor.escribir(" \t\tIGNORE; ");

           }else if (ignore.isSelected() == false)

                {

                      if (saveSourcePeerAddress.isSelected() == true)
{miGestor.escribir(" \t\tSAVE SourcePeerAddress/ "+ relSaveSource.getText()+";");}

                      if (saveDestPeerAddress.isSelected() == true) {miGestor.escribir("
\t\tSAVE DestPeerAddress/ "+ relSaveDest.getText()+";");}

                      if (saveSourcePeerType.isSelected() == true)
{miGestor.escribir("\t\tSAVE SourcePeerType;");}

                      if (saveDestPeerType.isSelected() == true) {miGestor.escribir("
\t\tSAVE DestPeerType; ");}
```

```
                    if (saveSourceTransAddress.isSelected() == true)
{miGestor.escribir(" \t\tSAVE SourceTransAddress/16; ");}
                    if (saveDestTransAddress.isSelected() == true)
{miGestor.escribir(" \t\tSAVE DestTransAddress/16; ");}
                    if (saveSourceTransType.isSelected() == true) {miGestor.escribir("
\t\tSAVE SourceTransType; ");}
                    if (saveDestTransType.isSelected() == true) {miGestor.escribir("
\t\tSAVE DestTransType; ");}
                    if (saveSourceInterface.isSelected() == true) {miGestor.escribir("
\t\tSAVE SourceInterface; ");}
                    if (saveDestInterface.isSelected() == true) {miGestor.escribir("
\t\tSAVE DestInterface; ");}
                    miGestor.escribir(" \t\tCOUNT; ");
               }
           miGestor.escribir(" \t} ");
           miGestor.escribir(" ");
       } //fin del generarCondiciones()

       // Se utiliza para pasar los defines creados en el primer paso al JComboBox.
       public void pasarAtributos()
       {
          if (!relAtrib)
          {
               defineListas = miRegla.getArray();
               relAtrib = true;
               for (int i = 0; i<defineListas.length; i++ )
               {
                  if (defineListas[i] != null)
                  {
                       relAtrib1.insertItemAt(defineListas[i],i);
                       relAtrib2.insertItemAt(defineListas[i],i);
                       relAtrib3.insertItemAt(defineListas[i],i);
                       relAtrib4.insertItemAt(defineListas[i],i);
                  }
               }
          }//fin if (!relAtrib))
       } // fin del método pasarAtributos()

       public void quitarAtributos()
       {
          relAtrib1.removeAllItems();
          relAtrib2.removeAllItems();
          relAtrib3.removeAllItems();
          relAtrib4.removeAllItems();
          for (int i = 0; i<defineListas.length; i++ )
          {
               defineListas[i] = null;
          }
```

```
            relAtrib = false;
        } // fin del método quitarAtributos()
    } //fin de la clase
```

Formato.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;

public class Formato extends JPanel implements ActionListener
{
  // Vamos a crear el panel para introducir el formato que queremos en el fichero
  //de reglas


    private JCheckBox firstTime, flowIndex, flowRuleSet, sourcePeerType,
destPeerType;
    private JCheckBox sourcePeerAddress, destPeerAddress, sourceInterface,
destInterface;
    private JCheckBox sourceTransAddress, destTransAddress, sourceTransType,
destTransType;
    private JCheckBox fromOctets, toOctets;
    private JCheckBox statistic;
    private JLabel formato;
    private JButton aceptar;
    private JPanel panelFormato,panelEtiqueta,panelAcepEsta;
    private boolean pasado = false;
    private final String ruta = "temp.srl";
    private Reglas miRegla;
    private final int tamanoLetraAtrib = 18;
    private final int tamanoLetraResto = 20;


//############## Constructor #################
    public Formato(Reglas miRegla)
    {
        this.miRegla = miRegla;
        // Inicializamos todos los JCheckBox
        formato = new JLabel("Formato para el fichero de Flujo que se creará en
el filtrado.");
           formato.setFont(new Font("formato",4,tamanoLetraResto));
        firstTime = new JCheckBox ("FirstTime",true);
           firstTime.setFont(new Font("firstTime",4,tamanoLetraAtrib));
           firstTime.setToolTipText("Momento inicial del Flujo.");
        flowIndex = new JCheckBox ("FlowIndex",true);
           flowIndex.setFont(new Font("flowIndex",4,tamanoLetraAtrib));
           flowIndex.setToolTipText("Índice del Flujo.");
        flowRuleSet = new JCheckBox ("FlowRuleSet",true);
           flowRuleSet.setFont(new Font("flowRuleSet",4,tamanoLetraAtrib));
           flowRuleSet.setToolTipText("Regla utilizada por el medidor");
        sourcePeerType = new JCheckBox ("SourcePeerType",true);
```

```
                sourcePeerType.setFont(new Font("sourcePeerType" ,4,tamanoLetraAtrib));
                sourcePeerType.setToolTipText("Tipo de Dirección Origen.");
            destPeerType = new JCheckBox ("DestPeerType",true);
                destPeerType.setFont(new Font("destPeerType",4,tamanoLetraAtrib));
                destPeerType.setToolTipText("Tipo de Dirección Destino.");
            sourcePeerAddress = new JCheckBox ("SourcePeerAddress",true);
                sourcePeerAddress.setFont(new Font("sourcePeerAddress",4,
tamanoLetraAtrib));
                sourcePeerAddress.setToolTipText("Dirección Origen");
            destPeerAddress = new JCheckBox ("DestPeerAddress",true);
                destPeerAddress.setFont(new Font("destPeerAddress",4,
tamanoLetraAtrib));
                destPeerAddress.setToolTipText("Dirección Destino");
            sourceInterface = new JCheckBox ("SourceInterface",true);
                sourceInterface.setFont(new Font("sourceInterface",4
,tamanoLetraAtrib));
                sourceInterface.setToolTipText("Interfaz SNMP Origen");
            destInterface = new JCheckBox ("DestInterface",true);
                destInterface.setFont(new Font("destInterface",4 ,tamanoLetraAtrib));
                destInterface.setToolTipText("Interfaz SNMP Destino");
            sourceTransAddress = new JCheckBox ("SourceTransAddress",true);
                sourceTransAddress.setFont(new Font("sourceTransAddress",4 ,
tamanoLetraAtrib));
                sourceTransAddress.setToolTipText("Puerto (de la capa de transporte)
Origen");
            destTransAddress = new JCheckBox ("DestTransAddress",true);
                destTransAddress.setFont(new Font("destTransAddress",4
,tamanoLetraAtrib));
                destTransAddress.setToolTipText("Puerto (de la capa de transporte)
Destino");
            sourceTransType = new JCheckBox ("SourceTransType",true);
                sourceTransType.setFont(new Font("sourceTransType",4
,tamanoLetraAtrib));
                sourceTransType.setToolTipText("Tipo de Protocolo de Transporte
Origen.");
            destTransType = new JCheckBox ("DestTransType",true);
                destTransType.setFont(new Font("destTransType",4 ,tamanoLetraAtrib));
                destTransType.setToolTipText("Tipo de Protocolo de Transporte
Destino.");
            fromOctets = new JCheckBox ("FromOctets",true);
                fromOctets.setFont(new Font("fromOctets",4,tamanoLetraAtrib));
                fromOctets.setToolTipText("Tamaño en Bytes de los Paquetes de Destino
a Origen");
            toOctets = new JCheckBox ("ToOctets",true);
                toOctets.setFont(new Font("toOctets",4,tamanoLetraAtrib));
                toOctets.setToolTipText("Tamaño en Bytes de los Paquetes de Origen a
Destino");
            statistic = new JCheckBox("Estadísticas",false);
                statistic.setFont(new Font("statistic",4,tamanoLetraResto));
                statistic.setToolTipText("Se utiliza para obtener estadísticas del
flujo exportado.");
```

```
                // Inicializamos los paneles
                panelFormato = new JPanel();
                panelFormato.setLayout(new GridLayout (4,4));
                   panelFormato.add(firstTime);
                   panelFormato.add(flowIndex);
                   panelFormato.add(flowRuleSet);
                   panelFormato.add(sourcePeerType);
                   panelFormato.add(destPeerType);
                   panelFormato.add(sourcePeerAddress);
                   panelFormato.add(destPeerAddress);
                   panelFormato.add(sourceInterface);
                   panelFormato.add(destInterface);
                   panelFormato.add(sourceTransAddress);
                   panelFormato.add(destTransAddress);
                   panelFormato.add(sourceTransType);
                   panelFormato.add(destTransType);
                   panelFormato.add(fromOctets);
                   panelFormato.add(toOctets);
                panelAcepEsta = new JPanel();
                panelAcepEsta.setLayout(new GridLayout (1,2));
                   aceptar = new JButton("Aceptar");
                   aceptar.setFont(new Font("aceptar",4,tamanoLetraResto));
                   panelAcepEsta.add(statistic);
                   panelAcepEsta.add(aceptar);
                panelEtiqueta = new JPanel();
                panelEtiqueta.add(formato);

                // Añadimos los dos paneles a panel principal
                setLayout(new BorderLayout());
                   add(panelEtiqueta,BorderLayout.NORTH);
                   add(panelFormato,BorderLayout.CENTER);
                   add(panelAcepEsta,BorderLayout.SOUTH);
                aceptar.addActionListener(this);
             } // fin del constructor

          // Metodos para utilizar las variables de instancia
          public void formaOpciones()
          {
             String texto = new String("FORMAT");
             GestionArchivos miGestor = new GestionArchivos(ruta);
          //FORMAT   FlowIndex FlowRuleSet  FirstTime  SourcePeerAddress  DestPeerAddress
FromOctets ToOctets SourceInterface DestInterface
             if (flowIndex.isSelected() == true) {texto = texto + " " +
flowIndex.getText();  }
             if (flowRuleSet.isSelected() == true) {texto = texto + " " +
flowRuleSet.getText();}
             if (firstTime.isSelected() == true) {texto = texto + " " +
firstTime.getText();  }
```

```
            if (sourcePeerType.isSelected() == true) {texto = texto + " " +
sourcePeerType.getText();     }

            if (destPeerType.isSelected() == true) {texto = texto + " " +
destPeerType.getText();       }

            if (sourcePeerAddress.isSelected() == true) {texto = texto + " " +
sourcePeerAddress.getText();  }

            if (destPeerAddress.isSelected() == true) {texto = texto + " " +
destPeerAddress.getText();    }

            if (sourceInterface.isSelected() == true) {texto = texto + " " +
sourceInterface.getText();    }

            if (destInterface.isSelected() == true) {texto = texto + " " +
destInterface.getText();      }

            if (sourceTransAddress.isSelected() == true) {texto = texto + " " +
sourceTransAddress.getText(); }

            if (destTransAddress.isSelected() == true) {texto = texto + " " +
destTransAddress.getText();   }

            if (sourceTransType.isSelected() == true) {texto = texto + " " +
sourceTransType.getText();    }

            if (destTransType.isSelected() == true) {texto = texto + " " +
destTransType.getText();      }

            if (fromOctets.isSelected() == true) {texto = texto + " " +
fromOctets.getText(); }

            if (toOctets.isSelected() == true) {texto = texto + " " + toOctets.getText();}

           miGestor.escribir(" ");

            miGestor.escribir("SET 2;");

            miGestor.escribir(texto + ";");

            if (statistic.isSelected() == true) {miGestor.escribir("STATISTICS;");}


        }//Fin del método formaOpciones()


        public boolean getPasado()
        {return this.pasado;}


        public void setPasado(boolean pasado)
        {this.pasado = pasado;}


        //Eventos utilizados en la clase
        public void actionPerformed(ActionEvent e)
        {
                Object source = e.getSource();
                if (source == aceptar)
                {
                        if (!this.pasado) formaOpciones();
                        this.pasado = true;
                        miRegla.ponerTrueSigu();
                }
        } //fin del actionPerformed(ActionEvent e)
      } //Fin de la clase
```

Apéndice C:

Código Fuente de la Aplicación DUFNE

Este es el código en C de la aplicación DUFNE, en el que explicaremos sus métodos mediante comentarios.

Servidor: VP5.c

```
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <time.h>
#include "flowdata.h"

/*#####################################*/
/*##### Declaración de variables #####*/
/*#####################################*/
IPStat5Msg flujo; /*paquetes para el flujo*/
struct sockaddr_in servidor, cliente, servidorUDP,clienteUDP[20];

int long_cliente, long_servidor,long_clienteUDP[20],long_servidorUDP;
int sfd_servidor, sfd_cliente,sfd_servidorUDP,sfd_clienteUDP[20];
int nsfd,error;
int num_pet = 0; /*Numero de clientes entre 0 y 19*/
char buffer[1024];
int puertos[20]; // Debe coincidir con el numero máximo de clientes (max_clientes) que el servidor va a aceptar
int ind_puertos = 0;
int puerto = 3002;

/*#####################################*/
/*##### Declaración de constantes #####*/
/*#####################################*/

#define tam_flujo_v_5 (sizeof flujo)
#define max_clientes 20 // Si se modifica hay que cambiar también el valor del array puertos[] que esta mas arriba
```

```
/*####################################*/
/*######## METODOS UTILIZADOS ########*/
/*####################################*/
int hay_mensajes(int sfd);
int hay_mensajes(int sfd)
{
    fd_set fdread;
    struct timeval timeout;
    FD_ZERO (&fdread);
    FD_SET (sfd,&fdread);
    timeout.tv_sec = 0;
    timeout.tv_usec = 0;
    if (select (sfd + 1, &fdread, 0, 0, &timeout) == -1)
    {
 printf("hay mensaje(select)");
    }
    if (FD_ISSET(sfd,&fdread)) return (!0);
    else return (0);
}

/*####################################*/
/*######## PROGRAMA PRINCIPAL ########*/
/*####################################*/
void main (int argc, char *argv[])
{
    if (argc != 2){
 printf ("Hay que introducir como parámetro el puerto por donde CISCO te exportara
el flujo:\n Por ejemplo: v_p5 3001\n");
 exit(1);
    }

      /*Lo utilizo para recibir los paquetes exportado por el router cisco*/
      /*Utilizamos una conexión UDP*/
      sfd_servidorUDP  =  socket(AF_INET,SOCK_DGRAM,0);  /*Canal  para  recibir  los
flujos*/
      servidorUDP.sin_family = AF_INET;
      servidorUDP.sin_port = htons(atoi(argv[1])); /*puerto 3001 momentáneo*/
      servidorUDP.sin_addr.s_addr = htonl(INADDR_ANY);

      long_servidorUDP = sizeof(servidorUDP);

      error                    =                    bind(sfd_servidorUDP,(struct
sockaddr*)&servidorUDP,long_servidorUDP);

      /*Socket por donde va a recibir las peticiones de los clientes*/
      /*Utilizamos una conexión TCP*/
      sfd_servidor    =    socket(AF_INET,SOCK_STREAM,0);    /*CREAMOS    EL    CANAL
BIDIRECCIONAL DEL SOCKET*/
```

```
            servidor.sin_family  =  AF_INET;  /*Indicamos  que  es  de  la  familia  de
internet*/
            servidor.sin_port = htons(2501);
            servidor.sin_addr.s_addr = htonl(INADDR_ANY);


            printf("Servidor a la espera de recibir peticiones de clientes para exportar
flujo.\n");


            long_servidor = sizeof(servidor);


            /*Unimos el socket a la dirección ip determinada, en nuestro caso es la
dirección local.*/
            error = bind(sfd_servidor,&servidor,long_servidor);


            if (error == -1){
         printf("error bind\n");
         exit(0);
            }


            /*Ponemos el servidor a la escucha de usuarios*/
            error = listen (sfd_servidor,5); /*5 peticiones*/
            if (error == -1){
         printf("error listen\n");
         exit(0);
            }
            long_cliente = sizeof(cliente);
      /*Este bucle de repite de forma indefinida para ir aceptando las peticiones de los
clientes, cuando recibe una petición se genera un proceso hijo el cual lee el flujo que
le llega por la conexión UDP y lo exporta al cliente que ha generado la petición.*/
      while(1){
         if (hay_mensajes(sfd_servidor))
         {
             printf("Hay una petición de un cliente...\n");
             nsfd = accept(sfd_servidor,&cliente,&long_cliente);


             if (num_pet < max_clientes)
             {
                 error = sprintf(buffer,"%d",puerto);
                 error = write(nsfd,buffer,1024);
                 clienteUDP[num_pet] = cliente;
                 puertos[num_pet] = puerto;
                 sfd_clienteUDP[num_pet] = socket(AF_INET,SOCK_DGRAM,0);
                 clienteUDP[num_pet].sin_family = AF_INET; //familia de internet
                 clienteUDP[num_pet].sin_port  =  htons(puerto);  //puerto  del  cliente
remoto
                 clienteUDP[num_pet].sin_addr.s_addr      =      cliente.sin_addr.s_addr;
//Dirección del cliente remoto
                 long_clienteUDP[num_pet] = sizeof (clienteUDP[num_pet]);
                 num_pet++; //incrementamos el numero de peticiones realizadas
```

```
                puerto++;
            }else{
                error = sprintf(buffer,"%d",0);
                error = write(nsfd,buffer,1024);
                printf("La petición ha sido denegada por exceso del limite de
      clientes...\n");
            }
        } // fin del if (hay_mensajes(sfd_servidor))

        if (hay_mensajes(sfd_servidorUDP))
        {
            printf ("ha llegado flujo...\n");
            error       =       recvfrom(sfd_servidorUDP,&flujo,tam_flujo_v_5,0,(struct
sockaddr*)&servidorUDP,&long_servidorUDP);
            if (error == -1) {
                printf ("Se ha producido un error al recibir el flujo exportado por
cisco...");
                exit(0);
            }
            for (ind_puertos = 0 ; ind_puertos < num_pet ; ind_puertos++)
            {
                error = sendto(sfd_clienteUDP[ind_puertos],&flujo,tam_flujo_v_5,0,
(struct sockaddr*)&clienteUDP[ind_puertos] ,long_clienteUDP[ind_puertos]);

                if (error == -1) {
                    printf ("Se ha producido un error al enviar los datos al
cliente...");
                    exit(0);
                }

            } //fin del for
        } //Fin del if (hay_mensajes(sfd_servidorUDP))
          } /*fin del while*/
      } //fin del main
```

Cliente: cliente_puertos.c

```
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include "flowdata.h"
/*Declaración de variables*/
struct sockaddr_in cliente;
int long_cliente;
int sfd_cliente;
int nsfd, error;
/* Declaración de constantes*/
#define tam_flujo_v_5 (sizeof flujo)
char buffer[1024];


/*###################################*/
/*####### PROGRAMA PRINCIPAL #######*/
/*###################################*/
main (int argc, char *argv[]){
/*Abrimos un socket TCP para que el cliente se comunique con el servidor
y éste le pueda pasar el puerto por el que le va a exportar el flujo. */
    sfd_cliente = socket(AF_INET,SOCK_STREAM,0);
/*Rellenamos los campos del cliente, la familia de Internet y la dirección ip y el
puerto lo coge de la línea de comandos*/
    cliente.sin_family = AF_INET;
    cliente.sin_port = htons(atoi(argv[2]));
    cliente.sin_addr.s_addr = inet_addr(argv[1]);
    long_cliente = sizeof(cliente);
    /*Conectamos con el servidor para que nos pase el puerto.*/
    error = connect(sfd_cliente,&cliente,long_cliente);
    if (error == -1){
         printf("error en connect");
         exit(0);
    }
/*Leemos el puerto por donde vamos a recibir el flujo exportado por cisco, si el
puerto que nos llega es 0, es porque el servidor no acepta mas peticiones, por lo que no
recibiremos flujo.*/
    error = read(sfd_cliente,buffer,1024);
    if (buffer !=0 ){
         printf("Te llegará flujo por el puerto: %s \n",buffer);
    }else{
         printf("no se pueden establecer mas conexiones...\n");
    }
}
```

Librería: Flowdata.h

```
#ifndef FLOWDATA_H
#define FLOWDATA_H
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#define FLOW_VERSION_1         1
#define V1FLOWS_PER_PAK        24
#define FLOW_VERSION_5         5
#define V5FLOWS_PER_PAK        30
#define ulong      in_addr_t
#define ushort     int16_t
#define uchar      int8_t
#define ipaddrtype in_addr_t
#ifndef MAX
#define MAX(a,b) ((a) >= (b) ? (a) : (b))
#endif
typedef struct {
    ushort version;         /* 1 for now. */
    ushort count;           /* The number of records in PDU. */
    ulong  SysUptime;       /* Current time in millisecs since router booted */
    ulong  unix_secs;       /* Current seconds since 0000 UTC 1970 */
    ulong  unix_nsecs;      /* Residual nanoseconds since 0000 UTC 1970 */
} Flow1StatHdr;
typedef struct {
    ipaddrtype srcaddr;    /* Source IP Address */
    ipaddrtype dstaddr;    /* Destination IP Address */
    ipaddrtype nexthop;    /* Next hop router's IP Address */
    ushort input;          /* Input interface index */
    ushort output;         /* Output interface index */
    ulong dPkts;           /* Packets sent in Duration */
    ulong dOctets;         /* Octets sent in Duration. */
    ulong First;           /* SysUptime at start of flow */
    ulong Last;            /* and of last packet of flow */
    ushort srcport;        /* TCP/UDP source port number or equivalent */
    ushort dstport;        /* TCP/UDP destination port number or equivalent */
    ushort pad;
    uchar  prot;           /* IP protocol, e.g., 6=TCP, 17=UDP, ... */
    uchar  tos;            /* IP Type-of-Service */
    uchar  flags;          /* Reason flow was discarded, etc...  */
    uchar  tcp_retx_cnt;   /* Number of mis-seq with delay > 1sec */
    uchar  tcp_retx_secs;  /* Cumulative seconds between mis-sequenced pkts */
    uchar  tcp_misseq_cnt; /* Number of mis-sequenced tcp pkts seen */
    ulong  reserved;
```

```
} IPFlow1Stat;
typedef struct {
    Flow1StatHdr header;
    IPFlow1Stat  records[V1FLOWS_PER_PAK];
} IPStat1Msg;
typedef struct {
    ushort version;
    ushort count;           /* The number of records in PDU. */
    ulong  SysUptime;       /* Current time in millisecs since router booted */
    ulong  unix_secs;       /* Current seconds since 0000 UTC 1970 */
    ulong  unix_nsecs;      /* Residual nanoseconds since 0000 UTC 1970 */
    ulong  flow_sequence;   /* Seq counter of total flows seen */
    ulong  reserved;
} Flow5StatHdr;
typedef struct {
    ipaddrtype srcaddr;    /* Source IP Address */
    ipaddrtype dstaddr;    /* Destination IP Address */
    ipaddrtype nexthop;    /* Next hop router's IP Address */
    ushort input;          /* Input interface index */
    ushort output;         /* Output interface index */
    ulong dPkts;           /* Packets sent in Duration */
    ulong dOctets;         /* Octets sent in Duration. */
    ulong First;           /* SysUptime at start of flow */
    ulong Last;            /* and of last packet of flow */
    ushort srcport;        /* TCP/UDP source port number or equivalent */
    ushort dstport;        /* TCP/UDP destination port number or equivalent */
    uchar  pad;
    uchar  tcp_flags;      /* Cumulative OR of tcp flags */
    uchar  prot;           /* IP protocol, e.g., 6=TCP, 17=UDP, ... */
    uchar  tos;            /* IP Type-of-Service */
    ushort src_as;         /* originating AS of source address */
    ushort dst_as;         /* originating AS of destination address */
    uchar  src_mask;       /* source address prefix mask bits */
    uchar  dst_mask;       /* destination address prefix mask bits */
    ushort reserved;
} IPFlow5Stat;
typedef struct {
    Flow5StatHdr header;
    IPFlow5Stat  records[V5FLOWS_PER_PAK];
} IPStat5Msg;
#define MAX_V1_FLOW_PAK_SIZE (sizeof(Flow1StatHdr) + \
                              sizeof(IPFlow1Stat) * V1FLOWS_PER_PAK)
#define MAX_V5_FLOW_PAK_SIZE (sizeof(Flow5StatHdr) + \
                              sizeof(IPFlow5Stat) * V5FLOWS_PER_PAK)
#define MAX_FLOW_PAK_SIZE MAX(MAX_V1_FLOW_PAK_SIZE,  \
                                 MAX_V5_FLOW_PAK_SIZE)
```

```
ushort getVersionNumber(flow)
char* flow;
{
  return *((ushort*)flow);
}
#undef ulong  /* Don't leave these lying around! */
#undef ushort
#undef uchar
#undef ipaddrtype
#endif  /* FLOWDATA_H */
```

Bibliografía

[1] Active IETF Working Groups

URL: http://www.ietf.org/html.charters/wg-dir.html

[2] Cisco Systems, Inc, "NetFlow Services and Applications"

URL: http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm

[3] Luciano Moreno. "Redes - El modelo de referencia OSI - TCP-IP"

URL: http://www.htmlweb.net/redes/osi/osi_1.html

[4] The Internet Engineering Task Force

URL: http://www.ietf.org/

[5] Sun Microsystems, Inc. "Web de la compañía SUN."

URL: http://java.sun.com/

[6] Sun Microsystems, Inc. "Java 2 Platform, Standard Edition (J2SE)"

URL: http://java.sun.com/j2se/

[7] Programación en Java y ejemplos

URL: http://java.programacion.net/

[8] N. Brownlee, "Web de la universidad de Auckland sobre NeTraMet"

URL: http://www2.auckland.ac.nz/net/NeTraMet/

[9] N. Brownlee, "Web de la universidad de Auckland sobre RTFM(Realtime Traffic Flow Measurement)"

URL: http://www2.auckland.ac.nz/net//Internet/rtfm/

[10] C. Mills, D. Hirsh, G. Ruth. "Internet Accounting Background", RFC 1272

URL: http://www.faqs.org/rfcs/rfc1272.html

[11] N. Brownlee, C. Mills, G. Ruth. "Traffic Flow Measurement: Architecture", RFC 2063

URL: http://www.faqs.org/rfcs/rfc2063.html

[12] N. Brownlee. "Traffic Flow Measurement: Experiences with NeTraMet", RFC 2123

URL: http://www.faqs.org/rfcs/rfc2123.html

[13] N. Brownlee. "Traffic Flow Measurement: Meter MIB", RCF 2720

URL: http://www.faqs.org/rfcs/rfc2720.html

[14] N. Brownlee. "RTFM: Applicability Statement", RFC 2721

URL: http://www.faqs.org/rfcs/rfc2721.html

[15] N. Brownlee, C. Mills, G. Ruth. "Traffic Flow Measurement: Architecture", RFC 2722

URL: http://www.faqs.org/rfcs/rfc2722.html

[16] N. Brownlee. "SRL: A Language for Describing Traffic Flows and Specifying Actions for Flow Groups". RFC 2723

URL: http://www.faqs.org/rfcs/rfc2723.html

[17] N. Brownlee, G. Ruth, S. Handelman, S. Stibler. "RTFM: New Attributes for Traffic Flow Measurement", RFC 2724

URL: http://www.faqs.org/rfcs/rfc2724.html

[18] N. Brownlee. "FTP de la Universidad de Auckland"

URL: ftp://ftp.auckland.ac.nz/pub/iawg/NeTraMet/

[19] N. Brownlee. "History of the Internet Accounting Working Group"

URL: http://www2.auckland.ac.nz/net/Internet/rtfm/meetings/34-dallas/iawg-hist/iawg-hist-all.html

[20] J. Case, M. Fedor, M. Schoffstall, J. Davin. "A Simple Network Management Protocol", RFC 1157

URL: http://www.faqs.org/rfcs/rfc1157.html

[21] K. McCloghrie, M. Rose. "Management Information Base for Network Management of TCP/IP-based internets: MIB-II", RFC 1213

URL: http://www.faqs.org/rfcs/rfc1213.html

[22] Kathy Walrath and Mary Campione. "The JFC Swing Tutorial (A Guide to Constructing GUIs)". Ed. Addison-Wesley. ISBN 0-201-32576-4