ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Desarrollo de una herramienta software de programación del router Cisco 2600

AUTOR: Juan Asensio Sánchez
DIRECTORA: María Dolores Cano Baños

Marzo de 2004

Autor	Juan Asensio Sánchez
E-mail del Autor	jas_esp@hotmail.com
Director(es)	María Dolores Cano Baños
E-mail del Director	mdolores.cano@upct.es
Título del PFC	Desarrollo de una herramienta software de programación del router Cisco 2600
Descriptores	Java, Routers Cisco, Software de configuración, Calidad de Servicio

Resúmen

El objetivo del proyecto es, previo estudio de las posibilidades de configuración de los routers Cisco de la serie 2600, el desarrollo de un programa que permita realizar todas las tareas relacionadas con la configuración de Servicios Diferenciados, además de opciones de configuración básicas, en dicho equipo. Las herramientas de servicios diferenciados incluirán la generación de listas de control de acceso, la creación de mapas de clases, adición y modificación de mapas de políticas y, por último, la asociación de las anteriores políticas a las interfaces. El programa se implementará en Java, y será exportable tanto a plataformas Windows como Linux/Unix.

Titulación	Ingeniero Técnico de Telecomunicación, especialidad en Telemática
Intensificación	
Departamento	Departamento de Tecnologías de la Información y las Comunicaciones
Fecha de Presentación	Marzo de 2004

Índice de Contenidos

Capítulo 1
Introducción

1.1 Introducción

El presente proyecto Final de Carrera guiará al lector durante el proceso de realización de una herramienta software para la configuración y programación de los *routers* Cisco de la serie 2600. Estos *router* están ampliamente instalados en entornos empresariales y soportan gran parte del tráfico que circula actualmente por Internet.

Los *routers* de Cisco poseen un sistema operativo propio, el Cisco IOS, que permite la configuración de los mismos mediante la introducción de comandos en línea a través de programas tales como el HyperTerminal de Windows o Kermit en Linux. Otras alternativas para configurar los *routers* Cisco es el software Cisco ConfigMaker. Mediante esta herramienta se configuran las opciones básicas de los enrutadores en un ordenador cliente, sin necesidad de tener el *router* conectado, generando un archivo de configuración que luego se copiará en la memoria del *router*, sin ser posible con esta herramienta la configuración de opciones tales como servicios diferenciados. CiscoView es otra herramienta desarrollada por el fabricante mediante la cual tan solo es posible mostrar el estado de las interfaces de los dispositivos, mostrando si hay algún problema. Por último, existe la opción de configuración del *router* a través de la web, aunque este método está muy limitado, a la par de haber sido necesario haberlo configurado anteriormente de algún otro modo y condicionado al funcionamiento de la red.

La herramienta a desarrollar en este proyecto permitirá poder configurar el *router* sin necesidad de tener que conocer en profundidad los comandos necesarios para tal configuración, de una manera gráfica e intuitiva, además de obtener una mayor velocidad en los procesos de configuración, y por tanto, mayor rendimiento y mejores prestaciones. Será posible conocer y observar en todo momento los pasos que se siguen para la configuración de las opciones implementadas, mediante una ventana de terminal, algo que no es posible mediante otras herramientas como Cisco ConfigMaker.

La herramienta permitirá configurar opciones básicas en un *router* cualquiera, como son:

- Direcciones de interfaces
- Enrutamiento
- Contraseña de administración
- Etcétera.

Y otras más avanzadas como las opciones relacionadas con la calidad de servicio (*Quality of Service*, QoS). Entenderemos por calidad de servicio la arquitectura que permite garantizar diferentes niveles de calidad de servicio a las aplicaciones de los usuarios finales. Un ejemplo de esta calidad de servicio sería garantizar a las aplicaciones de Voz sobre IP un bajo retardo en sus comunicaciones, mediante el uso de colas prioritarias en las máquinas que gestionan las redes.

Entre las opciones de configuración en este sentido se incluyen:

- Listas de control de acceso
- Mapas de clases
- Mapas de políticas

La herramienta será desarrollada en lenguaje Java, un lenguaje multiplataforma y en continua evolución. Se ha elegido este lenguaje ya que en la actualidad está muy extendido y permite ejecutar las aplicaciones en distintos sistemas operativos sin tener que recompilarlos; simplemente es necesario que haya una máquina virtual de Java para tal plataforma.

Capítulo 2
Conocimientos previos I: Java

2.1 Introducción

En este segundo capítulo se dará una breve introducción a la tecnología Java [J1], los fundamentos del lenguaje y los principales aspectos que se han tenido en cuenta a la hora de realizar el proyecto, tales como los componentes de la interfaz gráfica y su modo de utilización y la librería de comunicaciones de Java [J3], que no pertenece a la distribución por defecto (JSE, *Java Standard Edition*) del JSDK (*Java Software Development Kit*, Kit de Desarrolladores de Software de Java), sino que es un paquete que se distribuye por separado.

2.2 Orígenes de Java

2.2.1 Antecedentes

Java fue diseñado por James Gosling, de Sun Microsystems, en 1990, como software para dispositivos electrónicos de consumo [J4] [J6]. En los primeros años de la década de los noventa, Sun Microsystems decidió intentar introducirse en el mercado de la electrónica de consumo y desarrollar programas para pequeños dispositivos electrónicos. Tras unos comienzos dudosos, Sun decidió crear una filial, denominada FirstPerson Inc., para dar margen de maniobra al equipo responsable del proyecto.

Figura 1 - Logo de la empresa Sun

Inicialmente Java se llamó OAK (roble en inglés), aunque tuvo que cambiar debido a la existencia de que dicho nombre ya estaba registrado por otra empresa. Tres de las principales razones que llevaron a crear Java son las siguientes:

1. Creciente necesidad de interfaces mucho más cómodos e intuitivos que los sistemas de ventanas que proliferaban hasta el momento.
2. Fiabilidad del código y facilidad de desarrollo. Gosling observó que muchas de las características que ofrecían C o C++ para este tipo de dispositivos aumentaban de forma alarmante el gran coste de pruebas y depuración. Por ello en sus ratos libres creó un lenguaje de programación donde intentaba solucionar los fallos que encontraba en C++.
3. El funcionamiento de dispositivos electrónicos se controla mediante la utilización de microprocesadores de bajo precio y reducidas prestaciones, de los que existe una diversidad abrumadora. Además, se desarrollan nuevos modelos más perfeccionados y baratos cada pocas semanas y los fabricantes tratan siempre de aprovecharlos para reducir costes en las grandes producciones que realizan. Sin embargo, las diferentes familias de microprocesadores emplean un juego de instrucciones diferentes, e incluso las nuevas versiones de modelos antiguos suelen añadir o modificar instrucciones para aprovechar al máximo las nuevas posibilidades

de los chips. Por este motivo, el software escrito debe ser revisado por completo antes de poder utilizarlo en otro. Naturalmente, la existencia de compiladores alivia esta labor, facilitando la utilización de C en lugar de ensamblador, pero esto no es suficiente: muchas de las diferencias básicas entre microprocesadores (direcciones concretas de memoria, tamaño y número de registros, acceso a puertos de entrada/salida) se manifiestan directamente en el código C, del que debe desarrollarse una nueva versión para cada chip que desea utilizarse.

Por todo ello, en lugar de tratar únicamente de optimizar las técnicas de desarrollo y dar por sentado la utilización de C o C++, el equipo de Gosling se planteó que tal vez estos lenguajes eran demasiado complicados como para conseguir reducir de forma apreciable la complejidad asociada a este campo. Por este motivo, su primera propuesta fue idear un nuevo lenguaje de programación lo más sencillo posible, con el objeto de que se pudiese adaptar con facilidad a cualquier entorno de ejecución. Basándose en el conocimiento y estudio de gran cantidad de lenguajes, este grupo decidió recoger las características esenciales que debía tener un lenguaje de programación moderno y potente, pero eliminando todas aquellas funciones que no eran absolutamente imprescindibles.

2.2.2 Primeros proyectos en Java

El proyecto Green fue el primero en el que aplicó Java, y consistía en un sistema de control completo de los aparatos electrónicos y el entorno de un hogar. Con este fin se construyó un ordenador experimental denominado *7 (*Star Seven*). El sistema presentaba una interfaz basada en la representación de la casa de forma animada y el control se llevaba a cabo mediante una pantalla sensible al tacto. En el sistema aparecía ya Duke, la actual mascota de Java.

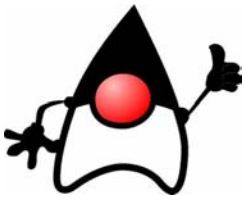


Figura 2 - Duke, mascota de Java

Más tarde, Java se aplicó a otro proyecto denominado VOD (*Video On Demand*, Vídeo bajo demanda) en el que se empleaba como interfaz para la televisión interactiva que, como hemos comentado anteriormente, se pensaba iba a ser el principal campo de aplicación de Java. Ninguno de estos proyectos se convirtió nunca en un sistema comercial, pero fueron desarrollados enteramente en un Java primitivo.

Una vez que en Sun se dieron cuenta de que a corto plazo la televisión interactiva no iba a ser un gran éxito, urgieron a FirstPerson a desarrollar con rapidez nuevas estrategias que produjeran beneficios entre los que se encontraba su aplicación en Internet aunque, en ese momento no se juzgó productivo y FirstPerson cerró en la primavera de 1994.

2.2.3 Resurgimiento de Java

Aunque Java no llegó a caer en un olvido, lo cierto es que tuvo que ser Bill Joy, cofundador de Sun y uno de los desarrolladores principales del sistema operativo Unix de Berkeley, el que le sacó de él, ya que juzgó que Internet podría llegar a ser el campo de juego adecuado para disputar a Microsoft su primacía casi absoluta en el terreno del software, y vio en Oak el instrumento idóneo para llevar a cabo estos planes. Para poderlo presentar en sociedad se tuvo que modificar el nombre de este lenguaje de programación y se tuvo que realizar una serie de modificaciones de diseño para poderlo adaptar al propósito mencionado. Y así Java fue presentado en sociedad en agosto de 1995.

Algunas de las razones que llevaron a Bill Joy a pensar que Java podría llegar a ser rentable son:

- Es un lenguaje sencillo, por lo que el entorno necesario para su ejecución es de pequeño tamaño y puede adaptarse incluso al interior de un navegador. Eso supone que Java es, verdaderamente, un lenguaje multiplataforma.
- La sencillez no está reñida con la potencia. Java es un lenguaje orientado a objetos que permite abordar la resolución de cualquier tipo de problema.
- La ejecución del código Java es segura y fiable, y los programas no acceden directamente a la memoria del ordenador. Esto facilita su confinamiento en una zona aislada, siendo imposible que un programa escrito en Java pueda acceder a los recursos del ordenador sin que esta operación le sea permitido de forma explícita. De este modo, los datos del usuario quedan a salvo de la existencia de posibles virus escritos en Java, que podrían traerse de Internet de forma inadvertida. La ejecución segura y controlada del código Java es una característica única, que no puede encontrarse en ninguna otra tecnología.
- La integración de Java dentro de los navegadores más populares supuso por primera y única vez la posibilidad de realizar uno de los principales objetivos que se habían perseguido durante años: escribir un programa en cualquier parte y que éste pudiese funcionar sin cambios en cualquier otro ordenador independientemente de su marca y del sistema operativo utilizado. Las consecuencias de la utilización de Java junto a la expansión universal de Internet todavía están comenzando a vislumbrarse.

Cabe mencionar también su capacidad multi-hilo, su robustez o lo integrado que tiene el protocolo TCP/IP, lo que lo hace un lenguaje ideal para Internet. Pero es su sencillez, portabilidad y seguridad lo que le han hecho un lenguaje de tanta importancia.

2.2.4 Futuro de Java

Existen muchas críticas a Java debido a su lenta velocidad de ejecución, aproximadamente unas 20 veces más lento que un programa en lenguaje C. Sun está trabajando intensamente en crear versiones de Java con una velocidad mayor.

Figura 3 - La taza de café característica de Java

El problema fundamental de Java es que utiliza *bytecodes* para solventar los problemas de portabilidad, que posteriormente en cada máquina se tendrán que transformar en código máquina, lo que ralentiza considerablemente el proceso de ejecución.

La solución que se deriva de esto parece bastante obvio: fabricar ordenadores capaces de comprender directamente los *bytecodes*; sería una máquina que utilizara Java como sistema operativo y que no requeriría en principio de disco duro porque regiría sus recursos de la red. A los ordenadores que utilizan Java como sistema operativo se les llama *Network Computer*, *WePC* o *WebTop*.

La primera gran empresa que se ha dado cuenta de todo esto y ha apostado en esto ha sido Oracle, que en enero de 1996 presentó en Japón su primer NC (*Network Computer*), basado en un procesador RISC con 8 Mbytes de RAM. Tras Oracle, han sido compañías del tamaño de Sun, Apple e IBM las que han anunciado desarrollos similares.

Por ahora, toda esta tecnología está comenzando a desarrollarse, aunque en la actualidad los anchos de banda y el coste de las conexiones no favorecen demasiado estos ordenadores. No obstante todos estos problemas desaparecerán a corto plazo, y la actitud de

Sun de mantener un sistema transparente y accesible a cualquier programador parece que van a ser determinantes en la comercialización de este tipo de sistemas.

La principal empresa en el mundo del software, Microsoft, que en los comienzos de Java no estaba a favor de su utilización, ha licenciado Java, lo ha incluido en Internet Explorer 3.0 y posteriores, y ha lanzado un entorno de desarrollo para Java, que se denomina Visual J++.

2.3 Características de Java

No es arriesgado afirmar que Java supone un significativo avance en el mundo de los entornos software, y esto viene avalado por tres elementos claves que diferencian a este lenguaje desde un punto de vista tecnológico:

- Pone al alcance de cualquiera la utilización de *applets*.
- Es un lenguaje de programación que ofrece la potencia del diseño orientado a objetos con una sintaxis fácilmente accesible y un entorno robusto y agradable.
- Proporciona un conjunto de clases potente y flexible.

No en vano, Java aporta a la Web una interactividad que se había buscado durante mucho tiempo entre usuario y aplicación. Viendo más en detalle sus características, descubriríamos los siguientes aspectos.

2.3.1 Applets

Un *applet* es un pequeño programa en Java transferido dinámicamente a través de Internet. Es inteligente, pudiendo reaccionar a la entrada de un usuario y cambiar de forma dinámica. Sin embargo, la verdadera novedad es el gran potencial que Java proporciona en este aspecto, haciendo posible que los programadores ejerzan un control sobre los programas ejecutables de Java que no es posible encontrar en otros lenguajes.

2.3.2 Simpleza

El único requerimiento para aprender Java es tener una comprensión de los conceptos básicos de la programación orientada a objetos; se ha pretendido un lenguaje simple sin caer en la ineficacia y la falta de expresividad, pudiendo mostrarse cualquier planteamiento por parte del programador sin que las interioridades del sistema subyacente sean desveladas.

2.3.3 Potencia

Pese a su simpleza se ha conseguido un considerable potencial, y aunque cada tarea se puede realizar de un número reducido de formas, se ha conseguido un gran potencial de expresión e innovación desde el punto de vista del programador.

2.3.4 Seguridad

Existe una preocupación lógica en Internet por el tema de la seguridad: virus, caballos de Troya, y similares navegan de forma usual por la red, constituyendo una amenaza palpable. Java ha sido diseñado poniendo un énfasis especial en el tema de la seguridad, y se ha conseguido lograr cierta inmunidad en el aspecto de que un programa realizado en Java no puede realizar llamadas a funciones globales y acceder a recursos arbitrarios del sistema, por

lo que el control sobre los programas ejecutables no es equiparable a otros lenguajes. Niveles de seguridad:

- Fuertes restricciones al acceso a memoria, como son la eliminación de punteros aritméticos y de operadores ilegales de transmisión.
- Rutina de verificación de bytecode que asegura que no se viole ninguna construcción del lenguaje.
- Verificación del nombre de clase y de restricciones de acceso durante la carga.
- Sistema de seguridad de la interfaz que refuerza las medidas de seguridad en muchos niveles.

2.3.5 Orientación a objetos

Java fue diseñado en este aspecto partiendo de cero, no es un derivado de otro lenguaje anterior y no tiene compatibilidad con ninguno de ellos. En Java el concepto de objeto resulta sencillo y fácil de ampliar conservando elementos "no objetos", como números y otros tipos simples. Proporciona un mecanismo de clasificación simple y presenta un modelo de interfaz dinámica intuitiva.

2.3.6 Robusto

Java verifica su código al mismo tiempo que lo escribe, y una vez más antes de ejecutarse, de manera que se consigue un alto margen de codificación sin errores. Se realiza un descubrimiento de la mayor parte de los errores durante el tiempo de compilación, ya que Java es estricto en cuanto a tipos y declaraciones, y lo que es rigidez y falta de flexibilidad se convierte en eficacia. Respecto a la gestión de memoria, Java libera al programador del compromiso de tener que controlar especialmente la asignación que de ésta hace a sus necesidades específicas. Este lenguaje posee una gestión avanzada de memoria llamada gestión de basura, y un manejo de excepciones orientado a objetos integrados. Estos elementos realizarán muchas tareas antes tediosas a la vez que obligadas para el programador.

2.3.7 Interactivo

Uno de los requisitos de Java desde sus inicios fue la posibilidad de crear programas en red interactivos, por lo que es capaz de hacer varias cosas a la vez sin perder el rastro de lo que debería suceder y cuándo. Para que esto sea posible cuenta con múltiples hilos de utilización sencilla que le permiten pensar en el comportamiento específico que se intenta codificar, sin tener que integrarlo en un modelo de programación global de "bucle de suceso".

2.3.8 Arquitectura neutral

La portabilidad se consigue haciendo de Java un lenguaje medio interpretado y medio compilado. Es decir, a partir del código fuente se compila a un lenguaje intermedio cercano al lenguaje máquina pero independiente del ordenador y el sistema operativo en que se ejecuta (llamado en el mundo Java *bytecode*) y, finalmente, se interpreta ese lenguaje intermedio por medio de un programa denominado máquina virtual de Java.

Este esquema lo han seguido otros lenguajes, como por ejemplo Visual Basic. Sin embargo, nunca se había empleado como punto de partida a un lenguaje multiplataforma ni se había hecho de manera tan eficiente. Cuando Java apareció en el mercado se hablaba de que era entre 10 y 30 veces más lento que C++. Ahora, con los compiladores JIT (*Just in Time*) se habla de tiempos entre 2 y 5 veces más lentos. Con la potencia de las máquinas actuales, esa

lentitud es un precio que se puede pagar sin problemas contemplando las ventajas de un lenguaje portable.

2.3.9 Fácil aprendizaje

Java es más complejo que un lenguaje simple, pero más sencillo que cualquier otro entorno de programación. El único obstáculo que se puede presentar es la consecución de la comprensión de la programación orientada a objetos, aspecto que es independiente del lenguaje y se presenta pues como insalvable.

2.3.10 Entorno de objeto rico

Existen varias clases que contienen las abstracciones básicas para el mundo con el que interaccionan sus programas. Se contará por tanto, con un conjunto de clases comunes que pueden crecer para admitir todas las necesidades del programador. Además la biblioteca de clases de Java proporciona un modelo unificador único de protocolos de Internet. El conjunto de clases más complicado de Java es la AWT (*Abstract Window Toolkit*, equipo de herramientas de ventana abstractas) y Swing. Estas clases implementan los componentes de la interfaz de usuario gráfica básica de Xerox PARC ’80, Macintosh ’84, X/Motif ’88 y Windows ’95 que son comunes a todos los ordenadores personales modernos.

2.3.11 Trabajo en red

Java anima las páginas Web y hace posible la incorporación de aplicaciones interactivas y especializadas. Aporta la posibilidad de distribuir contenidos ejecutables, de manera que los suministradores de información de la Web pueden crear una página hipertexto con una interacción continuada y compleja en tiempo real; el contenido ejecutable es transferido literalmente al ordenador del usuario.

Los protocolos básicos para trabajar en Internet están encapsulados en unas cuantas clases simples. Se incluyen implementaciones ampliables de FTP, HTTP, NNTP, SMTP junto con conectores de red de bajo nivel e interfaces de nombrado. Esto le permite interactuar con esos servicios de red poderosos sin tener que comprender realmente los detalles de bajo nivel de esos protocolos. Este lenguaje está diseñado para cumplir los requisitos de entrega de contenidos interactivos mediante el uso de *applets* insertados en sus páginas HTML. Además, las clases de Java admiten muy bien estos protocolos y formatos. El envío de las clases de Java a través de Internet se realiza con gran facilidad, ya que existe una interfaz unificada, resolviendo así los típicos problemas de diferencia de versiones.

Java proporciona un conjunto de clases para tratar con una abstracción de los conectores de red (*sockets*) originales de la versión UNIX de Berkeley. Este paquete encapsula la noción de una dirección de Internet y de flujos de E/S hacia y desde conectores de otras máquinas de Internet.

Con todas estas posibilidades aumenta el dinamismo y competitividad de la Web, puesto que es capaz de captar el interés del usuario durante largo tiempo y permite a los programadores convertir la Web en un sistema de entrega de software.

2.3.12 Lenguaje

Java fue desarrollado basándose en C++, pero eliminando rasgos del mismo poco empleados, optándose por una codificación comprensible. Básicamente, encontramos las siguientes diferencias con C++:

- Java no soporta los tipos *struct*, *union* y *pointer*.

- No soporta *typedef* ni *#define*.
- Se distingue por su forma de manejar ciertos operadores y no permite una sobrecarga del operador.
- No soporta herencia múltiple.
- Java maneja argumentos en la línea de comandos de forma diversa a como lo hacen C o C++.
- Tiene una clase `String` que es parte del paquete `java.lang` y se diferencia de la matriz de caracteres terminada con un nulo que usan C y C++.
- Java cuenta con un sistema automático para asignar y liberar memoria, con lo que no es necesario utilizar las funciones previstas con este fin en C y C++.
- En Java podemos crear los siguientes tipos de construcciones:
    - Miniaplicaciones (*applets*).
    - Aplicaciones. Se ejecutan sin necesidad de un navegador.
    - Manipuladores de protocolo. Interpretan protocolos.
    - Manipuladores de contenido. Interpretan archivos.
    - Métodos nativos. Métodos declarados en una clase Java pero implementados en C.

2.3.13 Utilidades

El paquete de utilidades de Java viene con un conjunto completo de estructuras de datos complejas y sus métodos asociados, que serán de inestimable ayuda para implementar *applets* y otras aplicaciones más complejas. Se dispone también de estructuras de datos habituales, como son tablas, pilas,… como clases claramente implementadas. Existirá una interfaz `Observer` que permitirá la implementación simple de objetos dinámicos visualizando su estado en pantalla. Otro tipo de software disponible es el manipulador; un manipulador de protocolo permitirá al programador especificar cómo debe interpretar el programa de navegación un tipo especial de protocolo. Existen otros manipuladores, como son los de contenido, que traducen una especificación particular de un tipo de archivo basado en las *Multipurpose Internet Mail Extensions* (MIME, Extensiones multipropósito para correo de Internet). Especificará el manejo de un determinado tipo de archivos. El *Java Development Kit* suministrado por Sun Microsystems incluye los siguientes elementos:

- *Java Applet Viewer*. Visualizador de miniaplicaciones Java. Permite ejecutarlas sin crear una página HTML referida a ellas.
- *Java Compiler*.- compilador de Java.
- *Java Languaje Runtime*. Entorno de interpretación de aplicaciones.
- *Java Debugger API* y *Prototype Debugger*. Depurador de la línea de comandos.

2.3.14 Gestor de la entrada/salida

Se usa un modelo de flujos uniformes para traducirlo a todas las formas de E/S, de modo que independientemente de con quién esté hablando el código. Java proporciona un extenso conjunto de clases para el manejo de E/S desde diferentes dispositivos; los objetos `InputStream` y `OutputStream` serán los que esconderán todos los detalles del caso concreto que se está tratando.

2.3.15 Animación e interacción

Las aplicaciones de Java permiten situar figuras animadas en las páginas Web, y éstas pueden concebirse con logotipos animados o con texto que se desplace por la pantalla. También pueden tratarse gráficos generados por algún proceso. Estas animaciones pueden ser interactivas, permitiendo al usuario un control sobre su apariencia.

2.4 Fundamentos del lenguaje

El lenguaje Java es un lenguaje orientado a objetos, que toma sus características principalmente de C++ y C.

El lenguaje Java es a la vez compilado e interpretado. Con el compilador se convierte el código fuente que reside en archivos cuya extensión es *.java*, a un conjunto de instrucciones que recibe el nombre de *bytecodes* que se guardan en un archivo cuya extensión es *.class*. Estas instrucciones son independientes del tipo de ordenador. El intérprete ejecuta cada una de estas instrucciones en un ordenador específico (Windows, Macintosh, Linux, Solares, etc.). Solamente es necesario, por tanto, compilar una vez el programa, pero se interpreta cada vez que se ejecuta en un ordenador.

Cada intérprete Java es una implementación de la Máquina Virtual Java (*Java Virtual Machine*, JVM). Los *bytecodes* posibilitan el objetivo de "*write once, run anywhere*", de escribir el programa una vez y que se pueda correr en cualquier plataforma que disponga de una implementación de la JVM. Por ejemplo, el mismo programa Java puede correr en Windows 98, Solaris, Macintosh, Linux, etc.

Básicamente, se pueden tener dos tipos de programas en Java, dependiendo de dónde se produzca la ejecución de los mismos:

- Aplicaciones: se ejecutan como otra aplicación en el sistema operativo.
- *Applets*: se ejecutan dentro de un navegador de web.

Java es, por tanto, algo más que un lenguaje, ya que la palabra Java se refiere a dos cosas inseparables: el lenguaje que nos sirve para crear programas y la Máquina Virtual Java que sirve para ejecutarlos.

2.4.1 La máquina virtual de Java

La Máquina Virtual Java (JVM) es el entorno en el que se ejecutan los programas Java. Su misión principal es la de garantizar la portabilidad de las aplicaciones Java. Define esencialmente un ordenador abstracto y especifica las instrucciones (*bytecodes*) que este ordenador puede ejecutar. El intérprete Java específico ejecuta las instrucciones que se guardan en los archivos cuya extensión es *.class*. Las tareas principales de la JVM son las siguientes:

- Reservar espacio en memoria para los objetos creados.
- Liberar la memoria no usada (*Garbage Collector*).
- Asignar variables a registros y pilas.
- Llamar al sistema huésped para ciertas funciones, como los accesos a los dispositivos.
- Vigilar el cumplimiento de las normas de seguridad de las aplicaciones Java.

Esta última tarea, es una de las más importantes que realiza la JVM. Además, las propias especificaciones del lenguaje Java contribuyen extraordinariamente a este objetivo:

- Las referencias a *arrays* son verificadas en el momento de la ejecución del programa.

- No hay manera de manipular de forma directa los punteros.
- La JVM gestiona automáticamente el uso de la memoria, de modo que no queden huecos.
- No se permiten realizar ciertas conversiones (*casting*) entre distintos tipos de datos.

2.4.2 El lenguaje Java

Java es un lenguaje de programación orientado a objetos puro, en el sentido de que no hay ninguna variable, función o constante que no esté dentro de una clase [J2]. Se accede a los miembros dato y las funciones miembro a través de los objetos y de las clases. Por razones de eficiencia, se han conservado los tipos básicos de datos `int`, `float`, `double`, `char`, etc, similares a los del lenguaje C/C++.

Los tipos de programas más comunes que se pueden hacer con Java son los *applets* (se ejecutan en el navegador de la máquina cliente) y las aplicaciones (programas que se ejecutan directamente en la JVM). Otro tipo especial de programa se denomina *servlet* que es similar a los *applets* pero se ejecutan en los servidores web Java.

La API de Java es muy rica; está formada por un conjunto de paquetes de clases que le proporcionan una gran funcionalidad. El núcleo de la API viene con cada una de las implementaciones de la JVM:

- Lo esencial: tipos de datos, clases y objetos, *arrays*, cadenas de caracteres (*strings*), hilos (*threads*), entrada/salida, propiedades del sistema, etc.
- *Applets*.
- Manejo de la red (*networking*).
- Internacionalización.
- Seguridad.
- Componentes (*JavaBeans*).
- Persistencia (*Object serialization*).

2.4.3 Herencia y polimorfismo

La herencia es una propiedad esencial de la Programación Orientada a Objetos que consiste en la creación de nuevas clases a partir de otras ya existentes. La herencia es la característica fundamental que distingue un lenguaje orientado a objetos, como el C++ o Java, de otro convencional como C, BASIC, etc. Java permite heredar a las clases características y conductas de una o varias clases denominadas base. Las clases que heredan de clases base se denominan derivadas; éstas a su vez pueden ser clases bases para otras clases derivadas.

La herencia ofrece una ventaja importante: permite la reutilización del código. Una vez que una clase ha sido depurada y probada, el código fuente de dicha clase no necesita modificarse. Su funcionalidad se puede cambiar derivando una nueva clase que herede la funcionalidad de la clase base y le añada otros comportamientos. Reutilizando el código existente, el programador ahorra tiempo y dinero, ya que solamente tiene que verificar la nueva conducta que proporciona la clase derivada.

En el lenguaje Java, todas las clases derivan implícitamente de la clase base `Object`, por lo que heredan las funciones miembro definidas en dicha clase. Las clases derivadas pueden redefinir algunas de estas funciones miembro como `toString()` y definir otras nuevas.

El polimorfismo se implementa por medio de las funciones abstractas; en las clases derivadas se declara y se define una función que tiene el mismo nombre, el mismo número de parámetros y del mismo tipo que en la clase base, pero que da lugar a un comportamiento

distinto, específico de los objetos de la clase derivada. Enlace dinámico significa que la decisión sobre la función a llamar se demora hasta el tiempo de ejecución del programa.

2.4.4 Interfaces

Una interfaz es una colección de declaraciones de métodos (sin definirlos), que también puede incluir constantes. Por ejemplo, `Runnable` es un ejemplo de interfaz en el cual se declara, pero no se implementa, una función miembro `run()`. Las clases que implementen (`implements`) el interfaz `Runnable` han de definir obligatoriamente la función `run()`. El papel del interfaz es el de describir algunas de las características de una clase.

Un interfaz es simplemente una lista de métodos no implementados. Una clase abstracta puede incluir métodos implementados y no implementados o abstractos, miembros dato constantes y otros no constantes.

Una clase solamente puede derivar (`extends`) de una clase base, pero puede implementar varios interfaces. Los nombres de los interfaces se colocan separados por una coma después de la palabra reservada `implements`.

2.4.5 Excepciones

Las excepciones son el mecanismo por el cual pueden controlarse en un programa Java las condiciones de error que se producen. Estas condiciones de error pueden ser errores en la lógica del programa como un índice de un arreglo (*array*) fuera de su rango, una división por cero o errores disparados por los propios objetos que denuncian algún tipo de estado no previsto, o condición que no pueden manejar.

La idea general es que cuando un objeto encuentra una condición que no sabe manejar crea y dispara una excepción que deberá ser capturada por el que le llamó o por alguien más arriba en la pila de llamadas. Las excepciones son objetos que contienen información del error que se ha producido y que heredan de la clase `Throwable` o de la clase `Exception`. Si nadie captura la excepción interviene un manejador por defecto que normalmente imprime información que ayuda a encontrar quién produjo la excepción.

Existen dos categorías de excepciones:

- Excepciones verificadas: El compilador obliga a verificarlas. Son todas las que son lanzadas explícitamente por objetos de usuario.
- Excepciones no verificadas: El compilador no obliga a su verificación. Son excepciones como divisiones por cero, excepciones de puntero nulo, o índices fuera de rango.

Es posible crear excepciones propias creando nuevas clases que hereden de la clase `Exception`.

La captura de excepciones se realiza encerrando el cuerpo del código que pueda lanzar una cierta excepción en un bloque `try`, tras el cual deberá haber otro bloque `catch`, que manejará el error en caso de que se produzca. Final y opcionalmente, puede haber un bloque `finally`, que se ejecutará siempre, se lance o no la excepción.

2.4.6 Hilos (*Threads*)

La Máquina Virtual Java (JVM) es un sistema multi-hilo. Es decir, es capaz de ejecutar varias secuencias de ejecución (programas) simultáneamente. La JVM gestiona todos los detalles, asignación de tiempos de ejecución, prioridades, etc., de forma similar a como gestiona un Sistema Operativo múltiples procesos. La diferencia básica entre un proceso de Sistema Operativo y un *thread* Java es que los *threads* corren dentro de la JVM, que es un

proceso del Sistema Operativo y por tanto comparten todos los recursos, incluida la memoria y las variables y objetos allí definidos.

Java da soporte al concepto de *thread* desde el mismo lenguaje, con algunas clases e interfaces definidas en el paquete `java.lang` y con métodos específicos para la manipulación de *threads* en la clase `Object`. Desde el punto de vista de las aplicaciones, los *threads* son útiles porque permiten que el flujo del programa sea divido en dos o más partes, cada una ocupándose de alguna tarea. De hecho todos los programas con interfaces gráficos (AWT, *Abstract Window Toolkit*, o Swing) son multi-hilo porque los eventos y las rutinas de dibujado de las ventanas corren en un *thread* distinto al principal.

La forma más directa para hacer un programa multi-hilo es extender la clase `Thread`, y redefinir el método `run()`. Este método es invocado cuando se inicia el *thread* (mediante una llamada al método `start()` de la clase `Thread`). El *thread* se inicia con la llamada al método `run()` y termina cuando termina éste.

La interfaz `Runnable` proporciona un método alternativo a la utilización de la clase `Thread`, para los casos en los que no es posible hacer que nuestra clase extienda la clase `Thread` (Java no soporta herencia múltiple). En este caso nuestra clase debe implementar la interfaz `Runnable`, variando ligeramente la forma en que se crean e inician los nuevos *threads*. En este caso se deberá crear un nuevo objeto `Thread` pasándole como parámetro el objeto de la clase que implemente la interfaz `Runnable`. Tras ello, llamaremos al método `start()` del *thread* recién creado para comenzar la ejecución del hilo.

2.5 Entornos gráficos en Java

Las JFC (*Java Foundation Classes*) son parte de la API de Java compuesto por clases que sirven para crear interfaces gráficas visuales para las aplicaciones y *applets* de Java [J8]. Las JFC contienen dos paquetes gráficos: AWT (*Abstract Window Toolkit*) y Swing.

AWT presenta componentes pesados, que en cada plataforma sólo pueden tener una representación determinada. Está disponible desde la versión 1.1 del JDK como `java.awt`.

Swing presenta componentes ligeros, que pueden tomar diferente aspecto y comportamiento pues lo toman de una biblioteca de clases. Está disponible desde la versión 1.2 del JDK como `javax.swing` aunque antes se podían encontrar versiones previas como `com.sun.java` o como `java.awt.swing`.

Tanto AWT como Swing tienen en común un sistema para gestionar los eventos que se producen al interactuar con el usuario de la interfaz gráfica.

Para cada objeto que represente una interfaz gráfica, se pueden definir objetos "oyentes" (`Listener`), que esperen a que suceda un determinado evento sobre la interfaz. Por ejemplo, se puede crear un objeto oyente que esté a la espera de que el usuario pulse sobre un botón de la interfaz, y si esto sucede, él es avisado, ejecutando determinada acción. La clase base para todos estos eventos que se pueden lanzar es la clase `AWTEvent` (perteneciente al paquete `java.awt`). El modelo de eventos de AWT depende del paquete `java.awt.event`, que en Swing se amplía con el paquete `javax.swing.event`. Existen dos tipos básicos de eventos:

- Físicos: Corresponden a un evento hardware claramente identificable. Por ejemplo, se ha pulsado una tecla (`KeyStrokeEvent`).
- Semánticos: Se componen de un conjunto de eventos físicos, que sucedidos en un determinado orden tienen un significado más abstracto: El usuario ha elegido un elemento de una lista desplegable (`ItemEvent`).

Dado que el paquete Swing es más reciente, más versátil y es el que se ha utilizado mayoritariamente en el proyecto, a continuación nos centraremos en la descripción de este paquete, dejando de lado el AWT.

El paquete Swing es el nuevo paquete gráfico que ha aparecido en la versión 1.2 de Java (Java 2). Está compuesto por un amplio conjunto de componentes de interfaces de usuario que funcionan en el mayor número posible de plataformas.

Cada uno de los componentes de este paquete puede presentar diversos aspectos y comportamientos en función de una biblioteca de clases. En la versión 1.0 de Swing, que corresponde a la distribuida en la versión 1.2 de la API de Java se incluyen tres bibliotecas de aspecto y comportamiento para Swing:

- `metal.jar`: Aspecto y comportamiento independiente de la plataforma.
- `motif.jar`: Basado en la interfaz Sun Motif.
- `windows.jar`: Muy similar a las interfaces Microsoft Windows 95 (sólo disponible en Windows).

Es la nueva clase denominada `UiManager` la que se encarga del aspecto y comportamiento de una aplicación Swing en un entorno de ejecución.

La arquitectura Swing presenta una serie de ventajas respecto a su antecedente AWT:

- Amplia variedad de componentes: En general las clases que comiencen por "J" son componentes que se pueden añadir a la aplicación. Por ejemplo: `JButton` (el componente equivalente en AWT era `Button`).
- Aspecto modificable (*Look And Feel*): Se puede personalizar el aspecto de las interfaces o utilizar varios aspectos que existen por defecto (*Metal Max*, *Basic Motif*, *Windows Win32*).
- Arquitectura Modelo-Vista-Controlador: Esta arquitectura da lugar a todo un enfoque de desarrollo muy arraigado en los entornos gráficos de usuario realizados con técnicas orientadas a objetos. Cada componente tiene asociado una clase de modelo de datos y una interfaz que utiliza. Se puede crear un modelo de datos personalizado para cada componente, con sólo heredar de la clase `Model`.
- Gestión mejorada de la entrada del usuario. Se pueden gestionar combinaciones de teclas en un objeto `KeyStroke` y registrarlo como componente. El evento se activará cuando se pulse dicha combinación si está siendo utilizado el componente, la ventana en que se encuentra o algún hijo del componente.
- Objetos de acción (*Action Objects*): Estos objetos, cuando están activados (*enabled*), controlan las acciones de varios objetos componentes de la interfaz. Son hijos de `ActionListener`.
- Contenedores anidados: Cualquier componente puede estar anidado en otro. Por ejemplo, un gráfico se puede anidar en una lista.
- Escritorios virtuales: Se pueden crear escritorios virtuales o "interfaz de múltiples documentos" mediante las clases `JDesktopPane` y `JInternalFrame`.
- Bordes complejos: Los componentes pueden presentar nuevos tipos de bordes. Además el usuario puede crear tipos de bordes personalizados.
- Diálogos personalizados: Se pueden crear multitud de formas de mensajes y opciones de diálogo con el usuario, mediante la clase `JOptionPane`.
- Clases para diálogos habituales: Se puede utilizar `JFileChooser` para elegir un fichero, y `JColorChooser` para elegir un color.
- Componentes para tablas y árboles de datos: Mediante las clases `JTable` y `JTree`.
- Potentes manipuladores de texto: Además de campos y áreas de texto, se presentan campos de sintaxis oculta `JPassword`, y texto con múltiples fuentes `JTextPane`. Además hay paquetes para utilizar ficheros en formato HTML o RTF.

2.6 La API de comunicaciones de Java

Java no proporciona en el núcleo de su JDK clases para el acceso a los puertos de comunicaciones serie y paralelo. Aún así, Java ha desarrollado una serie de clases para tal fin, incluidas en la API de comunicaciones de Java [J3], siendo una extensión estándar de Java. El paquete que provee estas clases de acceso es el llamado `javax.comm`.

Tal y como se describe en su página web, la API de comunicaciones de Java puede ser usada para escribir aplicaciones independientes de la plataforma donde se ejecute para tecnologías como correo de voz, fax y tarjetas inteligentes (*smart-cards*). Actualmente existen dos versiones de esta API de comunicaciones: la 2.0.3 para Solaris/SPARC y la 2.0 para Windows y Solaris/x86. El soporte para Linux/Unix no está soportado implícitamente, pero como veremos en el capítulo 3, es posible hacer funcionar el programa en Linux gracias a las librerías RXTX. La API de comunicaciones de Java da soporte para puertos serie RS-232 y puertos paralelo IEEE 1284.

El paquete de extensiones `javax.comm` contiene tres niveles de clases en la API de comunicaciones de Java:

- Clases de alto nivel, como `CommPortIdentifier` y `CommPort`, que manejan el acceso y posesión de los puertos de comunicaciones.
- Clases de bajo nivel, como `SerialPort` y `ParallelPort`, que proveen una interfaz para la comunicación física con los puertos.
- Clases a nivel de controlador (*driver*), que proveen una interfaz entre las clases de bajo nivel y el sistema operativo subyacente. Estas clases son parte de la implementación, pero no de la API de comunicaciones de Java. No deben ser usadas por los programadores de la aplicación.

Así mismo, el paquete `javax.comm` provee los siguientes servicios básicos:

- Enumerar los puertos disponibles en el sistema. El método estático `CommPortIdentifier.getPortIdentifiers()` devuelve una enumeración con un objeto `CommPortIdentifier` correspondiente a cada puerto disponible en el sistema. El objeto `CommPortIdentifier` es el mecanismo central para controlar el acceso a los puertos de comunicaciones.
- Abrir y reclamar la posesión de los puertos de comunicaciones usando los métodos de alto nivel de los objetos `CommPortIdentifier`.
- Resolver el enfrentamiento entre posesiones de puertos entre varias aplicaciones de Java. Los eventos son propagados para notificar el enfrentamiento de las aplicaciones interesadas en la posesión del puerto, y permite a las aplicaciones que actualmente tengan la posesión del puerto renunciar a él. La excepción `PortInUseException` es lanzada cuando una aplicación no consigue abrir el puerto.
- Ejecutar operaciones de entrada y salida sincrónicas y asíncronas sobre los puertos de comunicaciones. Las clases de bajo nivel como `SerialPort` y `ParallelPort` contienen métodos para manejar la entrada y salida de datos en los puertos.
- Recibir eventos que describan los cambios de estado de los puertos. Por ejemplo, cuando un puerto serie ha cambiado de estado en la propiedad de indicador de llamada, un objeto de la clase `SerialPortEvent` se propaga describiendo el cambio de estado.

2.6.1 Interfaces del paquete `javax.comm`

2.6.1.1 CommDriver

Es parte de la interfaz de controladores de dispositivos que se pueden cargar. Esta clase no debe ser usada por programas a nivel de aplicación. Contiene los siguientes métodos:

- `public abstract CommPort getCommPort(String portName, int portType)`. Este método será llamado por el método `open()` de la clase `CommPortIdentifier`. El parámetro portName es una cadena que fue registrada previamente usando el método `CommPortIdentifier.addPortName()`. `getCommPort()` devuelve un objeto que extiende bien de `SerialPort` o bien de `ParallelPort`.
- `public abstract void initialize()`. Este método será llamado por el inicializador estático de la clase `CommPortIdentifier`. La responsabilidad de este método es:
  - Asegurar que el hardware está presente.
  - Cargar cualquier librería nativa que sea requerida.
  - Registrar los nombres de los puertos con `CommPortIdentifier`.

2.6.1.2 CommPortOwnershipListener

Propaga varios eventos de posesión de los puertos de comunicaciones. Cuando un puerto se abreo, un evento del tipo `PORT_OWNED` se propaga. Cuando un puerto se cierra, un evento de tipo `PORT_UNOWNED` se propaga. La contención de posesiones de puertos entre varias aplicaciones puede ser resuelta del siguiente modo:

1. La aplicación A llama al método `open()` y consigue la posesión de un puerto.
2. La aplicación B, más tarde, llama también al método `open()` sobre el mismo puerto.
3. Mientras se procesa la apertura del puerto de la aplicación B, la clase `CommPortIdentifier` propagará un evento de `CommPortOwnership` con un evento de tipo `PORT_OWNERSHIP_REQUESTED`.
4. Si la aplicación A se ha registrado para escuchar los eventos del puerto, y quiere que la otra aplicación pueda usar el puerto, llamará al método `close()`.
5. Una vez que el evento ha sido propagado, `CommPortdentifier` comprueba si la posesión del puerto ha sido cedida por la aplicación A, otorgando la posesión del puerto a la aplicación B.

Si una aplicación cierra un puerto tras una solicitud de otra aplicación para abrir el puerto, no se genera ningún evento.

Las variables presentes en esta clase son:

- `PORT_OWNED`. El puerto ha pasado de un estado sin propietario a uno con propietario, cuando una aplicación lo ha obtenido con éxito mediante el método `CommPortIdentifier.open()`.
- `PORT_OWNERSHIP_REQUESTED`. Enfrentamiento de posesión.
- `PORT_UNOWNED`. El puerto ha pasado de un estado con propietario a uno sin propietario, cuando una aplicación ha llamado al método `CommPort.close()`.

Los métodos que ofrece esta interfaz son los siguientes:

- `public abstract void ownershipChange(int type)`. Propaga un evento `CommPortOwnership`. Este método será llamado con uno de estos valores: `PORT_OWNED`, `PORT_UNOWNED`, o `PORT_OWNERSHIP_REQUESTED`.

2.6.1.3 ParallelPortEventListener

Propaga los eventos del puerto paralelo. Posee un solo método:

- `public abstract void parallelEvent(ParallelPortEvent ev)`. Propaga un evento `ParallelPortEvent`.

2.6.1.4 SerialPortEventListener

Propaga los eventos del puerto serie. Posee un solo método:

- `public abstract void serialEvent(SerialPortEvent ev)`. Propaga un evento `SerialPortEvent`.

2.6.2 Clases del paquete javax.comm

2.6.2.1 CommPort

Identifica un puerto de comunicaciones. `CommPort` es una clase abstracta que describe un puerto disponible por el sistema operativo subyacente. Incluye métodos de alto nivel para controlar operaciones de entrada salida comunes a diferentes tipos de puertos de entrada y salida. `SerialPort` y `ParallelPort` son subclases de `CommPort` que incluyen métodos adicionales para el control a bajo nivel de los puertos físicos de comunicaciones.

Esta clase no tiene constructores públicos. En lugar de esto, una aplicación debe usar el método estático `CommPortIdentifier.getPortIdentifiers()` para generar una lista de puertos disponibles. Cuando se elija un puerto de esta lista, se llamará al método `CommPortIdentifier.open()` para generar un objeto `CommPort`. Finalmente se puede hacer un *casting* del objeto `CommPort` a un dispositivo físico de comunicaciones como `SerialPort` o `ParallelPort`.

Después de que un puerto haya sido identificado y abierto, puede ser configurado con los métodos de bajo nivel de las clases `SerialPort` y `ParallelPort`. Así, un flujo de entrada/salida puede ser abierto para leer y escribir información. Una vez que la aplicación ha terminado de utilizar el puerto, debe llamar al método `close()`. Más tarde, la aplicación no debe llamar más a los métodos del puerto. Si se hace, se lanzará una excepción `IllegalStateException`.

Esta clase contiene una sola variable de instancia:

- `protected String name`.

Además provee estos métodos:

- `public String getName()`. Obtiene el nombre del Puerto de comunicaciones. El nombre debe corresponderse con algo que el usuario pueda identificar, como la etiqueta de hardware. Devuelve el nombre del puerto.
- `public String toString()`. Devuelve una representación textual de este puerto de comunicaciones.
- `public abstract InputStream getInputStream() throws IOException`. Devuelve el flujo de entrada asociado al puerto. Ésta es la única forma de recibir datos desde el Puerto. Si el puerto es unidireccional y no soporta la recepción de datos, entonces el método devuelve *null*.

- `public abstract OutputStream getOutputStream() throws IOException`. Devuelve el flujo de salida asociado al puerto. Es la única forma de enviar datos al puerto. Si el puerto es unidireccional y no soporta el envío de datos, entonces este método devuelve *null*.
- `public void close()`. Cierra el puerto de comunicaciones. La aplicación debe llamar a este método cuando haya terminado de trabajar con el puerto. Se propagará una notificación de cambio del propietario a todas las clases registradas usando el método `addPortOwnershipListener()`.
- `public abstract void enableReceiveThreshold(int thresh) throws UnsupportedCommOperationException`. Activa el umbral de recepción, si esta característica está soportada por el controlador. Cuando la condición del umbral de recepción es verdadera, una lectura del flujo de entrada retorna inmediatamente. Este método tan solo es consultivo, y el controlador puede o no implementarlo. Por defecto, el umbral de recepción está deshabilitado.
- `public abstract void disableReceiveThreshold()`. Desactiva el umbral de recepción.
- `public abstract boolean isReceiveThresholdEnabled()`. Comprueba si el umbral de recepción está activado.
- `public abstract int getReceiveThreshold()`. Obtiene el valor entero del umbral de recepción. Si el umbral de recepción no está soportado por el controlador, entonces el valor devuelto no tiene significado ninguno.
- `public abstract void enableReceiveTimeout(int rcvTimeout) throws UnsupportedCommOperationException`. Activa el intervalo de recepción, si esta característica está soportada por el controlador. Cuando la condición del intervalo de recepción es verdadera, el método de lectura del flujo de entrada para el puerto retorna inmediatamente. Este método es tan solo consultivo, y el controlador puede o no implementarlo. Por defecto, el intervalo de recepción está desactivado.
- `public abstract void disableReceiveTimeout()`. Desactiva el intervalo de recepción.
- `public abstract boolean isReceiveTimeoutEnabled()`. Comprueba si el intervalo de recepción está activado.
- `public abstract int getReceiveTimeout()`. Obtiene el valor entero del intervalo de recepción. Si el umbral de recepción no está soportado por el controlador, entonces el valor devuelto no tiene significado ninguno.
- `public abstract void enableReceiveFraming(int framingByte) throws UnsupportedCommOperationException`. Activa el encuadre de recepción, si esta característica está soportada por el controlador. Cuando la condición de encuadre de recepción es verdadera, el método de lectura del flujo de entrada retorna inmediatamente. Este método es tan solo consultivo y puede o no ser implementado por el controlador. Por defecto el encuadre de recepción está desactivado.
- `public abstract void disableReceiveFraming()`. Desactiva el encuadre de recepción.
- `public abstract boolean isReceiveFramingEnabled()`. Comprueba si el encuadre de recepción está activado.
- `public abstract int getReceiveFramingByte()`. Devuelve el *byte* actual usado para el encuadre de recepción. Si el encuadre de recepción está desactivado o no está soportado por el controlador, entonces el valor devuelto está carente de significado. El valor devuelto por este método es un entero, del cual los

8 bits de menor peso representan el *byte* que actualmente se usa para el encuadre de recepción.

- `public abstract void setInputBufferSize(int size)`. Establece el tamaño del espacio de memoria de entrada (*buffer*). Este método es tan solo consultivo y la disponibilidad de memoria viene impuesta por el tamaño del *buffer* usado por el controlador.
- `public abstract int getInputBufferSize()`. Devuelve el tamaño del *buffer* de entrada. Este método es tan solo consultivo y el sistema operativo subyacente puede elegir el no informar los valores correctos para el tamaño del *buffer*.
- `public abstract void setOutputBufferSize(int size)`. Establece el tamaño del espacio de memoria de salida (*buffer*). Este método es tan solo consultivo y la disponibilidad de memoria viene impuesta por el tamaño del *buffer* usado por el controlador.
- `public abstract int getOutputBufferSize()`. Devuelve el tamaño del *buffer* de salida. Este método es tan solo consultivo y el sistema operativo subyacente puede elegir el no informar los valores correctos para el tamaño del *buffer*.

2.6.2.2 CommPortIdentifier

Es la clase central para controlar el acceso a los puertos de comunicaciones. Incluye métodos para:

- Determinar los puertos disponibles por el controlador.
- Abrir puertos de comunicaciones para operaciones de entrada y salida.
- Determinar la posesión de puertos.
- Resolver conflictos de posesión de puertos.
- Manejar eventos que indiquen cambios en la posesión de los puertos.

Una aplicación primero usará la clase `CommPortIdentifier` para obtener los puertos disponibles en el sistema, para después abrirlo. Tras ello, usará otras clases como `CommPort`, `SerialPort` o `ParallelPort` para comunicarse con el puerto.

Posee dos variables de instancia:

- `public static final int PORT_SERIAL`. Indica que es un puerto RS-232.
- `public static final int PORT_PARALLEL`. Indica que es un puerto paralelo IEE 1284.

Además, ofrece los siguientes métodos:

- `public static Enumeration getPortIdentifiers()`. Devuelve un objeto de tipo `Enumeration` que contiene una serie de objetos `CommPortIdentifier` para cada uno de los puertos del sistema.
- `public static CommPortIdentifier getPortIdentifier(String portName) throws NoSuchPortException`. Obtiene un objeto de clase `CommPortidentifier` usando un nombre de puerto. El nombre del puerto debe haber sido almacenado anteriormente por la aplicación.
- `public static CommPortIdentifier getPortIdentifier(CommPort port) throws NoSuchPortException`. Obtiene un objeto de la clase `CommPortIdentifier` correspondiente a un puerto que ya ha sido abierto por la aplicación.

- `public static void addPortName(String portName, int portType, CommDriver driver)`. Añade un puerto a la lista de puertos.
- `public String getName()`. Devuelve el nombre del puerto. El nombre del puerto debe ser identificable por el usuario. Por ejemplo "COM1" en Windows, "Serial A" en estaciones de trabajo Sun Ultra o "/dev/ttyS0" en Linux. El nombre del puerto puede ser guardado por una aplicación y posteriormente usado para crear un `CommPortIdentifier` usando el método `getPortIdentifier(String portname)`.
- `public int getPortType()`. Devuelve el tipo de puerto (serie o paralelo).
- `public synchronized CommPort open(String appname, int timeout) throws PortInUseException`. Abre el puerto de comunicaciones. Este método obtiene la posesión exclusive del puerto. Si el puerto ya ha sido abierto por otra aplicación, un evento de tipo `PORT_OWNERSHIP_REQUESTED` es propagado usando el mecanismo de eventos de la clase `CommPortOwnershipListener`. Si la aplicación que posee el puerto lo cierra durante el procesamiento de los eventos, entonces `open()` tendrá éxito. Hay un `InputStream` (flujo de entrada) y un `OutputStream` (flujo de salida) asociado a cada puerto. Después de que un puerto haya sido abierto, todas las llamadas a `getInputStream()` devolverán el mismo objeto hasta que se cierre el puerto.
- `public String getCurrentOwner()`. Devuelve el propietario actual del puerto.
- `public boolean isCurrentlyOwned()`. Comprueba si el puerto está actualmente en uso.
- `public void addPortOwnershipListener(CommPortOwnershipListener listener)`. Registra a una aplicación para que sea notificada de los cambios en la propiedad del puerto.
- `public void removePortOwnershipListener(CommPortOwnershipListener lsnr)`. Elimina un `CommPortOwnershipListener` registrado anteriormente usando `addPortOwnershipListener()`.
- `public CommPort open(FileDescriptor fd) throws UnsupportedCommOperationException`. Abre el puerto de comunicaciones usando un objeto de la clase `FileDescriptor` sobre plataformas que soporten esta técnica.

2.6.2.3 ParallelPort

Describe la interfaz de bajo nivel para las comunicaciones por puerto paralelo disponibles por el sistema operativo subyacente. Esta clase define la funcionalidad mínima requerida para las comunicaciones por puerto paralelo.

Posee las siguientes variables de instancia:

- `LPT_MODE_ANY`. Elige el mejor modo disponible.
- `LPT_MODE_ECP`. Puerto con capacidades enriquecidas.
- `LPT_MODE_EPP`. Puerto paralelo extendido.
- `LPT_MODE_NIBBLE`. Modo *Nibble* (medio byte). Bidireccional.
- `LPT_MODE_PS2`. Modo byte. Bidireccional.

- `LPT_MODE_SPP`. Modo de compatibilidad. Unidireccional.

Posee un único constructor:

- `public ParallelPort()`. Constructor por defecto.

Los métodos implementados por esta clase son los siguientes:

- `public abstract void addEventListener(ParallelPortEventListener lsnr) throws TooManyListenersException`. Registra un objeto de la clase `ParallelPortEventListener` para que escuche eventos del puerto paralelo (`ParallelPortEvent`). El interés en eventos específicos puede ser expresado mediante los métodos `notifyOnError()` y `notifyOnBuffer()`. Sólo está soportado un escuchador por puerto. Si se llama varias veces a este método, simplemente se sustituirá el escuchador anterior por el nuevo. Después de que un puerto haya sido cerrado, no se generarán más eventos. Si se vuelve a abrir el puerto, se deberá registrar de nuevo el escuchador para recibir eventos.
- `public abstract void removeEventListener()`. Elimina un escuchador que ya haya sido añadido anteriormente con el método `addEventListener()`. Esto es ejecutado automáticamente cuando el puerto es cerrado.
- `public abstract void notifyOnError(boolean notify)`. Activa o desactiva la notificación de eventos ante errores.
- `public abstract void notifyOnBuffer(boolean notify)`. Activa o desactiva la notificación de eventos cuando el *buffer* de salida esté vacío.
- `public abstract int getOutputBufferFree()`. Devuelve el número de *bytes* disponibles en el *buffer* de salida.
- `public abstract boolean isPaperOut()`. Comprueba si el puerto está indicando un estado "*Out of Paper*".
- `public abstract boolean isPrinterBusy()`. Comprueba si el puerto está indicando un estado "*Printer Busy*".
- `public abstract boolean isPrinterSelected()`. Comprueba si la impresora está en estado seleccionado.
- `public abstract boolean isPrinterTimedOut()`. Comprueba si el tiempo de espera de la impresora ha expirado.
- `public abstract boolean isPrinterError()`. Comprueba si la impresora ha encontrado un error.
- `public abstract void restart()`. Reinicia la salida tras un error.
- `public abstract void suspend()`. Prorroga la salida.
- `public abstract int getMode()`. Obtiene el modo configurado actualmente.
- `public abstract int setMode(int mode) throws UnsupportedCommOperationException`. Establece el modo del puerto de la impresora.

2.6.2.4 ParallelPortEvent

Eventos del puerto paralelo.

Esta clase posee las siguientes variables de instancia:

- `public int eventType`. Desfasado. Reemplazado por el método `getEventType()`. Se conserva para compatibilidad con la versión 1.0 exclusivamente,

- `public static final int PAR_EV_ERROR`. Error ocurrido en el puerto.
- `public static final int PAR_EV_BUFFER`. El *buffer* de salida del puerto está vacío.

Posee un único constructor:

- `public ParallelPortEvent(ParallelPort srcport, int eventtype, boolean oldvalue, boolean newvalue)`. Construye un nuevo objeto `ParallelPortEvent` con el puerto paralelo especificado, tipo de evento y los valores antiguo y el actual. Las aplicaciones no deben crear directamente objetos de esta clase.

Y tres métodos:

- `public int getEventType()`. Devuelve el tipo de evento.
- `public boolean getNewValue()`. Devuelve el nuevo valor del cambio de estado que causó que el evento sea propagado.
- `public boolean getOldValue()`. Obtiene el valor antiguo del cambio de estado que causó que el evento sea propagado.

2.6.2.5 SerialPort

Describe la interfaz de bajo nivel para comunicaciones con el puerto serie disponibles por el sistema operativo subyacente. `SerialPort` define la funcionalidad mínima requerida para las comunicaciones con el puerto serie.

Posee las siguientes variables de instancia:

- `DATABITS_5`. 5 bits de datos.
- `DATABITS_6`. 6 bits de datos.
- `DATABITS_7`. 7 bits de datos.
- `DATABITS_8`. 8 bits de datos.
- `FLOWCONTROL_NONE`. Sin control de flujo.
- `FLOWCONTROL_RTSCTS_IN`. Control de flujo hardware de entrada.
- `FLOWCONTROL_RTSCTS_OUT`. Control de flujo hardware de salida.
- `FLOWCONTROL_XONXOFF_IN`. Control de flujo software de entrada.
- `FLOWCONTROL_XONXOFF_OUT`. Control de flujo software de salida.
- `PARITY_EVEN`. Paridad par.
- `PARITY_MARK`. Paridad por marca.
- `PARITY_NONE`. Sin paridad.
- `PARITY_ODD`. Paridad impar.
- `PARITY_SPACE`. Paridad por espacio.
- `STOPBITS_1`. 1 bit de parada.
- `STOPBITS_1_5`. 1 bit y medio de parada.
- `STOPBITS_2`. 2 bits de parada.

Posee un único constructor:

- `public SerialPort()`. Constructor por defecto.

Los métodos implementados son los siguientes:

- `public abstract int getBaudRate()`. Obtiene la velocidad configurada actualmente del puerto.
- `public abstract int getDataBits()`. Devuelve el número de bits de datos configurado actualmente.
- `public abstract int getStopBits()`. Devuelve el número de bits de parada.
- `public abstract int getParity()`. Devuelve la opción de paridad configurada.
- `public abstract void sendBreak(int millis)`. Envía una parada de la cantidad especificada como parámetro de milisegundos de duración. Esta opción puede que no sea posible en ciertos sistemas operativos.
- `public abstract void setFlowControlMode(int flowcontrol) throws UnsupportedCommOperationException`. Establece el control de flujo.
- `public abstract int getFlowControlMode()`. Devuelve el control de flujo configurado actualmente.
- `public void setRcvFifoTrigger(int trigger)`. Método en desuso. Establece el nivel de la señal de recepción FIFO.
- `public abstract void setSerialPortParams(int baudrate, int dataBits, int stopBits, int parity) throws UnsupportedCommOperationException`. Establece los parámetros del puerto serie.
- `public abstract void setDTR(boolean dtr)`. Activa o desactiva en la UART el bit DTR (*Data Terminal Ready*, Terminal de Datos Prepearada).
- `public abstract boolean isDTR()`. Obtiene el dato del bit DTR en la UART.
- `public abstract void setRTS(boolean rts)`. Activa o desactiva en la UART el bit RTS (*Request To Send*, Petición Para Enviar).
- `public abstract boolean isRTS()`. Obtiene el estado del bit RTS de la UART.
- `public abstract boolean isCTS()`. Obtiene el estado del bit CTS (*Clear To Send*, Preparado Para Enviar) de la UART.
- `public abstract boolean isDSR()`. Obtiene el estado del bit DSR (*Data Set Ready*, Datos Preparados) de la UART.
- `public abstract boolean isRI()`. Obtiene el estado del bit RI (*Ring Indicador*, Indicador de llamada) de la UART.
- `public abstract boolean isCD()`. Obtiene el estado del bit CD (*Carrier Detect*, Portadora Detectada) de la UART.
- `public abstract void addEventListener(SerialPortEventListener lsnr) throws TooManyListenersException`. Registra un `SerialPortEventListener` para escuchar eventos `SerialEvent`. Si se está interesado en ciertos eventos se pueden usar los métodos `notifyOnXXX()`. El método `serialEvent` de `SerialPortEventListener` será llamado con un objeto `SerialEvent` describiendo el evento. Sólo un escuchador por puerto serie está soportado. Si se llama más de una vez a este método, se sobrescribirá el escuchador anterior por el actual. Todo los eventos generados por este escuchador son generados por un hilo (*Thread*) dedicado que pertenece al objeto `SerialPort`. Cuando se cierra el puerto, no se generan más eventos. Si se vuelve a abrir el puerto, se deberá añadir de nuevo el escuchador si se quieren recibir eventos.

- `public abstract void removeEventListener()`. Elimina un escuchador previamente registrado con el método `addEventListener()`. Esto se hace automáticamente al cerrar el puerto.
- `public abstract void notifyOnDataAvailable(boolean enable)`. Indica el interés específico en que se informe o no sobre los eventos generados cuando haya datos de entrada disponibles. El evento sólo se generará cuando nuevos datos lleguen al puerto.
- `public abstract void notifyOnOutputEmpty(boolean enable)`. Indica el interés específico en que se informe o no sobre los eventos generados cuando el *buffer* de salida se vacíe.
- `public abstract void notifyOnCTS(boolean enable)`. Indica el interés específico en que se informe o no sobre los eventos generados cuando cambia el bit CTS.
- `public abstract void notifyOnDSR(boolean enable)`. Indica el interés específico en que se informe o no sobre los eventos generados cuando cambia el bit DSR.
- `public abstract void notifyOnRingIndicator(boolean enable)`. Indica el interés específico en que se informe o no sobre los eventos generados cuando cambia el bit RI.
- `public abstract void notifyOnCarrierDetect(boolean enable)`. Indica el interés específico en que se informe o no sobre los eventos generados cuando cambia el bit CD.
- `public abstract void notifyOnOverrunError(boolean enable)`. Indica el interés específico en que se informe o no sobre los eventos generados cuando se produce un error de desbordamiento.
- `public abstract void notifyOnParityError(boolean enable)`. Indica el interés específico en que se informe o no sobre los eventos generados cuando se produce un error de paridad.
- `public abstract void notifyOnFramingError(boolean enable)`. Indica el interés específico en que se informe o no sobre los eventos generados cuando produce un error de encuadre.
- `public abstract void notifyOnBreakInterrupt(boolean enable)`. Indica el interés específico en que se informe o no sobre los eventos generados cuando se produce una interrupción de parada en la línea.

2.6.2.6 SerialPortEvent

Eventos del puerto serie. Esta clase posee las siguientes variables de instancia:

- `public int eventType`. En desuso. Reemplazado mediante el método `getEventType()`. Tipo de evento.
- `public static final int DATA_AVAILABLE`. Datos disponibles en el puerto serie. Este evento será generado una vez que llegue nuevos datos al puerto serie. Incluso si el usuario no lee los datos, no se generará un nuevo evento hasta que lleguen nuevos datos.
- `public static final int OUTPUT_BUFFER_EMPTY`. *Buffer* de salida vacío. Este evento será generado después de que una escritura sea completa, cuando el *buffer* del sistema quede vacío de nuevo.
- `public static final int CTS`. Preparado para enviar.
- `public static final int DSR`. Datos preparados.

- `public static final int RI`. Indicador de ring.
- `public static final int CD`. Portadora detectada.
- `public static final int OE`. Error de desbordamiento.
- `public static final int PE`. Error de paridad.
- `public static final int FE`. Error de encuadre.
- `public static final int BI`. Interrupción por parada.

Posee un único constructor:

- `public SerialPortEvent(SerialPort srcport, int eventtype, boolean oldvalue, boolean newvalue)`. Construye un Nuevo objeto SerialPortEvent con los parámetros especificados.

Y tres métodos:

- `public int getEventType()`. Devuelve el tipo del evento.
- `public boolean getNewValue()`. Obtiene el nuevo valor del estado que causó que el evento `SerialPortEvent` fuera generado.
- `public boolean getOldValue()`.Obtiene el antiguo valor del estado que causó que el evento `SerialPortEvent` fuera generado.

2.6.3 Excepciones del paquete javax.comm

2.6.3.1 NoSuchPortException

Lanzada cuando el controlador no puede hallar un puerto específico.

2.6.3.2 PortInUseException

Lanzada cuando el puerto especificado está en uso. Posee una variable de instancia:

- `public String currentOwner`. Corresponde al propietario actual del puerto de comunicaciones.

2.6.3.3 UnsupportedCommOperationException

Lanzada cuando un controlador no permite la operación especificada. Posee dos constructores:

- `public UnsupportedCommOperationException(String str)`. Construye una nueva excepción de este tipo con el mensaje especificado como detalle.
- `public UnsupportedCommOperationException()`. Construye una nueva excepción de este tipo sin ningún mensaje de explicación.

Capítulo 3
Conocimientos previos II: Servicios diferenciados en Cisco

3.1 Introducción

La segunda parte del proyecto que se pasa a explicar es la referente a los aspectos del *router* utilizado. Se han utilizado dos *routers* Cisco modelo 2620 y 2611 ambos pertenecientes a la serie 2600. En la segunda parte de este capítulo se dará una breve introducción a la empresa que los fabrica; a continuación se describirán los aspectos técnicos, ventajas y características de esta serie de *routers* para pasar después a tratar el sistema operativo propio que gestiona estos dispositivos: el Cisco Internetworking Operating System (Cisco IOS). Por último, en el quinto epígrafe se introducirán conceptos básicos sobre los servicios diferenciados y calidad de servicio (QoS) y las utilidades que poseen los *routers* Cisco y el software Cisco IOS para aplicar estas tecnologías.

3.2 La empresa

Cisco Systems, Inc. [C1] es la compañía líder mundial en tecnología de redes para Internet. Hoy, las redes son una parte esencial de los negocios, la educación el gobierno y las comunicaciones en el hogar, y las soluciones de red de Cisco basadas en el protocolo IP (*Internet Protocol*) son la base de estas redes. El *hardware*, el *software* y los servicios que Cisco ofrece se usan para crear soluciones en Internet que permitan a individuales, compañías y países incrementar su productividad, lograr la satisfacción del consumidor y alcanzar una ventaja competitiva. El nombre de Cisco ha llegado a ser sinónimo de Internet, así como las mejoras en la productividad y las soluciones para negocios en Internet que provee.

Figura 4 - Logo de Cisco

Cisco fue fundado en 1984 por un pequeño grupo de investigadores informáticos de la Universidad de Stanford. Desde el nacimiento de la compañía, los ingenieros de Cisco han sido líderes en el desarrollo de tecnologías de red basadas en IP. Esta tradición de innovación en IP continúa con productos líderes en el sector en áreas básicas como el enrutamiento y la conmutación, así como en tecnologías avanzadas como:

- Redes caseras
- Telefonía sobre IP
- Redes ópticas
- Seguridad de redes
- Almacenamiento en redes
- Redes de área local inalámbricas

Cisco está comprometido en entregar un valor diferenciado a consumidores y socios a través de sus ofertas en productos y servicios, mientras se acerca a las necesidades y prioridades de los usuarios, como la productividad, bajos costes, investigación,…

Además de productos *hardware* y *software*, Cisco dispone de un amplio rango de servicios para sus clientes, incluyendo soporte técnico y servicios avanzados. Cisco vende sus servicios y productos directamente a través de su red de ventas e indirectamente a través de una red de grandes empresas en alianzas, pequeñas y medianas compañías, proveedores de servicios y consumidores. Cisco enfoca el crecimiento de su negocio en tres grandes áreas: tecnologías básicas, enrutamiento y conmutación; mercado de proveedores de servicios; y mercados de tecnología avanzada.

3.3 Los *routers* Cisco de la serie 2600

La serie Cisco 2600 [C2], con una amplia base instalada, ofrece una solución rentable para satisfacer las necesidades actuales y futuras de las empresas en lo referente a:

- Integración multiservicio de voz y datos.
- Acceso a redes privadas virtuales (*Virtual Private Network*, VPN) con opciones de cortafuegos.
- Servicios de acceso telefónico analógico y digital.
- Enrutamiento con gestión de ancho de banda.
- Enrutamiento entre VLAN (*Virtual Local Area Network*, Redes de área local virtuales).

La arquitectura modular de la serie Cisco 2600 permite actualizar las interfaces para ajustarlas a la expansión de la red o a los cambios tecnológicos que se producen cuando se instalan nuevos servicios y aplicaciones. Al compartir las interfaces modulares con las series Cisco 1600, 1700 y 3600, la serie Cisco 2600 proporciona una protección de la inversión inigualable. Mediante la integración de las funciones de varios dispositivos independientes en una sola unidad compacta, la serie Cisco 2600 reduce la complejidad de gestionar la solución para redes remotas.

Figura 5 - Varios modelos de routers de la serie 2600

Los *routers* Cisco de la serie 2600 poseen un potente procesador RISC y DSP de alto rendimiento y procesadores auxiliares en varias interfaces. Admite calidad de servicio (*Quality of Service*, QoS) avanzada, seguridad y características de integración en redes.

La serie Cisco 2600 está disponible en tres niveles de rendimiento y seis configuraciones base:

- Cisco 2650 y Cisco 2651: hasta 37.000 paquetes por segundo (pps), uno y dos puertos *Ethernet* 10/100 Mbps con autodetección.
- Cisco 2620 y Cisco 2621: hasta 25.000 pps, uno y dos puertos *Ethernet* 10/100 Mbps con autodetección.

- Cisco 2610 a Cisco 2613: hasta 15.000 pps.
    - Cisco 2613: un puerto *Token Ring*.
    - Cisco 2612: un puerto *Ethernet*, un puerto *Token Ring*.
    - Cisco 2611: dos puertos *Ethernet*.
    - Cisco 2610: un puerto *Ethernet*.

Todos los modelos también disponen de dos ranuras para tarjetas de interfaz WAN (WIC), una ranura para el módulo de red y una ranura para un módulo de integración avanzada (AIM). Estas ranuras comparten más de cincuenta módulos distintos entre cuatro líneas de productos de Cisco.

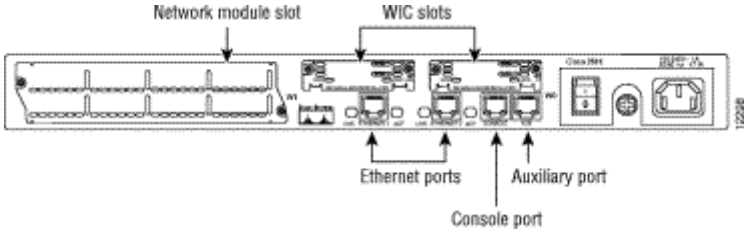


Figura 6 - Detalle de la parte trasera de los routers

Las tarjetas de interfaz WAN disponibles para los *routers* Cisco 1600, 1700, 2600 y 3600 admiten una amplia gama de opciones seriales, RDSI de acceso básico y de unidad de servicio de canal y unidad de servicio de datos (CSU/DSU) para conexión WAN principal y de respaldo. Los módulos de red disponibles para las series Cisco 2600 y 3600 admiten una amplia gama de aplicaciones, incluyendo la integración multiservicio voz/datos, acceso telefónico analógico y RDSI, así como concentración de dispositivos serie. El módulo de integración avanzada de compresión de datos de la serie Cisco 2600 descarga a la CPU principal de las tareas de compresión de datos de alta velocidad, permitiendo que el tráfico de datos comprimidos alcance los 8 Mbps mientras libera las ranuras de interfaces externas para otras aplicaciones.

3.3.1 Ventajas principales

Al formar parte de las soluciones de red Cisco de extremo a extremo, la serie Cisco 2600 permite a las empresas ampliar la infraestructura de red de forma transparente y rentable y les ofrece las siguientes ventajas:

- Protección de la inversión. Ya que la serie Cisco 2600 admite componentes modulares actualizables en la instalación, los clientes pueden cambiar con facilidad las interfaces de red sin tener que realizar una "actualización integral" de la solución implementada en la red de la oficina. Otra forma en que la ranura AIM de la plataforma Cisco 2600 protege la inversión económica es ofreciendo la capacidad de expansión necesaria para admitir servicios avanzados, tales como compresión y cifrado de datos asistidos por hardware.
- Costo de operación reducido. Mediante la integración de las funciones de las CSU/DSU, dispositivos de terminación de red RDSI (NTI), módems, cortafuegos, dispositivos de compresión o cifrado y demás equipamiento de los recintos de cableado de la empresa en una sola unidad, la serie Cisco 2600 ofrece una solución que ahorra espacio y que puede gestionarse remotamente usando aplicaciones de administración de redes tales como CiscoWorks y CiscoView.
- Integración multiservicio voz/datos. La serie Cisco 2600 permite a los administradores de redes reducir los costos de las llamadas de larga distancia entre oficinas y permite utilizar aplicaciones de nueva generación, como la mensajería integrada y centros de llamadas basados en Web. Utilizando los módulos de voz/fax, el *router* Cisco 2600 puede instalarse en redes de voz a través de IP (*Voice over IP*, VoIP) y voz a través de Frame Relay (*Voice over Frame Relay*, VoFR). Si se utilizan con el nuevo módulo de red troncal de voz por paquetes, un solo dispositivo de la

serie Cisco 2600 puede admitir 60 llamadas simultáneas de voz, además de enrutamiento y otros servicios.

- Solución de clase empresarial y proveedor. Cumple los requisitos que exigen las empresas de multiservicio y sus proveedores de servicios, con características de alta fiabilidad, conexiones WAN múltiples y la posibilidad de migrar de una infraestructura de sólo datos a una de TDM voz y datos.

3.3.2 Características

3.3.2.1 Versatilidad y protección de la inversión

Las tarjetas de interfaz WAN y los módulos de red son los mismos que los de los *routers* de las series Cisco 1600, 1700 y 3600

- Reducción del costo de mantenimiento de inventario para los componentes modulares de las series Cisco 1600, 1700, 2600 y 3600.
- Reducción de los costos de capacitación para el personal de soporte técnico.

Compatibilidad con tarjeta de interfaz voz/WAN Multiflex

- Puede utilizarse para la conexión de WAN (sólo datos) y, a continuación, volver a implementarse para admitir voz y datos canalizados o aplicaciones de paquetes de voz.

Ranura de módulo de integración avanzada

- Capacidad de ampliación para la integración de servicios avanzados de alto rendimiento tales como la compresión o cifrado de datos asistidos por hardware.

Opción de fuente de alimentación CC

- Permite la instalación en entornos de alimentación CC tales como oficinas centrales de portadoras de telecomunicaciones.

3.3.2.2 Simplificación de la gestión

Compatible con CiscoWorks y CiscoView

- Simplifica la gestión de todos los componentes apilados e integrados .

Compatible con Cisco Voice Manager (CVM)

- Reduce el costo de la instalación y gestión de soluciones de voz/datos integrados.

Característica de configuración mejorada

- Las preguntas de ayuda interactiva guían al usuario a través del proceso de configuración del *router* y permiten instalarlo más rápidamente.

Compatibilidad con Cisco AutoInstall

- Configura automáticamente *routers* remotos a través de una conexión WAN para evitar la necesidad de enviar personal técnico a la instalación remota.

Partes de las soluciones empresariales apilables de Cisco

- Para simplificar su gestión puede utilizarse conjuntamente con *switches* LAN, como por ejemplo los modelos Catalyst® 1900 ó 2820XL.

Compatibilidad con VLAN

- Permite el enrutamiento entre VLAN a través del protocolo ISL (*Inter-Switch Link*) y 802.1Q (Cisco 2621 con un conjunto de características "Plus" de IOS) de Cisco.

3.4 El software Cisco IOS

El software Cisco IOS (*Internetworking Operating System*, Sistema operativo para trabajo en red) es el software de redes líder de la industria y más extensamente desplegado. Proporciona servicios de red inteligentes en una infraestructura flexible de interconexión que permite un despliegue rápido de las aplicaciones de Internet.

Cisco IOS está en continua expansión de sus capacidades y usos. Las necesidades del cliente dirigen las capacidades del Cisco IOS, que proporciona un extenso rango de funcionalidades: desde conectividad básica, seguridad y gestión de redes hasta servicios avanzados tales como comercio en tiempo real, soporte interactivo, etc.

La funcionalidad del Cisco IOS es el resultado de una evolución. La primera generación de dispositivos de interconexión sólo podían almacenar y encaminar paquetes de datos. Actualmente, Cisco IOS software puede reconocer, clasificar y priorizar el trafico de la red, optimizar el enrutamiento, soportar aplicaciones de video y de voz y muchas cosas más.

Cisco IOS software se ejecuta en la mayoría de *routers* de Cisco e incrementalmente en los *switches* de Cisco. Estos dispositivos de red manejan la mayor parte del tráfico de Internet actualmente.

La última versión del software Cisco IOS disponible en la web del fabricante es la 12.3.

3.4.1 Modos de operación de Cisco IOS

La interfaz del sistema operativo Cisco IOS se divide en varios modos de configuración posibles [C3]. Pulsando la tecla "?" en cada modo, podremos obtener ayuda acerca de los comandos disponibles en ese modo o ayuda acerca del comando que estemos usando (opciones, parámetros…).

Cuando se conecta al software Cisco IOS se comienza con el modo usuario, denominado modo EXEC. En este modo se pueden realizar operaciones tales como:

- Pruebas de interconectividad (*ping* y *traceroute* básicos).
- Ver la configuración actual del *router* (interfaces, mapas de clases, políticas, etc.).
- Acceder como un usuario determinado al *router*.
- Acceder mediante *telnet* a otro equipo o *router*.

Para tener acceso a todos los comandos, se debe entrar en el modo administrador o EXEC privilegiado. Desde este modo, se puede entrar en el modo de configuración global, donde tendremos acceso a las diferentes opciones de configuración del *router*: interfaces, enrutamiento, mapas de clases, políticas, protocolos, etc.

Si nos encontramos en modo EXEC y queremos pasar al modo EXEC privilegiado, utilizaremos el comando `enable`. Una vez introducido el comando, se nos solicitará la contraseña del nuevo modo, ofreciendo tres intentos.

Si por el contrario queremos pasar del modo EXEC privilegiado al modo EXEC, primero regresaremos al prompt principal del modo EXEC privilegiado, en caso de que no lo estemos ya, mediante la combinación de teclas Control+Z. Tras ello escribiremos `disable`, y pulsaremos la tecla Intro, por lo que pasaremos al modo EXEC.

Cuando estemos en el modo EXEC privilegiado y hagamos cambios en la configuración del *router*, estos cambios se almacenan en memoria, en la llamada *running-config*, que desaparece una vez que se apague o reinicie el *router*. Para hacer que la configuración modificada quede almacenada permanentemente, se debe copiar la configuración actual a la configuración de inicio (*startup-config*) mediante el comando *`copy running-config startup-config`*, desde el modo EXEC privilegiado.

3.4.2 Filtrado de la salida de los comandos

A partir de la versión 12.0 del Cisco IOS, se pueden hacer búsquedas y filtrados sobre las salidas de ciertos comandos del *router*, al estilo `grep` en Linux, mediante los modificadores *begin*, *include* y *exclude*. El formato del comando para utilizar los filtros sería el siguiente:

- `comando | {begin | include | exclude} expresión`

Un ejemplo de esta aplicación sería el siguiente comando:

- `MiRouter> show class-map | include Class Map`

Este comando filtraría la salida del comando `show class-map`, devolviendo en la salida sólo aquellas líneas que contuvieran la cadena "Class Map". Si por el contrario se hubiera aplicado el modificador *exclude*, se habrían mostrado todas las líneas de salida del comando, excepto aquellas que contuvieran la cadena anterior. Con el modificador *begin*, se habría mostrado la salida completa del comando a partir de la primera ocurrencia de la cadena.

3.4.3 Negar y usar las opciones por defecto de un comando

La mayoría de los comandos del sistema operativo Cisco IOS permiten usar las opciones *no* y *default* delante de un comando para deshabilitar y establecer las opciones por defecto de un cierto comando, respectivamente.

Por ejemplo, si escribiésemos este comando:

- `MiRouter(config-line)#default length`

haría que el número de líneas mostradas por pantalla en una conexión al *router* a través del puerto serie se estableciera a la opción por defecto configurada en el *router*.

Por el contrario, el siguiente comando:

- `MiRouter(config)#no class-map clase_ejemplo`

haría que se eliminara la clase *clase_ejemplo*. En otros casos como el siguiente:

- `MiRouter(config)#no ip routing`

haría que se deshabilitará el enrutamiento de paquetes.

3.4.4 Uso de las configuraciones en ejecución y de inicio

Como ya vimos anteriormente, conforme se van haciendo cambios en la configuración del *router*, estos cambios no se guardan en la configuración por defecto del *router*, sino que se guardan en la memoria temporal, que se borra cada vez que se apaga o reinicia el *router*.

Para guardar la configuración que actualmente se está aplicando, incluyendo los cambios que se hayan hecho durante la sesión, se ejecutará el siguiente comando desde el modo EXEC privilegiado:

- `MiRouter# copy running-config startup-config`

Por el contrario, si queremos volver a la última configuración guardada mediante el comando anterior, descartando los cambios que hayamos hecho, usaremos el comando siguiente, también desde el modo EXEC privilegiado:

- `MiRouter# copy startup-config running-config`

Para hacer un reinicio del *router*, usaremos el comando `reload`. Si hemos efectuado cambios en la configuración, se nos preguntará si los queremos salvar o no, solicitando después una confirmación de reinicio.

- `MiRouter# reload`

3.4.5 Configuración básica de interfaces

Como se vio en el apartado 3.3, los *routers* Cisco de la serie 2600 pueden tener interfaces seriales, *Ethernet*, *Token-Ring*, *FastEthernet*, etc. Con el siguiente comando podemos comprobar todas las interfaces que posee nuestro *router*, así como ver las características, configuración y estado de cada una de ellas:

- `MiRouter> show interfaces`

Para poder cambiar la configuración de una cierta interfaz, accederemos al modo EXEC privilegiado y pasaremos al modo de configuración del terminal. Una vez allí seleccionaremos la interfaz que queremos configurar mediante un comando similar al siguiente:

- `MiRouter(config)#interface Ethernet 0/0`

Una vez allí tendremos la opción de activar o desactivar la interfaz con el prefijo no al comando `shutdown`. Es decir, para activar la interfaz ejecutaremos:

- `MiRouter(config-if)#no shutdown`

Y para desactivarla utilizaremos el comando:

- `MiRouter(config-if)#shutdown`

Para configurar la dirección IP de la interfaz tenemos dos posibilidades. La primera de ellas es configurarla con una dirección IP estática. Usaremos, por tanto, el siguiente comando:

- `MiRouter(config-if)#ip address 192.168.1.50 255.255.255.0 [secondary]`

El parámetro secondary es opcional e indica que la dirección indicada pase a ser una dirección secundaria para la interfaz.

La otra opción es hacer que la interfaz obtenga una dirección dinámica mediante el protocolo DHCP. Para ello, usaremos el siguiente comando:

- `MiRouter(config-if)#ip address dhcp [client-id { interfaz_ethernet | vlan <1-1001>}]`

Mediante esta opción haremos que la interfaz se configure mediante el protocolo DHCP. Podemos indicar la interfaz *Ethernet* que se utilizará para contactar con el servidor DHCP o la VLAN que servirá para el mismo propósito.

3.4.6 Habilitar el enrutamiento RIP

Para habilitar el enrutamiento RIP en nuestro *router* se utilizarán básicamente tres comandos. El primero de ellos activa el enrutamiento IP:

- `MiRouter(config)#ip routing`

A continuación le indicaremos que el protocolo que queremos que se utilice para el enrutamiento será el RIP (*Routing Information Protocol*, Protocolo de información de enrutamiento):

- `MiRouter(config)#router rip`

Tras ello, y una vez dentro de la configuración de RIP, debemos indicar todas las redes a las que está conectado el *router*. Por ejemplo:

- `MiRouter(config-router)#network 192.168.1.0`

- `MiRouter(config-router)#network 192.168.2.0`

Una vez hecho esto, podremos guardar la configuración actual del *router* en la memoria mediante el comando `copy running-config startup-config`.

3.4.7 Cambiar el nombre al *router*

En este apartado explicaremos cómo se puede cambiar el nombre al *router*, es decir, cambiar el prompt de la línea de comandos del software Cisco IOS. El comando que se utilizará para tal fin es `hostname`, dentro de la configuración de terminal:

- `MiRouter(config)#hostname MiRouter`

3.4.8 Cambiar la contraseña del modo EXEC privilegiado

Cuando queramos pasar del modo EXEC al modo EXEC privilegiado, el sistema operativo del *router* nos solicitará la contraseña correspondiente. Mediante el siguiente comando es posible cambiar dicha contraseña. Estando en el modo EXEC privilegiado, y en la configuración de terminal, el comando a utilizar es:

- `MiRouter(config)#enable secret nueva_contraseña`

Como ya se dijo anteriormente, para conservar la nueva contraseña una vez que se reinicie el *router* deberemos copiar la configuración actual a la configuración de inicio.

3.5 Herramientas de configuración del *router*

Aparte de la herramienta que se desarrollará en este proyecto, en el mercado existen otras aplicaciones cuyo fin es también la configuración de este tipo de dispositivos. La mayoría de estas aplicaciones son realizadas por el propio fabricante de los routers, Cisco.

A continuación se citan algunas de estas herramientas, indicando una breve referencia a cada una de ellas.

3.5.1 Cisco ConfigMaker

Mediante esta herramienta [C14] es posible diseñar redes completas, arrastrando y soltando dispositivos tales como *routers*, conmutadores, repetidores, etc. y enlazando todos ellos a través de líneas *Ethernet*, *Frame Relay*, seriales, fibra óptica, enlaces RTC ó RDSI, etc. La herramienta generará las imágenes de la configuración en un archivo, que se podrán enviar a los routers bien a través del puerto serie o a través de la propia red si esta ya está configurada.

El programa sólo tiene capacidad para asignar direcciones a las interfaces y generar las tablas de enrutamiento de acuerdo con la red dibujada, sin posibilidad de introducir opciones más avanzadas tales como servicios diferenciados.

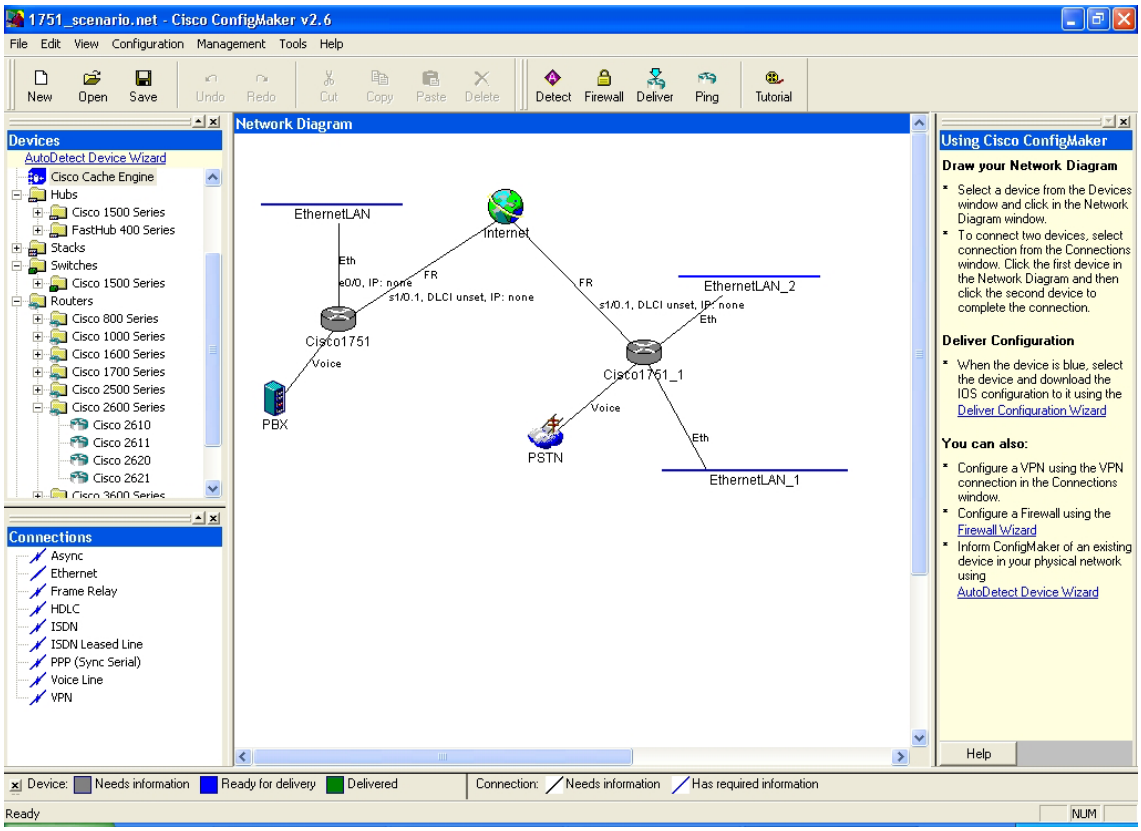


Figura 7 - Cisco ConfigMaker

3.5.2 CiscoWorks CiscoView

Esta herramienta [C13] permite mostrar el estado de las interfaces del *router* a través de una página web. La pantalla simula la parte trasera del dispositivo indicando con distintos colores las interfaces que funcionan correctamente y aquellas que tienen algún problema.

Figura 8 - CiscoWorks CiscoView

3.5.3 Interfaz de configuración a través de Web

El software Cisco IOS posee internamente un servidor web mediante el cual se pueden configurar aspectos básicos del *router*. El acceso a tal interfaz web se puede restringir de varias formas: mediante listas de acceso, a través de la contraseña EXEC privilegiado, etc. Para las capturas de pantalla siguientes se realizó la configuración para utilizar la contraseña de administración, de tal forma que ante la pantalla de autentificación se debe poner cualquier nombre de usuario y como contraseña la correspondiente al modo EXEC privilegiado:

Figura 9 - Pantalla de autentificación

Tras la autenticación, se mostrará la pantalla inicial desde la cual se puede acceder a distintas secciones como: mostrar información sobre las interfaces, mostrar un diagnóstico del *router*, realizar una prueba de conectividad, obtener información necesaria para solicitar soporte técnico, etc.

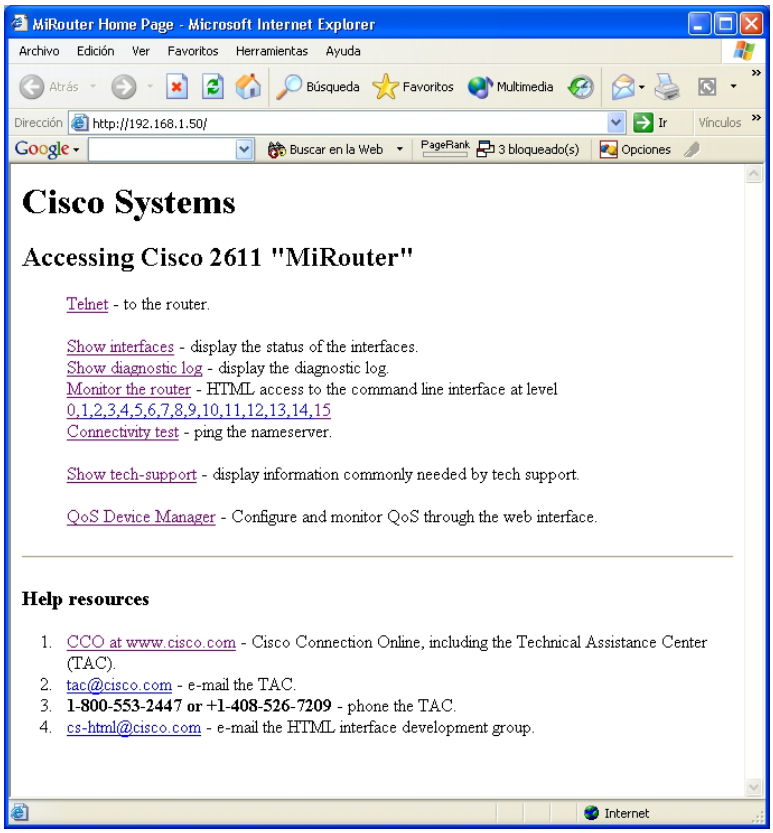


Figura 10 - Configuración a través de web mediante Cisco IOS

3.6 Herramientas del IOS para Servicios Diferenciados

En este apartado se verán los principales mecanismos que proporcionan los *routers* Cisco y su software IOS para implementar los servicios diferenciados que se verán en el siguiente apartado [C4].

3.6.1 Modular QoS CLI

EL Modular QoS CLI [C5] proporciona una estructura de interfaz de comandos para configurar las diferentes herramientas de calidad basadas en clases (*QoS class-based*). Así, MQC engloba al resto de herramientas de que dispone Cisco para proporcionar los servicios diferenciados de una forma sencilla.

Permite la separación entre la clasificación de paquetes (*class-maps*), las políticas (*policy-maps*) aplicadas en las clases definidas y asociar esas políticas a las interfaces correspondientes (*service-policy*). El MQC forma la base para proporcionar servicios diferenciados y todos los mecanismos de QoS son parte de los *class-maps* (mapas de clases, clasificación), o los *policy-maps* (mapas de políticas; dentro de los *policy-maps* se incluyen funciones de policía, espaciado, encolado, técnicas para evitar la congestión, marcado de paquetes, marcado de los bits de Clase de Servicio CoS (*Class of Service*) de la capa de enlace de datos, etc.).

Como se ha expuesto anteriormente, los tres pasos que se seguirían con el Modular QoS CLI para poder implementar los servicios diferenciados serían:

1. Definir una clase de tráfico (comando *class-map*).

2. Crear un mapa de políticas asociando a la clase creada anteriormente una o varias políticas de calidad de servicio (comando *policy-map*).
3. Asociar el mapa de políticas recién creado a una interfaz (*service-policy*).

En los siguientes apartados veremos como desarrollar cada uno de los tres apartados mencionados.

3.6.2 Definir una clase de tráfico

Mediante una clase de tráfico, se clasifican los diferentes flujos de tráfico cuando estos acceden al *router*. Para llevar esto a cabo, se utiliza el comando *class-map*. Mediante las clases de tráfico se pueden separar los paquetes que llegan al *router* para aplicarles un tratamiento diferenciado.

Para crear un nuevo mapa de clases de tráfico utilizaremos el comando `class-map` del siguiente modo, estando en la configuración de terminal del modo EXEC privilegiado:

- *Router*`(config)#class-map [ match-any | match-all ]` *nombre_clase*

Con este comando crearemos una nueva clase de tráfico con el nombre especificado. Si además especificamos la opción `match-any` o `match-all` haremos que para que un paquete pase a pertenecer a dicha clase deberá cumplir cualquiera o todas, respectivamente, las condiciones de la clase, que veremos a continuación. La opción por defecto es que se deben cumplir todas las condiciones para que un paquete pase a formar parte de la clase.

A continuación deberemos especificar los criterios que deben cumplir los paquetes para que formen parte de esa clase. Para ello utilizaremos el comando `match`. Según la configuración de la clase (`match-any` o `match-all`) se irán comprobando una a una las condiciones de la clase y si se cumple que cualquiera de las condiciones (`match-any`) o todas las condiciones (`match-all`) son ciertas, el paquete analizado pasará a formar parte de la clase. Las opciones disponibles para el comando match son las siguientes:

- `Match access-group grupo_acceso`. El paquete deberá cumplir la condición de la lista de acceso que se especifica.
- `Match any`. Cualquier paquete cumplirá esta condición.
- `Match class-map nombre_clase`. Hace posible el uso de clases anidadas. El paquete cumplirá esta condición si forma parte de la clase especificada.
- `Match cos <0-7>`... Permite especificar hasta cuatro valores basados en el marcado de Clase de Servicio de la capa de enlace de datos.
- `Match destination-address direccion_MAC`. Permite incluir como condición para el paquete que la dirección MAC de destino sea la especificada.
- `Match input-interface interfaz_entrada`. Permite incluir como condición que el paquete haya entrado por la interfaz indicada.
- `Match ip dscp valor_ip_dscp`... Posibilita introducir hasta 8 valores de IP DSCP como criterio de selección.
- `Match ip precedence <0-7>`... Permite introducir hasta 4 valores correspondientes al *IP Precedence*.
- `Match ip rtp puerto_inicio numero_puertos`. Permite especificar el puerto de inicio y el número de puertos usados por el Protocolo de Tiempo Real (Real Time Protocol, RTP).
- `Match mpls experimental <0-7>`... Especifica hasta 8 valores posibles de MPLS como criterio de selección para que el paquete pueda pasar a formar parte de la clase.

- `Match protocol protocolo`. Permite especificar que el paquete pertenezca a un determinado protocolo.
- `Match qos-group numero_grupo_qos`. Permite especificar un valor específico de QoS como criterio de selección para el paquete.
- `Match source-addrees direccion_MAC`. Permite incluir como condición para el paquete que la dirección MAC de origen sea la especificada.

Además, a todas estas opciones se les puede incluir un prefijo `not`, para indicar que los paquetes no deben cumplir esa condición para que puedan pasar a formar parte de la clase. Por ejemplo, el comando:

- `Match protocol smtp`

Haría que un paquete SMTP pudiera pasar a formar parte de la clase en cuestión. Sin embargo, el comando:

- `Match not protocol smtp`

Haría que todos los paquetes pudieran pasar a formar parte de la clase excepto aquellos que pertenezcan al protocolo SMTP.

Por último, si el paquete no cumple las condiciones para formar parte de ninguna clase configurada, pasará a formar parte de la clase por defecto, *class-default*.

3.6.3 Definir un mapa de políticas

El siguiente paso tras definir un mapa de clases en el MQC es crear las políticas que se aplicaran a los paquetes que pertenezcan a determinadas clases. El comando usado para crear los mapas de políticas el `policy-map`. Mediante estas políticas se podrán aplicar las diferentes herramientas de servicios diferenciados de Cisco como CBWFQ, WRED, *Class-Based Packet Marking*, *Class-Based Policing*, etc. a los paquetes de cada una de las clases creadas con los comandos del apartado anterior.

Para crear un nuevo mapa de políticas usaremos el comando de la siguiente forma:

- `MiRouter(config)#policy-map nombre_mapa_politicas`

A continuación especificaremos la clase a la cual vamos a aplicar las políticas que más tarde definiremos. En un mismo mapa de políticas es posible tener más de una clase; esto puede ser útil si queremos agrupar bajo un mismo mapa de políticas paquetes que sigan reglas parecidas pero que las características no sean comunes, sino que haya varios grupos en los que se pueden englobar. Para especificar la clase a la cual queremos aplicar las políticas usaremos el comando `class` de la siguiente forma:

- `MiRouter(config-pmap)#class nombre_clase`

Una vez dentro de la configuración de la clase, podremos aplicar las políticas oportunas. Para ello, disponemos de las siguientes opciones:

- `Bandwith { <8-2000000> | percent <1-100> }`. Permite especificar un ancho de banda asegurado.
- `Priority <8-2000000> [<32-2000000>]`. Configura las opciones para *Strict Scheduling Priority*.
- `Queue-limit <1-512>`. Permite especificar el valor de la cola.
- `Random-detect [ dscp valor_dscp | dscp-based | exponential-weighting-constant <1-16> | prec-based | precedence { <0-7> | rvsp } ]`. Activa la detección temprana aleatoria como política de borrado.
- `Service-policy nombre_politica`. Establece otro mapa de políticas como política para esta clase.

- `Shape { average <8000-154400000> [<256-154400000> [<0-154400000>]] | max-buffers <1-4096> | peak <8000-154400000> [<256-154400000> [<0-154400000]]}`. Configura las opciones para *Traffic Shaping*.
- `Police <8000-200000000> [<1000-51200000> [<1000-51200000>]] [conform-action {drop | set-clp-transmit | set-dscp-transmit <0-63> | set-prec-transmit <0-7> | set-qos-transmit <0-99> | transmit} [exceed-action {drop | set-clp-transmit | set-dscp-transmit <0-63> | set-prec-transmit <0-7> | set-qos-transmit <0-99> | transmit} [violate-action {drop | set-clp-transmit | set-dscp-transmit <0-63> | set-prec-transmit <0-7> | set-qos-transmit <0-99> | transmit}]]]`. Opciones de política. Permite establecer unos valores de velocidades y aplicarles ciertas opciones según su comportamiento.
- `Set {atm-clp | cos <0-7> | ip dscp valor_dscp | ip precedence <0-7> | mpls experimental <0-7> | qos-group <0-99>}`. Permite establecer valores de calidad de servicio (QoS).

Podemos usar el prefijo `no` delante de cualquiera de los comandos anteriores para eliminar una política concreta de la clase que se esté configurando.

Aparte de configurar políticas a las clases que configuremos, también podemos aplicar políticas a la clase por defecto (*default-class*); estas políticas se aplicarán a todos aquellos paquetes que no pertenezcan a ninguna clase en concreto definida por nosotros.

3.6.4 Asociación de mapas de políticas a interfaces

El último paso a la hora de implementar servicios diferenciados mediante el *Modular Quality Of Services Command Line Interface* es asociar los mapas de políticas creados anteriormente a las interfaces. De esta forma, las políticas se aplicarán cuando entren o salgan por una interfaz determinada del *router*. Para conseguir esto usaremos el comando `service-policy` dentro de la configuración de la interfaz:

- `MiRouter(config-if)#service-policy input mapa_politicas`
- `MiRouter(config-if)#service-policy output mapa_politicas`

Los dos parámetros que requiere este comando son el nombre del mapa de políticas que se quiere usar para esa interfaz, y el tipo de aplicación, es decir, si se quiere que la política se aplique cuando un paquete entre por esa interfaz o cuando un paquete salga por esa interfaz. De esta forma, cada interfaz puede tener asociadas dos políticas distintas o iguales: una para la entrada y otra para la salida.

3.7 Aplicación de comandos para Servicios Diferenciados

En este apartado se explicará como se pueden configurar los distintos mecanismos de servicios diferenciados disponibles en la actualidad mediante el *Modular QoS CLI* del software Cisco IOS.

3.7.1 Class-Based Packet Marking

Class-Based Packet Marking [C7] proporciona un modo fácil mediante CLI para un marcado eficiente de los paquetes: marca los paquetes de una determinada clase definida por

el usuario. Se puede activar cuando se configura la política para una clase. Esta herramienta permite a los usuarios ejecutar las siguientes acciones:

- Marcar los paquetes poniendo los bits del *IP Precedence* o el *IP DSCP* en el *byte* ToS de la cabecera IP.
- Marcar los paquetes poniendo el valor de la Clase de Servico (*Class of Service*, CoS) de la capa 2.
- Asociar un valor de grupo local de calidad de servicio a un paquete.
- Poner los bits de la prioridad de pérdida de celda (*Cell Loss Priority*, CLP) en la cabecera ATM de un paquete de 0 a 1.
- Poner el bit de *Frame Relay Discard Eligibility* (DE) en el campo de dirección del marco *Frame Relay* de 0 a 1.

En la mayoría de los casos el propósito del marcado de los paquetes es la identificación. Después de que un paquete se marque, los dispositivos del flujo de bajada identifican el tráfico basándose en ese marcado y lo tratan de acuerdo a las necesidades de la red. Esto ocurre cuando los comandos match dentro de las clases de tráfico están configurados para identificar los paquetes marcados.

El marcado de paquetes permite dividir la red en varios niveles o clases de servicio. *Class-Based Packet Marking* se usa frecuentemente para poner los valores de *IP Precedence* o de *IP DSCP* a los paquetes que entran en la red, Los elementos de red pueden usar esos valores marcados en los paquetes para determinar el tratamiento que se les aplicará al tráfico. Por ejemplo, el tráfico de voz puede ser marcado con un valor determinado (*IP Precedence* o *IP DSCP*) y la herramienta de encolamiento de baja latencia (*Low Latency Queueing*, LLQ), puede estar configurada para poner los paquetes marcados en una cola prioritaria (*priority queue*).

También usaremos *Class Based Packet Marking* para asignar paquetes a un grupo de QoS dentro del *router*. El valor del grupo de QoS normalmente es usado por una de las siguientes razones.

- Para aumentar el número de clases de tráfico. El grupo de QoS tiene 100 marcados diferentes de paquetes individuales, frente a *IP DSCP* e *IP Precedence* con 64 y 8 valores respectivamente.
- Si no queremos cambiar el valor de *IP Precedence* e *IP DSCP* del paquete.

Esta herramienta de Cisco también permite hacer un mapeo de la capa 2 a la capa 3. Por ejemplo, si un paquete que necesita ser marcado a un servicio de QoS diferenciado, está abandonando el *router* y entra en un *switch*, el *router* puede poner el valor de CoS (*Class of Service*) del paquete ya que el *switch* puede procesar el marcado de CoS de la cabecera de la capa 2.

Algo muy importante a tener en cuenta es que el *Class Based Packet Marking* puede marcar sólo paquetes que circulen a través de caminos de conmutación de la herramienta de encaminamiento urgente de Cisco (*Cisco Express Forwarding*, CEF). Por lo tanto, para que funcione, CEF deberá estar configurado tanto en las interfaces que envían como el las que reciben paquetes.

Class Based Packet Marking se puede configurar sobre una interfaz, subinterfaz o sobre un circuito virtual permanente (*Permanent Virtual Circuit*, PVC) ATM, pero no se soporta en las siguientes interfaces:

- *Fast EtherChannel*
- Túnel
- *Primary Rate Interface* (PRI) de la Red Digital de Servicios Integrados (RDSI)
- *ATM Switched Virtual Circuit* (SVC)
- *Frame Relay Data Link Connection Identifier* (DLCI)
- Alguna interfaz que no soporte CEF

También deberemos tener en cuenta que una política que contiene el comando `set qos-group` se puede asociar únicamente como política de entrada. No se emplea para paquetes que abandonen el *router*. Del mismo modo, una política que contiene el comando `set cos` se puede asociar sólo como política de salida.

Los pasos para activar *Class Based Packet Marking* son los siguientes:

- `MiRouter(config)#policy-map mapa_politicas`
- `MiRouter(config-pmap)#class mapa_clases`

Una vez aquí podremos configurar el campo del paquete que queramos marcar:

- `MiRouter(config-pmap-c)#set ip precedence 0`
- `MiRouter(config-pmap-c)#set ip dscp 0`
- `MiRouter(config-pmap-c)#set qos-group 0`

3.7.2 Traffic Policing

Traffic Policing [C9] permite limitar la tasa de transmisión de una clase de tráfico. Para ello se basa en criterios definidos por el usuario. También permite al sistema marcar los paquetes con el *IP DSCP* o el *IP Precedence*. Por ejemplo: cuando un paquete sobrepasa la tasa contratada hay que marcarlo con un DSCP diferente que en el caso de que el paquete cumpla la tasa contratada. Esto permitirá a la red deshacerse de este tipo de paquetes en caso de congestión. Notar que en el caso anterior (*Class-Based Packet Marking*) los paquetes se marcaban si pertenecían a una clase determinada definida por el usuario.

Traffic Policing permite controlar la tasa máxima transmitida o recibida sobre una interfaz. De este modo se podrá controlar el ancho de banda del enlace. *Traffic Policing* se configura frecuentemente sobre interfaces en los extremos de la red para limitar el tráfico que entra o sale de ella. En la mayoría de las configuraciones de *Traffic Policing*, el tráfico que cae dentro de los parámetros acordados es transmitido, mientras que el que excede es descartado o transmitido con una prioridad diferente.

Traffic Policing puede realizar un marcado o remarcado de los paquetes. Como se veía en *Class-Based Packet Marking*, el marcado de paquetes permite dividir la red en múltiples niveles de prioridad o clases de servicio:

- Podemos usar *Traffic Policing* para poner los valores del *IP DSCP* o del *IP Precedence* a los paquetes que entran a la red.
- También para asignar los paquetes a un grupo QoS. El *router* usará estos grupos QoS para dar prioridad a los paquetes dentro del mismo *router*.

De nuevo, *Traffic Policing* puede monitorizar sólo caminos de conmutación del *Cisco Express Forwarding* (CEF). Por lo tanto, para que funcione, CEF deberá estar configurado tanto en las interfaces que envían como en las que reciben paquetes. *Traffic Policing* se puede configurar sobre una interfaz, subinterfaz, o un circuito virtual permanente (PVC) ATM. Pero las siguientes interfaces no soportan el *Traffic Policing*:

- *Fast EtherChannel*
- Túnel
- *Primary Rate Interface* (PRI) de la Red Digital de Servicios Integrados (RDSI)
- Alguna interfaz que no soporte CEF
- Sobre la serie de Cisco *router* 7500, *Traffic Policing* no puede aplicarse a paquetes que se originan desde o van destinados a un *router*.

Class-Based Policing trabaja con un mecanismo de *token bucket*. Hay dos tipos de algoritmos de *token bucket*: un algoritmo de un solo *token bucket* y un algoritmo de doble *token bucket*. Cisco puede implementar estos dos algoritmos. Cisco IOS implementa un sistema de

un *token bucket* cuando la opción `violate-action` no se especifica, y un sistema de doble *token bucket* cuando la opción `violate-action` si se especifica.

El siguiente ejemplo muestra una posibilidad de configuración de *Traffic Policing* con todos los campos de la opción `police`, aunque sólo es obligatorio el primer campo (tasa en bits por segundo):

- `MiRouter(config)#policy-map mapa_politicas`
- `MiRouter(config-pmap)#class mapa_clases`
- `MiRouter(config-pmap-c)#police 8000 1000 2000 conform-action transmit exceed-action set-dscp-transmit 0 violate-action drop`

La forma general del comando `police` es la siguiente:

- `police bps burst_normal burst_max conform-action` acción `exceed-action` acción `violate-action` acción

Donde:

- *Bps* es la tasa media en bits por segundo.
- *Burst_normal* es el tamaño de la ráfaga normal en *bytes* (tamaño del cubo).
- *Burst_max* es el tamaño de la ráfaga máxima en *bytes*. Sólo es necesario cuando se especifica la opción `violate-action`.
- *Conform-action* especifica la acción que se aplicará sobre los paquetes que estén conformes con la tasa límite.
- *Exceed-action* especifica la acción que se aplicará sobre los paquetes que excedan la tasa límite.
- *Violate-action* especifica la acción que se aplicará sobre los paquetes que violan el tamaño de ráfaga normal y máximo. Si la opción `violate-action` está especificada, el algoritmo *token bucket* trabaja con un algoritmo de doble *token bucket*.
- *Acción* puede ser una de las siguientes:
  - *Transmit*. El paquete se transmite sin ser alterado.
  - *Set-clp-transmit*. Pone el bit ATM CLP de 0 a 1 sobre la celda ATM y transmite el paquete con el bit ATM CLP a 1.
  - *Set-dscp-transmit*. Pone el valor de *IP DSCP* que se especifique a continuación y transmite el paquete con el nuevo valor.
  - *Set-prec-transmit*. Pone el valor de *IP Precedence* que se especifique a continuación y transmite el paquete con el nuevo valor.
  - *Set-qos-transmit*. Pone el valor del grupo QoS que se especifique a continuación y transmite el paquete con el nuevo valor.
  - *Drop*. Elimina el paquete.

3.7.3 Traffic Shaping

Traffic Shaping [C8] permite controlar el tráfico que abandona una interfaz para casar su flujo con la velocidad de la interfaz remota, y asegurar así que el tráfico cumpla las políticas contratadas para él. Esto permite eliminar los cuellos de botella en las topologías. Tra*ffic Shaping* dispone de un mecanismo de *token bucket* y de memorias para almacenar paquetes. Cuando llega una ráfaga de tráfico la almacena y la sirve a una tasa constante con lo que suaviza las crestas de tráfico producidas por estas ráfagas, espaciando los paquetes que le llegan en el tiempo.

Las principales razones para usar *Traffic Shaping* son:

- Controlar el acceso al ancho de banda disponible.
- Asegurar que el tráfico cumple las políticas establecidas para él.
- Evitar la congestión que puede ocurrir cuando enviamos un exceso de tráfico a una interfaz remota.

Traffic Shaping previene la pérdida de paquetes. Su uso es especialmente importante en redes *Frame Relay* ya que el *switch* no puede determinar el orden de los paquetes y por lo tanto los paquetes son tirados cuando hay congestión. Más aún, es de vital importancia para tráfico en tiempo real tal como voz sobre *Frame Relay*. Retener los datos en el *router* le permite a éste priorizar el tráfico. *Traffic Shaping* limita la tasa de transmisión de datos. Es posible limitar la tasa de datos a una de las siguientes opciones:

- Una tasa específica configurada.
- Una tasa derivada del nivel de congestión.

La tasa de transferencia depende de tres componentes que constituyen el *token bucket*: tamaño de ráfaga (*burst size*), tasa media y el intervalo de medida (tiempo). La tasa media es igual al tamaño de ráfaga dividido por el intervalo.

Una variable adicional se le aplica al *Traffic Shaping*: *Be size* (el tamaño de ráfaga de exceso, *the exceed burst size*). El *Be size* permite estar enviando más tráfico que el del tamaño normal de ráfaga durante un intervalo de tiempo en ciertas situaciones. Se permitirá el paso de los paquetes que provengan del *Excess Burst* pero serán marcados.

Traffic Shaping suaviza el tráfico almacenando tráfico a la tasa configurada en una cola. Cuando un paquete llega a la interfaz para transmitirse, ocurre lo siguiente:

1. Si la cola está vacía, el paquete es procesado por el *Traffic Shaper*:
    a. Si es posible, el *Traffic Shaper* envía el paquete.
    b. En otro caso, el paquete se coloca en la cola.
2. Si la cola no está vacía, el paquete se coloca en la cola.

Cuando los paquetes están en la cola, el *Traffic Shaper* borra el número de paquetes que puede enviar de la cola cada intervalo de tiempo.

El comando empleado dentro de la configuración de una clase de un mapa de políticas para usar este tipo de servicio diferenciado es el `shape`. A continuación se muestra un ejemplo de uso de este comando:

- `MiRouter(config)#policy-map mapa_politicas`
- `MiRouter(config-pmap)#class mapa_clases`
- `MiRouter(config-pmap-c)#shape average 8000 256 0`

  En esta primera opción se envían solo *Bc* bits en cada intervalo. Los parámetros son los siguientes:

  - 8000. Indica la tasa bits en bits por segundo. Debe ser múltiplo de 8000 desde 8000 hasta 154400000.
  - 256. Bits por intervalo sostenido. Debe ser un múltiplo de 128 desde 256 hasta 154400000. No se recomienda usar este valor, ya que el algoritmo hallará el mejor valor.
  - 0. Bits por intervalo en exceso. Debe ser un múltiplo de 128 desde 0 hasta 154400000. El valor por defecto es 0.

- `MiRouter(config-pmap-c)#shape max-buffers 1`

  El único parámetro de esta opción indica el tamaño máximo del *buffer*.

- `MiRouter(config-pmap-c)#shape peak 8000 256 0`

  En esta primera opción se envían solo *Bc*+*Be* bits en cada intervalo. Los parámetros son los siguientes:

- 8000. Indica la tasa bits en bits por segundo. Debe ser múltiplo de 8000 desde 8000 hasta 154400000.
- 256. Bits por intervalo sostenido. Debe ser un múltiplo de 128 desde 256 hasta 154400000. No se recomienda usar este valor, ya que el algoritmo hallará el mejor valor.
- 0. Bits por intervalo en exceso. Debe ser un múltiplo de 128 desde 0 hasta 154400000. El valor por defecto es 0.

3.7.4 Class Based Weighted Fair Queueing (CBWFQ)

Class-Based Weighted Fair Queueing [C6] [C10] es un mecanismo usado para proporcionar un ancho de banda mínimo garantizado a las diferentes clases de tráfico durante periodos de congestión. CBWFQ extiende la funcionalidad estándar de *Weighted Fair Queueing* (WFQ) para dar soporte a clases de tráfico definidas por el usuario. Para CBWFQ, se definen clases de tráfico basadas en criterios de selección incluyendo protocolos, listas de control de acceso (*Access Control List*, ACL) e interfaces de entrada. Los paquetes que satisfagan los criterios de selección para una clase constituyen el tráfico para esa clase. Se reserva una cola para cada clase, y el tráfico perteneciente a una clase se añade directamente en la cola de esa clase.

Una vez que la clase ha sido definida de acuerdo con los criterios de selección, se le pueden asignar características. Para caracterizar una clase, se le asigna su ancho de banda, peso y límite máximo de paquete. El ancho de banda asignado para la clase es el ancho de banda garantizado para esa clase durante periodos de congestión. Para caracterizar una clase también se puede especificar el tamaño de su cola, es decir, el número máximo de paquetes que se permiten acumular en la cola de dicha clase. Los paquetes pertenecientes a una clase están sujetos al ancho de banda y el tamaño de cola asignados para esa clase.

Después de que una cola alcance el tamaño máximo de cola configurado, el encolado de nuevos paquetes de la clase causa que se produzca *Tail Drop* o *Packet Drop* con WRED, dependiendo de la política que esté configurada. Las clases que usan CBWFQ utilizan *Tail Drop* por defecto, a menos que se configure explícitamente *Weighted Random Early Detection* (WRED) para tirar los paquetes.

El estándar de WFQ es la clasificación por flujos. Esto es, los paquetes con la misma dirección IP origen, dirección IP destino, puerto origen TCP o UDP o puerto destino TCP o UDP, son clasificados como pertenecientes al mismo flujo. WFQ reserva una cantidad igual de ancho de banda para cada flujo. Al *Flow-based* WFQ se le denomina también de encolamiento justo ya que todos los flujos tienen el mismo peso.

Para CBWFQ, que extiende del estándar de WFQ, el peso especificado para la clase será el peso de cada paquete que cumpla los criterios de selección de la clase. Los paquetes que llegan a la interfaz de salida son clasificados de acuerdo con los filtros de selección que definimos, entonces a cada uno se le asigna el peso apropiado. El peso para un paquete que pertenece a una clase determinada se deriva del ancho de banda que se le asignó a la clase cuando se configuró; en ese sentido el peso de cada clase es configurable por el usuario.

Flow-based WFQ aplica pesos al tráfico clasificado en conversaciones y determina el ancho de banda permitido a cada conversación en relación con otras conversaciones. Para *Flow-Based* WFQ, estos pesos y clasificaciones, son dependientes y están limitados a los siete niveles de *IP Precedence*.

CBWFQ permite definir qué constituye una clase, basándose en criterios que exceden los confines del flujo. CBWFQ permite utilizar ACL's y protocolos o nombres de interfaces de entrada para definir cómo será clasificado el tráfico. No se necesita mantener la clasificación del tráfico basada en flujo. Además, se pueden configurar hasta 64 clases discretas en una *service-policy*.

Los pasos que se deben seguir para configurar CBWFQ en el *router* son los siguientes:

- `MiRouter(config)#policy-map mapa_politicas`
- `MiRouter(config-pmap)#class mapa_clases`

A continuación, mediante el comando `bandwith` se especificará la cantidad de kbps que se asignarán a la clase es decir, el peso que tendrán los paquetes que pertenezcan a la cola de esa clase. El comando admite dos posibles opciones; la primera es poner directamente el valor en kbps en el rango [8, 2000000], o especificar un porcentaje del ancho de banda disponible en el rango [1, 100].

- `MiRouter(config-pmap-c)#bandwidth 1000`
- `MiRouter(config-pmap-c)#bandwidth percent 10`

También se puede configurar la clase para que utilice *Tail-Drop* o WRED como modo para que descarte los paquetes. El comando que se utilizará para usar *Tail-Drop* como modo de descarte de paquetes es `queue-limit`. Este método es el que se utilizará por defecto aunque no se introduzca el comando implícitamente en el *router*. El único parámetro que se le debe indicar al comando es el número de paquetes que puede almacenar la cola:

- `MiRouter(config-pmap-c)#queue-limit 128`

Para configurar WRED como modo de descarte de paquetes se usará el comando `random-detect`. Este comando se verá en el siguiente apartado donde se explica con más detalle el método de descarte WRED.

Por último, si se quiere usar el método de *Flow-Based* WFQ se usará el comando `fair-queue` para indicar el número de colas dinámicas, dentro de la configuración de la interfaz:

- `MiRouter(config)#interface ethernet 0/0`
- `MiRouter(config-if)#fair-queue 128`

3.7.5 DiffServ Compliant Weighted Random Early Detection (WRED)

Diffserv Compliant Weighted Random Early Detection WRED [C11] permite que WRED pueda usar los valores del *IP DSCP* y el *IP Precedence* cuando calcula la probabilidad de descartar un paquete. Esta característica se puede usar en conjunto con CBWFQ. Cuando WRED no está configurado el comportamiento del *router* permite rellenar los *buffers* de salida en periodos de congestión usando la característica *Tail Drop* pero esto tirará muchos paquetes en periodos de congestión debido a la falta de *buffer* y se utilizará muy poco el enlace cuando no haya congestión. Por contra, WRED tira paquetes selectivamente cuando la interfaz de salida comienza a mostrar signos de congestión. Tirando algunos paquetes de manera prematura antes de que los *buffers* se llenen, WRED evita tener que tirar grandes cantidades de paquetes a la vez. De esta forma, WRED permite que la línea de transmisión se use completamente todo el tiempo.

Si se configura una clase dentro de un *policy-map* para usar WRED para el descarte de paquetes en vez de *Tail Drop*, hay que asegurarse de que WRED no está configurado sobre la interfaz en la que se tiene intención de asociar esa *service-policy*.

WRED se puede configurar para que emplee los bits del *IP Precedence* o los bits del *IP DSCP* a la hora de descartar paquetes. Es posible configurar WRED a nivel de interfaz, a nivel de clase o a nivel de circuito virtual. En este caso, sólo veremos el nivel de clase, siguiendo el procedimiento descrito en el apartado 3.5.1.

Los comandos que se utilizarán para configurar WRED sobre una clase dentro de un mapa de políticas serán los siguientes:

- `MiRouter(config)#policy-map mapa_politicas`

- `MiRouter(config-pmap)#class mapa_clases`

A continuación se activará CBWFQ mediante el comando `bandwidth` reservando una cantidad de ancho de banda para la clase especificada en kbps o en un porcentaje del ancho de banda total disponible para la interfaz.

- `MiRouter(config-pmap-c)#bandwith percent 15`

Tras esto, deberemos activar WRED mediante el commando `random-detect` sin opciones:

- `MiRouter(config-pmap-c)#random-detect`

A continuación es posible usar WRED basado en *IP DSCP* o en *IP Precedence*. Para activar WRED basado en *IP DSCP* usaremos los siguientes comandos:

- `MiRouter(config-pmap-c)#random-detect dscp-based`
- `MiRouter(config-pmap-c)#random-detect dscp af11 128 256 512`

Las opciones de este último comando son:

- AF11. Valor de DSCP de coincidencia.
- 128. Umbral mínimo (número de paquetes). De 1 a 4096.
- 256. Umbral máximo (número de paquetes). De 1 a 4096.
- 512. Denominador de probabilidad. De 1 a 65536. Parámetro opcional.

Si lo que se quiere es usar WRED basado en *IP Precedence* se usarán los siguientes comandos:

- `MiRouter(config-pmap-c)#random-detect prec-based`
- `MiRouter(config-pmap-c)#random-detect precedence 0 128 256 512`

Las opciones de este último comando son las siguientes:

- 0. Valor de IP Precedence o RVSP.
- 128. Umbral mínimo (número de paquetes). De 1 a 4096.
- 256. Umbral máximo (número de paquetes). De 1 a 4096.
- 512. Denominador de probabilidad. De 1 a 65536. Parámetro opcional.

Por último es posible especificar el peso para el cálculo del valor medio de la cola usado, mediante el siguiente comando (Cisco recomienda usar el valor 9, que es el que se usa por defecto):

- `MiRouter(config-pmap-c)#random-detect exponential-weighting-constant 9`

3.7.6 Low Latency Queueing (LLQ)

La herramienta de encolamiento de baja latencia *Low Latency Queueing* LLQ [C12] lleva la disciplina de servicio de colas por encolamiento de prioridad estricta (*Strict Priority Queueing*) a CBWFQ. *Strict Priority Queueing* permite que tanto el tráfico sensible al retardo como el tráfico de voz sea desencolado y enviado antes que paquetes de otras colas, dándole al tráfico sensible al retardo un tratamiento preferente sobre otros tipos de tráfico.

Sin LLQ, CBWFQ sirve todos los paquetes de manera equitativa basándose en su peso, pero no proporciona prioridad estricta (*strict priority*) para ninguna clase de paquetes. Esto puede suponer un problema para el tráfico de voz que es muy sensible al retardo y especialmente a la varianza del retardo o *jitter*.

LLQ proporciona *Strict Priority Queueing* a CBWFQ reduciendo la varianza del retardo *jitter* en las conversaciones de voz. Cuando se activa LLQ se emplea una única cola de prioridad estricta (*Strict Priority Queue*) dentro de CBWFQ a nivel de clase, permitiendo llevar el

tráfico perteneciente a una clase a una CBWFQ *Strict Priority Queue*. Dentro de una *policy-map* se pueden configurar más de una clase para que usen LLQ pero todo el tráfico de esas clases será encolado dentro de la misma *Strict Priority Queue*.

Una de las diferencias entre el *Strict Priority Queueing* empleado dentro de CBWFQ y el que se emplea sin CBWFQ está en los parámetros que toma. Fuera de CBWFQ, mediante el uso del comando `ip rtp priority`, se da prioridad solamente a los flujos del tráfico de voz que entran por un rango de puertos UDP. Mientras que en el *Strict Priority Queueing* empleado dentro de CBWFQ se pueden emplear muchos criterios de selección de tráfico como pueden ser: listas de control de accesso (*Access Control List*, ACL), protocolos, interfaces de entrada e incluso los valores de los *IP DSCP* o *IP Precedence*. Aunque es posible aplicar LLQ para varios tipos de aplicaciones de tiempo real, Cisco aconseja emplearlo solamente para el tráfico de voz.

Cuando se configura LLQ mediante el comando `priority` para una clase, toma un ancho de banda como argumento que es el ancho de banda máximo en kilobits por segundo (kbps). Este parámetro garantiza un ancho de banda para la clase `priority` pero también acota el flujo de paquetes de esa clase.

Si se produce congestión, cuando el ancho de banda configurado se excede se emplea un algoritmo de *token bucket* para descartar paquetes, midiéndose el tráfico destinado a la cola *priority* para asegurar que se cumple el ancho de banda configurado para el tráfico de la clase. El tráfico de voz encolado en la cola *priority* es UDP, por lo tanto no se adapta al descarte de paquetes realizado por WRED. Debido a que WRED es ineficiente, no se podrá usar WRED (comando `random-detect`) con el comando `priority`. Además, como se emplea un *policing* para descartar paquetes y no hay límites de cola impuestos, el comando `queue-limit` tampoco se podrá usar con el comando `priority`.

Las clases son tratadas por las funciones de *policing* de manera individual. Esto quiere decir que aunque una única *policy map* pueda contener cuatro clases prioritarias y se encolen todas en una única cola prioritaria, se tratan cada una como flujos de tráfico separados.

Esta herramienta será útil para gestionar el tráfico de la clase *Expedited Forwarding* EF de los Servicios Diferenciados y darle un tratamiento preferente frente a otros tipos de tráfico como el tráfico de la clase *Assured Forwarding*. El tráfico de la clase EF tendrá requerimientos de poco retardo y poca varianza del retardo (*jitter*). Cisco IOS proporciona una solución a esas necesidades mediante la herramienta LLQ.

Los pasos para activar LLQ sobre una clase de tráfico determinada son los siguientes:

- `MiRouter(config)#policy-map mapa_politicas`
- `MiRouter(config-pmap)#class mapa_clases`
- `MiRouter(config-pmap-c)#priority 5000 64`

El comando `priority` posee dos parámetros:

  - El primero de ellos indica el ancho de banda en *kilobits* por segundo.
  - El segundo parámetro (opcional) indica el tamaño de la ráfaga en *bytes*.

Como se ha podido comprobar en este apartado y en el anterior, los comandos `priority` y `bandwidth` permiten especificar un ancho de banda garantizado para una cierta clase de tráfico, pero existen importantes diferencias entre ambos comandos. En periodos de congestión ambos comandos proporcionan un ancho de banda mínimo asegurado para las clases de tráfico. La diferencia está en que el comando `priority` también implementa un ancho de banda máximo garantizado. Internamente, la cola prioritaria emplea un *token bucket* que mide la carga ofrecida y asegura que el flujo de tráfico no excede la tasa configurada. Sólo al tráfico conforme con el *token bucket* se le garantiza baja latencia. Por eso al contrario que ocurre para las clases configuradas con el comando `bandwidth`, las clases que usan el comando `priority` no usan nunca el ancho de banda que sobra, durante periodos de congestión.

Capítulo 4
Desarrollo de la herramienta

4.1 Introducción

En este capítulo se explicará el proceso de desarrollo de la herramienta. En primer lugar se analizarán las clases referentes a la comunicación con el puerto serie. A continuación se pasará a desarrollar cada una de las clases que componen la interfaz gráfica y demás clases que, aunque no poseen representación gráfica, son necesarias en el proyecto. Detallaremos su funcionalidad y su cometido bien en la configuración de opciones generales del *router* o bien de opciones específicas de servicios diferenciados.

4.2 Programación con JBuilder

La herramienta utilizada para el desarrollo del proyecto ha sido JBuilder 9, Personal Edition, de la compañía de software Borland. Esta versión es gratuita para uso personal, siempre y cuando no se distribuyan los programas generados con esta herramienta con ánimo de lucro.

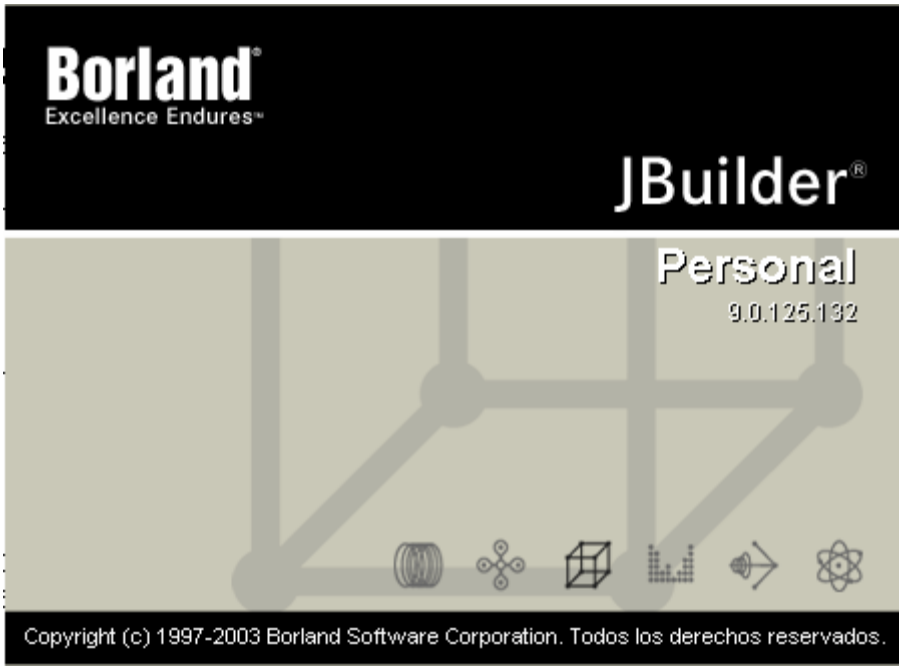


Figura 11 - Pantalla de presentación de Borland JBuilder 9

La herramienta permite diseñar interfaces gráficas arrastrando y soltando componentes, y en la versión usada es compatible con la versión 1.4 del JDK de Java, es decir, es compatible con componentes modernos como los `JSpinner`.

Además posee herramientas correctoras del código conforme se escribe, indicando dónde se produce el error; permite la optimización de las clases importadas; formateo automático del código; depuración de las clases; etc.

Este paquete incorpora además nuevas clases para la colocación de componentes, que han sido usadas en esta aplicación, como la clase `XYLayout` y la `VerticalFlowLayout`.

4.2.1 Creación de interfaces gráficas

Como se ha dicho, JBuilder permite la generación de interfaces gráficas con solo arrastrar y soltar los componentes. Todo el código generado para inicializar la interfaz gráfica está dentro del método generado automáticamente llamado `jbInit()`.

4.2.2 Manejo de eventos

JBuilder genera automáticamente el código para manejar los eventos generados cuando, por ejemplo, se pulsa un botón de la interfaz gráfica. Para ello, JBuilder crea una clase suplementario en el mismo archivo, y cuyo método manejador de eventos llamará a un método de la clase inicial donde se insertará el código correspondiente.

Un ejemplo se muestra a continuación:

```
public class Ventana_Principal extends JFrame  {

   private void jbInit() throws Exception  {
      MenuItem_Archivo_Salir.addActionListener(new
         Ventana_Principal_MenuItem_Archivo_Salir_ActionAdapter(this));
   }

   public void MenuItem_Archivo_Salir_actionPerformed(ActionEvent e) {
      salir();
   }

}

// Clase manejadora del evento

class Ventana_Principal_MenuItem_Archivo_Salir_ActionAdapter implements ActionListener {
   Ventana_Principal adaptee;
   Ventana_Principal_MenuItem_Archivo_Salir_ActionAdapter(Ventana_Principal adaptee) {
      this.adaptee = adaptee;
   }
   public void actionPerformed(ActionEvent e) {
      adaptee.MenuItem_Archivo_Salir_actionPerformed(e);
   }
}
```

4.3 Estructura del proyecto

A continuación se pasan a describir los paquetes que se han creado para ordenar las clases de las que se compone el programa:

- Paquete `cisco` (paquete raíz)
  - Clase `Cisco`
  - Clase `DobleSalida`
  - Clase `Ventana_Principal`
  - Clase `NavegadorAyuda`
  - Clase `Opciones`
  - Clase `Propiedades`
  - Clase `Timer`

- Clase `Utilidades`
- Paquete `AccessLists`
  - Clase `AccessLists`
  - Clase `Lista`
  - Interfaz `Interfaz`
  - Clase `Lista_Estandar`
  - Clase `Lista_Extendida_IP`
  - Clase `Lista_Extendida_TCP_Puertos`
  - Clase `Lista_Extendida_TCP`
  - Clase `Lista_Extendida_UDP_Puertos`
  - Clase `Lista_Extendida_UDP`
- Paquete `AsocPolicyMapInterfaz`
  - Clase `ConfigurarAsociacionPolicyMapInterfaz`
- Paquete `ClassMaps`
  - Clase `ClassMap`
  - Clase `ClassMap_Condicion`
- Paquete `Dialogos`
  - Interfaz `CancelarComunicacion`
  - Clase `ComunicandoConRouter`
  - Clase `InformarExcepcion`
  - Clase `MensajeDeEspera`
  - Clase `SemiModalDialog`
  - Clase `Ventana_Principal_AboutBox`
- Paquete `Interfaces`
  - Interfaz `DobleSalida_Escuchador`
  - Interfaz `EscuchadorEventos`
  - Interfaz `EscuchadorEventosTimer`
- Paquete `IPs`
  - Clase `ConfigurarIPs`
- Paquete `PolicyMaps`
  - Clase `PolicyMap`
  - Clase `PolicyMap_Clase`
  - Clase `PolicyMap_Politica`
- Paquete `Tablas`
  - Clase `Table_Cisco`
  - Clase `Table_JLabelTableCellRenderer`
  - Clase `Table_Model_Cisco`

4.4 Paquete cisco

4.4.1 Clase Cisco

La clase Cisco es la clase contiene el método `main()`, gracias al cual se puede ejecutar la aplicación. Esta clase contiene una variable estática de la clase Propiedades para que las demás clases del proyecto puedan acceder a las opciones del programa que el usuario haya seleccionado en alguna ejecución anterior. Esta clase posee únicamente dos métodos que se describen a continuación:

- `public static void main(String[] args)`. Este método es el que se ejecuta al iniciar la aplicación. Lo único que hace es crear un objeto de la clase Cisco.
- `public Cisco()`. Primeramente, este método crea un objeto de la clase MensajeDeEspera, que muestra una ventana indicando al usuario que el programa se está cargando. Tras ello se inicializa la variable estática de la clase Propiedades, llamando al constructor de la clase, pasándole como parámetro el archivo de propiedades que se va a usar. A continuación establece el estilo de las ventanas de acuerdo con las opciones guardadas. Tras ello crea un nuevo objeto de la clase Ventana_Principal, lo muestra y oculta el mensaje de espera.

4.4.2 Clase Ventana_Principal

Esta clase contiene la ventana principal del programa. La clase hereda de `JFrame` e implementa las interfaces `EscuchadorEventos`, `DobleSalida_Escuchador` y `EscuchadorEventos_Timer`.

De modo general, esta clase representa la interfaz gráfica de la ventana principal del programa. Contiene una barra de menús, una barra de herramientas, una serie de pestañas y una barra de estado. La barra de menús contiene comandos para trabajar con la configuración del *router*, los historiales, el puerto serie, los modos de administración y ayuda. La barra de herramientas contiene casi todos los comandos que se han implementado en la barra de menús. La primera de las pestañas es una pestaña de bienvenida donde se presenta un resumen del estado del programa y del *router*. La segunda presenta una serie de iconos para configurar opciones generales del *router*. La tercera contiene comandos para configurar servicios diferenciados. La cuarta pestaña contiene un terminal básico, con botones para el manejo de históricos y el envío de secuencias especiales de teclas. Para terminar, la última pestaña posee una tabla con los eventos que van sucediendo de forma interna en el programa.

Las variables de instancia más importantes que contiene esta clase son las siguientes:

- `Bolean puerto_abierto`. Indica si el puerto serie está abierto.
- `Bolean administración`. Indica si el programa se encuentra en modo de administración del *router* (EXEC privilegiado).
- `DobleSalida ds`. Objeto de la clase DobleSalida que implementa métodos para lectura y escritura del puerto serie.
- `CommPortIdentifier cpi`. Objeto que contiene el identificador del puerto seleccionado en las propiedades.
- `SerialPort sp`. Puerto serie correspondiente al identificador del puerto de comunicaciones anterior.

- `StringBuffer buffer_logrouter`. Cadena que contiene los últimos datos que se han recibido por el puerto serie. Es usado para comprobar si se ha producido un reinicio del *router* o si se ha iniciado el asistente de configuración del *router*.
- `Int contador_eventos`. Contador que lleva la cuenta del número de eventos que se generan.
- `Timer t`. Objeto utilizado al abrir el puerto para comprobar si hay algún dispositivo conectado en el puerto serie que dé respuesta en un tiempo determinado.
- `Bolean fin_timer`. Variable utilizada por el objeto anterior para indicar si el timer ha expirado.

A continuación se describen los métodos disponibles en esta clase en un orden aproximado de ejecución:

- `public Ventana_Principal()`. Método constructor de la clase. No recibe ningún parámetro. Llama a los métodos para construir la interfaz gráfica, centra el marco en la pantalla, termina de construir la interfaz gráfica y sus detalles, inicializa el puerto serie, y por último crea un resumen del estado actual del *router*.
- `protected void inicializarPuerto()`. Este método intenta abrir el puerto indicado en las opciones. En primer lugar obtiene el identificador del puerto correspondiente al puerto serie de las opciones. Esto puede lanzar una excepción `NoSuchPortException` en caso de que el identificador que se ha pasado como parámetro no sea correcto. Tras ello comprueba que es un puerto serie, y en caso contrario, muestra un mensaje de error. A continuación establece las opciones del puerto serie: velocidad, bits de datos, bits de parada, control de flujo y paridad. Estos métodos pueden generar una excepción `UnsupportedCommOperationException` que es capturada en caso de producirse. Después, se intentan obtener los flujos de entrada y salida para con ellos crear un nuevo objeto de la clase `DobleSalida`. A continuación se manda un retorno de carro (“\r\n”), una señal de Control+C para terminar cualquier proceso que se estuviera ejecutando, y se vuelve a escribir un retorno de carro para rescatar la línea de comandos. Tras esto se comprobará si hay algún dispositivo conectado al puerto serie. Para ello primero se crea un objeto `Timer` pasando como parámetro el tiempo especificado en las opciones, y se arranca. A continuación se crea un bucle infinito, comprobando primeramente si se ha expirado el *timer*, en cuyo caso se muestra un mensaje indicando que no se ha encontrado ningún mensaje en el tiempo especificado en las opciones. Si se consigue contactar con el *router* en el tiempo especificado, se leen todos los datos que haya disponibles y se comprueba si esa cadena termina en el carácter ‘#’, en cuyo caso indicará que el *router* se encuentra en modo administración; en caso contrario el *router* se encuentra en modo usuario.
- `private void resumen()`. Este método crea un resumen del estado actual del *router*. En primer lugar comprueba si el puerto está abierto o no, mostrando en el primer caso las opciones de configuración del puerto. Tras ello comprueba si se está en modo de administración o en modo de usuario, indicándolo también por pantalla. Por último hace uso de los métodos de la clase `DobleSalida` `getInterfaces()`, `getClassMaps()` y `getPolicyMaps()` para mostrar respectivamente las interfaces, los mapas de clases y los mapas de políticas configurados en el *router*.
- `public void finTimer()`. Este método es llamado por la clase `Timer` cuando ha expirado. Pone la variable `fin_timer` a `true` y detiene el *timer* que se estaba ejecutando.
- `private void setBarraDeEstado(String s)`. Establece un texto en la barra de estado.
- `private void restauraBarraDeEstado()`. Pone el texto por defecto en la barra de estado.

- `public void recibe(String s)`. Método llamado por la clase `DobleSalida` y especificado por la interfaz `DobleSalida_Escuchador`. Añade la cadena que se le pasa como parámetro a la ventana de terminal y al mismo tiempo a la variable `buffer_logrouter`, comprobando tras ello si ese buffer contiene las cadenas de reinicio de *router* o de un inicio del asistente de configuración, en cuyo caso se detendrá el asistente y a continuación se cerrará y se abrirá el puerto para comprobar el modo en el que se encuentra.
- `public void anadir_evento(String que)`. Añade un evento con la descripción que se pasa como parámetro, la fecha y hora actuales y el contador de eventos actual, incrementándolo después.
- `public void anadir_evento(String contador, String fecha, String hora, String que, boolean log)`. Este método realmente es el que añade un evento a la tabla de eventos (la última pestaña de la interfaz gráfica). Si la variable `log` está activada, indica que el evento ha sido producido al leer el archivo de historial de eventos, por lo que tendrá un color de fondo distinto a los eventos generados durante la ejecución del programa. Si la propiedad de mostrar los eventos por la salida estándar está activada, hará lo correspondiente, y por último hace visible la última fila de la tabla de eventos.
- `public boolean isAdministrador()`. Este método está especificado por la interfaz `EscuchadorEventos` e indica si se está en modo administrador o no.
- `private void inicializarCosas()`. Este método finaliza de construir la interfaz gráfica, estableciendo los iconos correspondientes a las etiquetas, botones, menús y pestañas. Tras ello configura la tabla de eventos, el encabezado y el ancho de las columnas. Por último, si las variables del archivo de propiedades están a un valor concreto, se leerán los archivos de historiales de eventos y de *logs* (terminal) y los añade a la tabla de eventos y a la ventana de terminal respectivamente.
- `protected void cerrarPuerto()`. Este método cierra el puerto serie y pone todas las variables asociadas a valores coherentes con el nuevo estado del programa.
- `private void inicializarModos()`. De acuerdo con las variables `puerto_abierto` y `administracion`, cambia los menús correspondientes a los valores que indican dichas variables.
- `private void jbInit() throws Exception`. Método generado automáticamente por JBuilder, donde se genera la interfaz gráfica. Este método es común a todas las clases que poseen interfaz gráfica, por lo que no se volverá a mencionar.
- `private void salir()`. Método llamado cuando se pulsa el botón de cerrar de la ventana, o cuando se pulsa el menú Archivo, Salir. Comprueba si el puerto está abierto, cerrándolo en su caso. Tras ello comprueba si se deben guardar los eventos y el historial de logs de acuerdo con las opciones de la clase `Propiedades`, haciendo lo apropiado.
- `private String fecha_evento()`. Devuelve la fecha actual en formato *dd/mm/aa*.
- `private String hora_evento()`. Devuelve la hora actual en formato *hh:mm:ss*.

Métodos referentes a los menús

- `protected void MenuItem_Archivo_Guardar_Configuracion_Inicial_actionPerformed(ActionEvent e)`. Método que se ejecuta al seleccionar la opción Archivo, Guardar configuración como inicial. Si el puerto está abierto, y se encuentra en modo administración, tras una confirmación se manda el comando `show running-config startup-config`. Tras mandar el comando, el *router*

pregunta el nombre del archivo con el que se quiere guardar; por ello, se manda un retorno de carro aceptando la opción que se da por defecto.

- `protected void MenuItem_Archivo_Cargar_Configuracion_Inicial_actionPerformed(ActionEvent e)`. Método correspondiente al menú Archivo, Cargar configuración inicial. Si el Puerto está abierto y se está en modo administración, se envía el comando `copy startup-config running-config`. Si se detecta que el *router* pide una confirmación debido a que la configuración ha cambiado, se envía un retorno de carro aceptando la confirmación.

- `protected void MenuItem_Archivo_ReiniciarRouter_actionPerformed(ActionEvent e)`. Método correspondiente al menú Archivo, Reiniciar *router*. Si el Puerto está abierto y se encuentra en modo administración, se manda un comando `reload`. Tras el comando y si se ha modificado la configuración, el *router* preguntará si se quieren guardar los cambios. En este caso se responderá que no, mandando una cadena "`no\n`".

- `protected void MenuItem_Archivo_Opciones_actionPerformed(ActionEvent e)`. Método correspondiente el menu Archivo, Opciones. Crea una nueva instancia de la clase `Opciones` y la hace visible.

- `protected void MenuItem_LOG_CargarLOG_actionPerformed(ActionEvent e)`. Pregunta al usuario sobre el archivo LOG a cargar y después lo establece como texto en la ventana de terminal.

- `protected void MenuItem_LOG_AnadirLOG_actionPerformed(ActionEvent e)`. Pregunta al usuario sobre el archivo LOG a añadir y después lo añade al contenido actual de la ventana del terminal.

- `protected void MenuItem_LOG_GuardarLOG_actionPerformed(ActionEvent e)`. Pregunta al usuario el nombre del archivo donde quiere guardar el contenido de la ventana de terminal. Tras ello comprueba si el archivo existe, advirtiéndolo, y tras ello guarda el texto de dicha ventana en el archivo especificado.

- `protected void MenuItem_LOG_BorrarLOG_actionPerformed(ActionEvent e)`. Elimina el contenido de la ventana de terminal.

- `protected void RadioButtonMenuItem_Puerto_Abierto_actionPerformed(ActionEvent e)`. Método correspondiente al menú Puerto, Abrir. Hace una llamada al método `inicializarPuerto()` si el puerto está cerrado.

- `protected void RadioButtonMenuItem_Puerto_Cerrado_actionPerformed(ActionEvent e)`. Método correspondiente al menú Puerto, Cerrar. Hace una llamada al método `cerrarPuerto()` si el puerto está abierto.

- `protected void RadioButtonMenuItem_Modo_Usuario_actionPerformed(ActionEvent e)`. Método correspondiente al menú Modo, Usuario. Manda un comando `disable` desde el modo EXEC privilegiado.

- `protected void RadioButtonMenuItem_Modo_Administrador_actionPerformed(ActionEvent e)`. Método correspondiente al Menú Modo, Administrador. Solicita la contraseña al usuario, y la envía al *router* tras introducir el comando `enable` desde el modo EXEC. Se le dan tres oportunidades.

- `protected void MenuItem_Ayuda_PaginaInicialAyuda_actionPerformed(ActionEvent e)`. Método correspondiente al menú Ayuda, Página inicial de la ayuda. Usa el método estático `mostrar()` de la clase `NavegadorAyuda` para mostrar la página correspondiente de la ayuda.

- `protected void MenuItem_Ayuda_AcercaDe_actionPerformed(ActionEvent e)`. Método correspondiente al menú Ayuda, acerca de. Crea una nueva instancia de la clase `Ventana_Principal_AboutBox` y la hace visible.

Métodos referentes a las opciones de configuración

- `protected void Label_ConfiguracionDireccionesInterfaces_mouseClicked(MouseEvent e)`. Comprueba si el puerto serie está abierto y crea una nueva instancia de la clase `ConfigurarIPs`, haciéndola después visible.
- `protected void Label_RIP_mouseClicked(MouseEvent e)`. Método ejecutado cuando se pulsa la etiqueta para configurar el enrutamiento RIP. Si el puerto está abierto y se encuentra en modo administración, manda los comandos `ip routing` y `router rip`, para a continuación solicitar al usuario todas las redes a las que está conectado el *router*. Comprueba si el dato introducido es una dirección IP correcta y tras ello manda el comando `network` seguido de la red introducida por el usuario.
- `protected void Label_Pass_EXEC_mouseClicked(MouseEvent e)`. Método ejecutado cuando se pulsa la etiqueta para cambiar la contraseña de administrador del *router*. Si el puerto está abierto y se encuentra en modo administrador, solicita al usuario la nueva contraseña y envía el comando `enable secret` seguido de la contraseña introducida desde el modo de configuración de terminal.
- `protected void Label_Cambiar_Nombre_Router_mouseClicked(MouseEvent e)`. Método que se ejecuta cuando se pulsa la etiqueta para cambiar el nombre al *router*. Solicita al usuario el nuevo nombre del *router* y tras ello, envía el comando `hostname` seguido del texto introducido por el usuario, desde la configuración de terminal del modo EXEC privilegiado.
- `protected void Label_Configuracion_Clases_mouseClicked(MouseEvent e)`. Método que se ejecuta cuando se pulsa la etiqueta para configurar los mapas de clases. Si el puerto está abierto, crea una nueva instancia de la clase `ClassMap` y la hace visible.
- `protected void Label_Configuracion_PolicyMap_mouseClicked(MouseEvent e)`. Método que se ejecuta cuando se pulsa la etiqueta para configurar los mapas de políticas. Crea una nueva instancia de la clase `ConfigurarPolicyMap` y la hace visible.
- `protected void Label_AsociacionPoliticasInterfaces_mouseClicked(MouseEvent e)`. Método que se ejecuta cuando se pulsa la etiqueta para asociar los mapas de políticas a las interfaces. Crea una nueva instancia de la clase `ConfigurarAsociacionPolicyMapInterfaz` y la hace visible.
- `protected void Label_Configuracion_ListasAcceso_mouseClicked(MouseEvent e)`. Método ejecutado cuando se pulsa la etiqueta para configurar las listas de control de acceso. Si el puerto está abierto, crea una nueva instancia de la clase `AccessLists` y la hace visible.

Métodos referentes a otros objetos de la interfaz gráfica

- `protected void Button_Actualizar_Resumen_actionPerformed(ActionEvent e)`. Método ejecutado cuando se pulsa el botón para actualizar resumen de la primera pestaña. Simplemente hace una llamada al método `resumen()`.
- `protected void Button_Info_Completa_actionPerformed(ActionEvent e)`. Método ejecutado cuando se pulsa el botón para mostrar toda la configuración del *router*, disponible en la primera pestaña cuando se está en modo administrador. Envía una señal de Control+Z (*byte* 26) y tras ello el comando `show startup-config`.

- `protected void Button_EnviarSenalEscape_actionPerformed( ActionEvent e)`. Manda una señal de escape (*byte* 30 y *byte* 120) si el puerto serie está abierto.
- `protected void Button_RescatarLineaComandos_actionPerformed( ActionEvent e)`. Cierra el puerto (método `cerrarPuerto()`) si ya está abierto y lo vuelve a abrir (método `inicializarPuerto()`), ejecutando de nuevo la cadena de inicialización.
- `protected void Button_CtrlZ_actionPerformed(ActionEvent e)`. Método ejecutado cuando se pulsa el botón para enviar una señal de Control+Z. Envía el *byte* 26, correspondiente a esta combinación de teclas.
- `protected void Button_CtrlC_actionPerformed(ActionEvent e)`. Método ejecutado cuando se pulsa el botón para enviar una señal de Control+C. Envía el *byte* 3, correspondiente a esta combinación de teclas.
- `protected void TextField_IntroduceComando_actionPerformed( ActionEvent e)`. Este método es ejecutado cuando se escribe algún texto en la casilla del terminal y se pulsa la tecla *Intro*. Envía el comando escrito, añadiendo un retorno de carro, a través del puerto serie. Antes de hacer esto, se comprueba si el usuario está intentando cambiar de modo (entre EXEC y EXEC privilegiado), en cuyo caso se le informa de que para hacer eso debe hacerlo a través del menú modo.
- `protected void Button_IntroduceComandoNoCR_actionPerformed( ActionEvent e)`. Hace lo mismo que el método `TextField_IntroduceComando_actionPerformed()` pero sin añadir un retorno de carro.

4.4.3 Clase Ventana_Principal_AboutBox

Esta clase muestra una ventana mostrando información acerca del programa, el autor y los datos referentes al Proyecto Final de Carrera. La clase extiende de `SemiModalDialog`, explicada posteriormente dentro del paquete `Dialogos`. El constructor de esta clase recibe los mismos parámetros que un `JDialog`. Tras llamar al constructor de la clase padre, llama al método `jbInit()`, que inicializa la interfaz gráfica. Tras ello, mediante el método `pack()`, ajusta el tamaño de la ventana al contenido y lo centra en la pantalla.

4.4.4 Clase DobleSalida

La clase `DobleSalida` se encarga de las comunicaciones entrantes y salientes del puerto serie. Dispone de métodos típicos en las clases como `InputStream` y `OutputStream`, además de otros métodos específicos necesarios para esta aplicación en concreto. Esta clase además tiene la opción de mostrar una ventana (por eso extiende de `JFrame`) a modo de terminal en la que se puede observar la comunicación que se leva a cabo con el *router*. Esta clase además implementa las interfaces `Runnable` y `CancelarComunicacion`.

La primera de las interfaces mencionadas, `Runnable`, es necesaria ya que esta clase, una vez creada, está continuamente intentando leer del puerto, en un hilo independiente del programa principal.

La otra interfaz, `CancelarComunicación`, es necesaria para interrumpir el proceso de comunicación con el *router*. Cuando se están mandando o recibiendo datos del *router*, se muestra una ventana de la clase `ComunicandoConRouter`. Esta ventana recibe como uno de los argumentos del constructor un objeto que implemente la interfaz `CancelarComunicacion`. Esta nueva ventana posee un botón para interrumpir el proceso de comunicación con el *router*. Si se pulsa dicho botón, la clase `ComunicandoConRouter` llamará al método `cancelarComunicacion()` del objeto `CancelarComunicacion` que se

le ha pasado como parámetro, indicando que el usuario quiere cancelar la comunicación. Esto hará que la clase `DobleSalida`, dentro del método `cancelarComunicacion()` especificado por la interfaz, cambie el valor, indicando que se cancele la comunicación.

El constructor de la clase posee el siguiente formato:

- `public DobleSalida(InputStream is, OutputStream os, DobleSalida_Escuchador ds, EscuchadorEventos ee)`. Un objeto de la clase `InputStream` y otro de la clase `OutputStream`, usados para leer y escribir respectivamente por el puerto serie. Un objeto que implemente la interfaz `DobleSalida_Escuchador`, para que vaya recibiendo los datos conforme los recibe esta clase. Y por último un objeto de la clase `EscuchadorEventos`, para agregar eventos que sucedan en el programa.

La clase contiene los siguientes métodos:

- `public void cancelarComunicacion()`. Método especificado por la interfaz `CancelarComunicacion`. Pone la variable de instancia `cancelar` a `true`.
- `public void parar()`. Detiene el *thread* asociado a este objeto, y oculta la ventana de terminal.
- `public void run()`. Este método debe ser implementado según la interfaz `Runnable`. Este método lee continuamente desde el puerto serie, y guarda los datos que lee en un `StringBuffer`, variable de instancia de la clase. A su vez, cada vez que lee datos, los convierte en una cadena y llama al método recibe, de la clase que implementa la interfaz `DobleSalida_Escuchador` que se pasa como parámetro al constructor. También añade los datos recibidos al terminal asociado a la clase.

Esta clase provee los siguientes métodos para únicamente leer datos del puerto serie.

- `public int read()`. Lee un byte del `StringBuffer`.
- `public int read(char b[])`. Lee hasta un máximo de *bytes* correspondiente a la longitud del *array* que se pasa como parámetro del `StringBuffer`. Devuelve el número de *bytes* leídos.

También provee métodos para escribir datos en el Puerto serie:

- `public void write(byte b[]) throws IOException`. Manda por el puerto serie un *array* de *bytes*.
- `public void write(int i) throws IOException`. Envía un *byte* por el puerto serie.

Como se ha dicho, esta clase contiene métodos especiales para este programa en concreto:

- `public Vector writeCtrlZ(Dialog parent) throws IOException`. Crea un nuevo objeto de la clase `ComunicandoConRouter` cuyo padre será el diálogo que se le pasa como parámetro y ejecuta el método `writeCtrlZ( ComunicandoConRouter ccr)`.
- `public Vector writeCtrlZ(Frame parent) throws IOException`. Crea un nuevo objeto de la clase `ComunicandoConRouter` cuyo padre será el marco que se le pasa como parámetro y ejecuta el método `writeCtrlZ( ComunicandoConRouter ccr)`.
- `private Vector writeCtrlZ(ComunicandoConRouter ccr) throws IOException`. Este método envía una señal de Control+Z (*byte* 26). Tras ello, se espera el tiempo indicado en las opciones y a continuación intenta leer del puerto serie la respuesta ante tal comando. Para ello, crea un bucle infinito, comprobando primero si el usuario ha cancelado el diálogo (variable `cancelar`), y si no se ha cancelado, intenta leer desde el puerto serie, añadiendo los datos leídos a una nueva cadena de caracteres. Una vez que se ha terminado de leer, se convierte esa

cadena en un `Vector`, correspondiente a cada una de las líneas devueltas por el *router* ante tal comando.

- `public Vector writeStr(String str, Dialog parent) throws IOException`. Crea un nuevo objeto de la clase `ComunicandoConRouter` cuyo padre será el diálogo que se le pasa como parámetro y ejecuta el método `writeStr(String str, ComunicandoConRouter ccr)`.
- `public Vector writeStr(String str, Frame parent) throws IOException`. Crea un nuevo objeto de la clase `ComunicandoConRouter` cuyo padre será el marco que se le pasa como parámetro y ejecuta el método `writeStr(String str, ComunicandoConRouter ccr)`.
- `private Vector writeStr(String str, ComunicandoConRouter ccr) throws IOException`. Este método envía el comando indicado en la cadena que se le pasa como parámetro. Tras ello, se espera el tiempo indicado en las opciones y a continuación intenta leer del puerto serie la respuesta ante tal comando. Para ello, crea un bucle infinito, comprobando primero si el usuario ha cancelado el diálogo (variable `cancelar`), y si no se ha cancelado, intenta leer desde el puerto serie, añadiendo los datos leídos a una nueva cadena de caracteres. Una vez que se ha terminado de leer, se convierte esa cadena en un `Vector`, correspondiente a cada una de las líneas devueltas por el *router* ante tal comando.
- `public String[] getInterfaces()`. Este método devuelve un *array* de `String` correspondiente a las interfaces que posee el *router*. El comando utilizado para tal fin es `show interfaces`, cuya salida es ésta:

```
MiRouter>show interfaces
Ethernet0/0 is up, line protocol is up
  Hardware is AmdP2, address is 0007.853b.2f00 (bia 0007.853b.2f00)
  Internet address is 192.168.2.254/24
[…]
Serial0/0 is down, line protocol is down
  Hardware is PowerQUICC Serial
  Internet address will be negotiated using IPCP
[…]
Ethernet0/1 is up, line protocol is down
  Hardware is AmdP2, address is 0007.853b.2f01 (bia 0007.853b.2f01)
  Internet address is 192.168.1.5/24
[…]
Serial0/1 is down, line protocol is down
  Hardware is PowerQUICC Serial
  Internet address is 192.168.4.52/24
[…]
MiRouter>
```

Para obtener el nombre de las interfaces se sigue el siguiente procedimiento:

  - Seleccionar aquellas líneas que no contienen un espacio al inicio.

  - De cada una de esas líneas, coger la cadena que va desde el inicio de la línea hasta el primer espacio.

- `public String[] getClassMaps()`. Devuelve un *array* de `String` con todos los mapas de clases configurados en el *router*. El comando usado para tal fin es show class-map | include Class Map, cuya salida se muestra a continuación:

```
MiRouter>show class-map | include Class Map
 Class Map match-all nueva_clase_2 (id 2)
 Class Map match-any class-default (id 0)
 Class Map match-all mapa_clases (id 3)
 Class Map match-all nueva_clase (id 4)
 Class Map match-all lamia2 (id 5)
 Class Map match-all lamia3_nueva (id 6)
MiRouter>
```

  El procedimiento utilizado para obtener el nombre de los mapas es el siguiente:

  - De cada una de las líneas de la respuesta, eliminar lo que haya después del espacio y el paréntesis (" (").
  - Tras ello, seleccionar la cadena que va desde el último espacio hasta el final de la línea.

- `public String[] getPolicyMaps()`. Devuelve una lista con los mapas de políticas creados en el *router*. El comando utilizado para tal fin es `show policy-map | include Policy Map`, y la respuesta del *router* ante tal comando es la siguiente:

```
MiRouter>show policy-map | include Policy Map
  Policy Map nueva_policymap_2
  Policy Map mapa_politicas
  Policy Map nueva_policymap
  Policy Map mi_politica1
MiRouter>
```

  El procedimiento para obtener los nombres de los mapas de clases de detalla a continuación:

  - Por cada línea seleccionar el texto comprendido entre el último espacio y el final de la línea.

Mediante estos comandos se pueden mandar señales de Control+Z o comandos al *router*. Teniendo en cuenta que:

- Cuando se manda una señal de Control+Z, si no se ha producido ningún error, la salida ante tal comando tendrá una ó dos líneas.
- Los comandos de configuración del *router* (`router rip`, por ejemplo), tienen una salida de dos líneas.
- El comando `configure terminal` tiene una salida de tres líneas.
- El resto de comandos (mostrar configuración, por ejemplo, `show interfaces`) posee un número indeterminado de líneas.

Con los métodos anteriores, podemos comprobar si un comando ha tenido éxito al enviarlo o no mediante el siguiente código:

```
// ------------------------------------------------------------------------------
```

```
// Mandando "Control + Z"
// --------------------------------------------------------------------------------
String inst = "Control + Z";
Vector lineas = ds.writeCtrlZ(this);
if (lineas == null) {
          JOptionPane.showMessageDialog(this, new JLabel("<html>Operación cancelada por
                    el usuario"), "Error", JOptionPane.ERROR_MESSAGE);
          setVisible(false);
          return;
}
if (lineas.size() > 2) {
          // Ha habido un error ------------------------------------------------------
          String exp = Utilidades.Vector2String(lineas, "\n");
          JOptionPane.showMessageDialog(this, new JLabel("<html>Se ha producido un error
                    al enviar las instrucciones al router.<br><br>El mensaje devuelto por
                    el router es éste:<br><pre>" + exp), "Error",
                    JOptionPane.ERROR_MESSAGE);
          setVisible(false);
          return;
}

// --------------------------------------------------------------------------------
// Mandando "show access-list | include access\n";
// --------------------------------------------------------------------------------
inst = "show access-list | include access\n";
lineas = ds.writeStr(inst, this);
if (lineas == null) {
          JOptionPane.showMessageDialog(this, new JLabel("<html>Operación cancelada por
                    el usuario"), "Error", JOptionPane.ERROR_MESSAGE);
          setVisible(false);
          return;
}
if (lineas.size() > 2) {
          // Procesar la respuesta ---------------------------------------------------
}

// --------------------------------------------------------------------------------
// Mandando "configure terminal"
// --------------------------------------------------------------------------------
inst = "configure terminal\n";
lineas = ds.writeStr(inst, this);
if (lineas == null) {
          // Ha habido un error ------------------------------------------------------
          JOptionPane.showMessageDialog(this, new JLabel("<html>Operación cancelada por
                    el usuario"), "Error", JOptionPane.ERROR_MESSAGE);
          setVisible(false);
          return;
}
if (lineas.size() != 3) {
          // Ha habido un error ------------------------------------------------------
          String exp = Utilidades.Vector2String(lineas, "\n");
          JOptionPane.showMessageDialog(this, new JLabel("<html>Se ha producido un error
                    al enviar las instrucciones al router.<br><br>El mensaje devuelto por
                    el router es éste:<br><pre>" + exp), "Error",
                    JOptionPane.ERROR_MESSAGE);
          setVisible(false);
          return;
}

// --------------------------------------------------------------------------------
// Mandando "no access-list " + (String) List_listas.getSelectedValue()
// --------------------------------------------------------------------------------
inst = "no access-list " + s + "\n";
lineas = ds.writeStr(inst, this);
if (lineas == null) {
          // Ha habido un error ------------------------------------------------------
          JOptionPane.showMessageDialog(this, new JLabel("<html>Operación cancelada por
                    el usuario"), "Error", JOptionPane.ERROR_MESSAGE);
          setVisible(false);
          return;
}
if (lineas.size() != 2) {
          // Ha habido un error ------------------------------------------------------
          String exp = Utilidades.Vector2String(lineas, "\n");
          JOptionPane.showMessageDialog(this, new JLabel("<html>Se ha producido un error
                    al enviar las instrucciones al router.<br><br>El mensaje devuelto por
                    el router es éste:<br><pre>" + exp), "Error",
```

```
                JOptionPane.ERROR_MESSAGE);
        setVisible(false);
        return;
}
```

4.4.5 Clase NavegadorAyuda

Esta clase proporciona un visor de archivos HTML, formato en el que se encuentra la ayuda del programa. La clase extiende de `JFrame`, y su constructor no posee parámetros. Los métodos más significativos que posee esta clase son los siguientes:

- `protected void Button_AbrirArchivo_actionPerformed(ActionEvent e)`. Evento ejecutado cuando se pulsa el botón apara abrir un archivo. Muestra un cuadro de diálogo (clase `JFileChooser`) al que se le aplica un filtro para que se muestren sólo los archivos HTML (clase `FileFilterHTML`, descrita más abajo). Una vez que el usuario selecciona el archivo, se muestra en el navegador, llamando al método `cambiarPagina()`.
- `protected void Button_Recargar_actionPerformed(ActionEvent e)`. Método ejecutado cuando se pulsa el botón de recargar página. Hace una llamada al método `cambiarPagina()`, pasando como parámetro la página actual (variable `actual`), y estableciendo el segundo parámetro a `false`, indicando que no se debe añadir la nueva página al historial.
- `protected void cambiarPagina(String url)`. Hace una llamada al método `cambiarPagina()`, pasando como parámetro la página deseada, y estableciendo el segundo parámetro a `true`, indicando que se debe añadir la nueva página al historial.
- `protected void cambiarPagina(String url, boolean cambiar)`. Este método se encarga de cambiar la página mostrada en la ventana. Primero comprueba si la variable `url` que se ha pasado como parámetro corresponde a un archivo, en cuyo caso se mostrará esa página. Si no, se comprobará si la dirección corresponde a un archivo local, y si no lo es, se supondrá que es una dirección de Internet. Tras ello se cargará la página, mostrando una excepción si no se ha podido realizar tal operación. Por último, si la variable `cambiar` está a `true`, se deberá añadir la página que antes se mostraba al historial de páginas anteriores.
- `protected void EditorPane_Contenido_hyperlinkUpdate(HyperlinkEvent e)`. Este método escucha los eventos que se general en el `JEditorPane` que muestra la página. En este caso, cuando se haga clic en un enlace se ejecutará este método, que llamará al método `cambiarPagina()`.
- `protected void Button_Atras_actionPerformed(ActionEvent e)`. Muestra la página anterior, cuya dirección está guardada en la variable de tipo `Vector atras`.
- `protected void Button_Adelante_actionPerformed(ActionEvent e)`. Muestra la página siguiente, cuya dirección está guardada en la variable de tipo `Vector adelante`.
- `protected void Button_AbrirDireccion_actionPerformed(ActionEvent e)`. Método que se ejecuta cuando se pulsa el botón para abrir una dirección. Tras la introducción por parte del usuario de la dirección, se cambiará la página mediante el método `cambiarPagina()`.

La clase posee un objeto de la misma clase. Esto hace que se puedan usar métodos estáticos para mostrar páginas sin necesidad de crear implícitamente un objeto de esa clase. Los métodos estáticos implementados son:

- `public static void mostrar(String url)`. Comprueba si el objeto estático de la clase ha sido creado, y en caso de que no, lo crea. Tras ello muestra la página que se pasa como parámetro y hace la ventana visible.
- `public static void ocultar()`. Si el objeto estático de la clase ha sido creado anteriormente, hace que la ventana no sea visible.

4.4.5.1 Clase FileFilterHTML

Esta clase extiende a la clase `FileFilter` del paquete `javax.swing.filechooser`. Sobrescribe dos métodos:

- `public boolean accept(File f)`. Devuelve `true` si el archive pasado como parámetro es un directorio o un archivo cuya extensión es *htm* ó *html*.
- `public String getDescription()`. Devuelve una cadena con la descripción de los archivos HTML

4.4.6 Clase Opciones

Esta clase permite cambiar las opciones del programa referentes al puerto serie, a la interfaz gráfica, opciones de automatización (guardado de *logs* y eventos) y opciones varias más enfocadas a la depuración del programa. La clase extiende de `SemiModalDialog`. El único constructor de la clase tiene los mismos parámetros que los de un diálogo (`JDialog`), además de un objeto de la clase `Ventana_Principal` (para acceso a los métodos `inicializarPuerto()` y `cerrarPuerto()`), un objeto de la clase `SerialPort` (para acceso a las propiedades del puerto), un objeto de la clase `DobleSalida` (para comunicación con el *router*) y por último un objeto de la clase `EscuchadorEventos`. Posee además los siguientes métodos:

- `private void inicializar()`. Añade las posibles opciones a cada uno de los menús desplegables, así como asigna los valores a los `JSpinner`, para seleccionar los tiempos de espera, así como seleccionar en cada uno de ellos los valores actualmente configurados.
- `protected void CheckBox_vista_previa_itemStateChanged(ItemEvent e)`. Este método se ejecuta cuando la casilla de verificación para cambiar de estilo de ventanas cambia de estado. Hace que la ventana actual cambie de estilo conforme a la selección del menú desplegable.
- `protected void ComboBox_aspecto_grafico_actionPerformed(ActionEvent e)`. Este método hace exactamente lo mismo que el método descrito anteriormente, mediante una llamada al método anterior.
- `protected void Button_Cancelar_actionPerformed(ActionEvent e)`. Método ejecutado cuando se pulsa el botón de cancelar. Simplemente oculta la ventana.
- `protected void Button_Aceptar_actionPerformed(ActionEvent e)`. Guarda las opciones seleccionadas en fichero de opciones mediante llamadas a los métodos de la clase Propiedades. Además cambia las opciones del puerto serie, preguntando al usuario en caso de que no se encuentre en modo administrador, si quiere que se cambien los datos del puerto local sin cambiar los del *router* (en este caso, la comunicación no sería posible tras establecer las opciones en el puerto local y no cambiarlas en el puerto del *router*). Además, si se ha seleccionado un puerto distinto del usado actualmente, se le preguntará si se quiere cerrar el puerto usado para usar el puerto seleccionado, o si se quiere seguir usando el puerto anterior.
- `protected void cambiarDatosPuerto()`. Hace las llamadas necesarias a los métodos de la clase `SerialPort` para establecer las opciones seleccionadas.

- `private void enviar(String s1)`. Este método envía al *router* el comando especificado dentro de la configuración de la línea de consola (`line console 0`), avisando previamente si no se encuentra en modo administrador.
- `protected void Button_Ayuda_actionPerformed(ActionEvent e)`. Muestra la página correspondiente de la ayuda en el navegador cuando se pincha el botón de ayuda.

4.4.7 Clase Propiedades

La clase `Propiedades` maneja las propiedades asociadas al programa. De esta forma, todas las opciones son guardadas en un fichero conforme se van modificando, y cuando se inicia el programa se leen. Se tienen dos métodos por cada variable usada: un método `getXXX()` que devuelve el valor correspondiente a la variable solicitada; y el método `setXXX()`, que establece el valor de la variable al valor que se le pasa como parámetro. Además posee un método `cargar()`, para que al crear un objeto de esta clase se lea el archivo de propiedades, y otro método, `guardar()`, que es llamado cada vez que se modifica una variable para que se guarde en el archivo de propiedades. Cada uno de los dos métodos anteriores hace las conversiones necesarias entre `String` y el valor devuelto o aceptado como parámetro, ya que la clase `Properties` (descrita a continuación) sólo acepta `String` como parámetro.

Esta clase se basa en la clase incluida en la API de Java llamada `Properties`, dentro del paquete `java.util`.

Así mismo, el constructor de la clase tan solo tiene un parámetro, correspondiente a una cadena de caracteres que determina el archivo de propiedades que se va a usar.

Las propiedades del programa que gestiona este programa son las siguientes:

- AutoCargadoEventos. Indica si al inicio del programa se leerá el archivo de eventos y se añadirán a la tabla de eventos situada en la quinta pestaña.
- AutoCargadoLogs. Indica si al inicio del programa se leerá el archivo de *log* y se añadirá su contenido a la ventana de terminal (cuarta pestaña).
- AutoGuardadoEventos. Indica si al cerrar el programa se guardarán los eventos en un archivo.
- AutoGuardadoLogs. Indica si al cerrar el programa se guardará el contenido de la ventana de terminal en un archivo.
- AutoGuardadoAppend. Indica, si alguna de las dos variables anteriores es verdadera, si se añadirá el contenido de la ventana de terminal o de eventos al contenido anterior del fichero, o si se sustituirá.
- BitsDatos. Indica el número de bits de datos que se utilizará como parámetro para el puerto serie.
- BitsParada. Indica el número de bits de *stop* que se utilizará como parámetro para el puerto serie.
- ControlFlujo. Indica el control de flujo que se utilizará como parámetro para el puerto serie.
- Estilo. Indica el estilo de las ventanas que se utilizará en el programa (Windows, Metal o CDE/Motif).
- MostrarEvetosDobleSalida. Indica si los eventos generados por la clase `DobleSalida` serán mostrados en la ventana de eventos.
- MostrarEventosSalidaEstandar. Indica si los eventos generados por el programa se mostrarán también por la salida estándar aparte de en la ventana de eventos.

- MostrarVentanaTerminalIdependiente. Indica si la clase `DobleSalida` debe mostrar una ventana independiente del terminal, aparte de la que posee la ventana principal.
- Paridad. Indica el tipo de paridad que se utilizará como parámetro para el puerto serie.
- Puerto. Indica el puerto serie que se abrirá al iniciar el programa.
- SleepTime. Indica el tiempo que transcurrirá desde que se recibe el primer dato tras un comando, hasta que se comenzará a leer la respuesta completa.
- TimeOutIO. Indica el puerto que se esperará al inicializar el puerto para que se reciba respuesta tras los comandos de inicialización, Si el *router* no da respuesta en un tiempo menor al especificado por esta variable, se considerará que no hay ningún dispositivo conectado al puerto serie.
- TimeOutPuerto. Indica el tiempo de espera de apertura del puerto. Esto hace posible que si otra aplicación ya está usando el puerto, pueda cerrarlo y ceder la posesión a esta aplicación.
- Velocidad. Indica la velocidad del puerto serie que se usará.

4.4.8 Clase Timer

La clase `Timer` creada específicamente para esta aplicación, es necesaria por la clase `Ventana_Prinicipal` para establecer el tiempo de espera para la inicialización del puerto serie. La clase implementa la interfaz `Runnable`, de modo que cuando se ejecute, lo hará en un hilo independiente del programa principal. El constructor del método posee dos parámetros: el primero de ellos indica el tiempo en milisegundos que se deberá esperar desde que se inicia el `Timer`, hasta que se avise a la clase que escucha los eventos; el segundo parámetro es un objeto cuya clase implemente la interfaz `EscuchadorEventosTimer`, y a cuyo método `finTimer()` se llamará cuando expire el `Timer`.

Posee los siguientes métodos:

- `public void start()`. Crea un nuevo objeto `Thread`, y lo arranca mediante el método `start()`.
- `public void stop()`. Hace que la variable de instancia `t` (objeto de la clase `Timer`) se ponga a `null`, de tal forma que la comparación dentro del método `run()` sea falsa y el hilo finalice.
- `public void run()`. Este método se encarga de gestionar el nuevo hilo cuando se inicia. Primeramente obtiene el `Thread` que actualmente se está ejecutando. A continuación obtiene la hora en milisegundos actual (`t_actua`), y tras ello la hora en milisegundos a la que debe finalizar el `Timer` (`t_final = t_actua + tiempo`). A continuación crea un bucle infinito, dentro del cual comprueba si el `Thread` actual es el mismo que el `Thread` que se inició (variable de instancia `t`, si no son iguales, es porque se ha llamado al método `stop()`). Si la condición es verdadera, obtiene de nuevo la hora actual y comprueba si es mayor o igual que la hora de fin del `Timer`, en cuyo caso llamará al método `finTimer()` de la clase `EscuchadorEventosTimer` que se le pasó como parámetro. Si el `Timer` no ha expirado todavía, se espera 10 milisegundos mediante el método `Thread.sleep()`. Debido a este tiempo de espera, este `Timer` tiene un margen de error de ± 10 milisegundos, despreciable para el fin que tiene este programa.
- `public boolean isRunning()`. Devuelve `true` si aún se está ejecutando el método `run()`.

4.4.9 Clase Utilidades

La clase Utilidades ofrece métodos útiles en el programa, referentes a cadenas de caracteres, formatos de direcciones IP, vectores, etc. Todos sus métodos son estáticos, por lo que no es necesario crear una instancia de esta clase, así que no tiene constructor definido implícitamente.

Los métodos definidos en el archivo son:

- `public static boolean esMensaje(String s)`. El *router*, al cambiar la configuración, cambiar es estado de una interfaz, u otros eventos, muestra por pantalla mensajes de información que no son solicitados por el usuario. Este método recibe como argumento una cadena de caracteres y los compara con unos patrones definidos internamente. Si la cadena contiene alguno de esos patrones, la línea que se pasa como parámetro no tendrá significado alguno en cuanto a la interpretación de los datos devueltos por el *router*.

- `public static String quitaEspaciosMultiples(String s)`. Este método recibe como argumento una cadena de caracteres. A esa cadena se le aplica el método `split()` de la clase `String`, pasándole como parámetro una expresión regular (" +"). Esta expresión indica que divida la cadena cuando se encuentre uno o más espacios en blanco. EL método `split()` devuelve un *array* de `String` con cada una de las piezas en las que se ha dividido. A continuación se unen todas incluyendo un único espacio en blanco y se devuelve la nueva cadena.

- `public static String Vector2String(Vector v, String une)`. Este método une cada uno de los elementos del vector que se le pasa como parámetro (realizando un *casting* a `String`) intercalando entre cada uno de ellos el segundo parámetro.

- `public static String Vector2String(Vector v)`. Este método únicamente hace una llamada al método anterior, usando como segundo parámetro una cadena en blanco ("").

- `public static boolean esIPValida(String s)`. Comprueba si la cadena que se le pasa como parámetro es una IP válida. En primer lugar comprueba si la cadena contiene 4 partes separadas por un punto. Tras ello, comprueba que cada una de las partes sea un entero válido y que además esté comprendido entre 0 y 255 inclusive ambos.

- `public static int getPosicionDe(String s1[], String s2)`. Obtiene la posición de la cadena que se le pasa como segundo parámetro en el *array* de cadenas que se usa como primer parámetro. Simplemente recorre el *array*, comparando el elemento correspondiente con el segundo parámetro mediante el método `compareToIgnoreCase()` de la clase `String`. Si es encontrado, se devuelve el índice del *array* correspondiente. Si se ha recorrido todo el *array* y no se ha encontrado la cadena, el valor de retorno es -1.

- `public static String[] eliminaPosicion(String s1[], int pos)`. Elimina la posición `pos` del *array* `s1`. Para ello convierte el *array* en un `Vector`; llama después al método `removeElementAt()` de la clase `Vector`, y para finalizar convierte el nuevo `Vector` de nuevo a un *array* de `String`.

- `public static boolean esEntero(String s)`. Indica si la cadena que se pasa como parámetro corresponde a un número entero válido; es decir, si el método `new Integer(s)` laza una excepción o no.

4.5 Paquete AccessLists

Este paquete comprende las clases necesarias para manejar las listas de control de acceso (ACL), tanto IP estándar, como IP extendidas, UDP y TCP.

4.5.1 Clase AccessLists

La clase `AccessLists` representa una ventana (extiende de `SemiModalDialog`) que lista las listas de acceso disponibles en el *router* y da las opciones de añadir, editar y eliminar dichas listas de acceso. Su constructor posee los mismos parámetros que los de un diálogo normal, así como un objeto de la clase `DobleSalida` y otro de la clase `EscuchadorEventos` para los mismos fines ya descritos en otras clases.

Además, posee los siguientes métodos:

- `private void inicializarCosas()`. Rellena la lista con las listas de acceso del *router*. Mediante el comando `show access-list | include access` se obtienen las lista de acceso en el siguiente formato:

  ```
  Standard IP access list 1
  Standard IP access list 2
  Extended IP access list 101
  ```

  La forma de obtener el número de la lista de acceso se realiza obteniendo la cadena que se encuentra entre el último espacio y el final de la línea.

- `protected void Button_anadir_actionPerformed(ActionEvent e)`. Método ejecutado cuando se pulsa el botón para añadir una nueva lista de acceso. Muestra una nueva ventana (instancia de `JOptionPane`) con una lista con los números disponibles para asignar las listas de acceso (entre 1 y 199), eliminando aquellos números que ya están siendo utilizados por otras listas de acceso. Una vez seleccionada la nueva lista por el usuario, se crea una nueva instancia de la clase `Lista`, pasando, entre otros parámetros, el número seleccionado.

- `protected void Button_editar_actionPerformed(ActionEvent e)`. Crea una nueva instancia de la clase `Lista` a partir del número de la lista seleccionada.

- `protected void Button_eliminar_actionPerformed(ActionEvent e)`. Tras una confirmación, elimina la lista seleccionada enviando al *router* el comando `no access-list` seguido del número de la lista seleccionada.

4.5.2 Clase Lista

Esta clase muestra la lista de condiciones que componen la lista de acceso seleccionada. Extiende, como la clase anterior, a la clase `SemiModalDialog`. Su parámetro recibe los mismos argumentos que la clase `JDialog`, aparte de un objeto de la clase `DobleSalida`, otro de la clase `EscuchadorEventos` y el número de la lista de acceso a mostrar.

Los métodos más importantes que posee esta clase son los siguientes:

- `private void inicializarCosas()`. Rellena la lista con las condiciones asociadas a la lista de acceso. Mediante el comando `show access-list` seguido del número de la lista se obtiene la siguiente salida.

  ```
  Standard IP access list 2
      deny   192.168.2.54 log
  ```

```
deny   192.168.5.0, wildcard bits 0.0.0.255
```

Por tanto, obtendremos las condiciones de la lista eliminando la primera línea de la respuesta al comando anterior.

- `protected void Button_Anadir_actionPerformed(ActionEvent e)`. Muestra una ventana para añadir una nueva condición a la lista de acceso. Según el número de la lista de acceso se creara una instancia de una u otra clase. Si el número está comprendido entre 1 y 99, se creará una nueva instancia de la clase `Lista_Estandar`. Si el número está comprendido entre 100 y 199, se le preguntará al usuario el tipo condición de lista extendida que quiere usar: IP, TCP o UDP; de acuerdo con su selección se creará una instancia de las clases `Lista_Extendida_IP`, `Lista_Extendida_TCP` o `Lista_Extendida_UDP`, respectivamente. Tras ello se mostrará la ventana, y cuando se pulse el botón de aceptar se obtendrá la condición seleccionada, que se enviará al *router*.

- `protected void Button_Editar_actionPerformed(ActionEvent e)`. Método que se ejecutará cuando se pulse el botón para editar una condición. Si el número de la lista está comprendido entre 1 y 99 se creará una nueva instancia de la clase `Lista_Estandar`, pasando como parámetro al constructor, entre otros, la condición a editar. Si por el contrario el número de la lista de acceso está entre 100 y 199, se comprobará si la segunda palabra de la condición contiene las cadenas IP, TCP o UDP, creando respectivamente instancias de las clases `Lista_Extendida_IP`, `Lista_Extendida_TCP` o `Lista_Extendida_UDP`. Tras que el usuario pulse el botón de aceptar se obtendrá la nueva condición.

  El proceso para editar la condición será:

  1. Eliminar la lista de acceso con el comando `no` seguido del número de la lista.
  2. Dentro del vector, variable de instancia de la clase, que contiene las condiciones de la lista, eliminar la condición seleccionada.
  3. Insertar en la posición eliminada la nueva condición.
  4. Crear de nuevo la lista con el comando `access-list` seguido del número de la lista.
  5. Procesar cada una de las condiciones creando una nueva instancia de la clase correspondiente y llamando al método `hazClick()` para después obtener el comando que se debe enviar al *router* mediante el método `getCondicion()`.
  6. Enviar las condiciones ya procesadas al *router* mediante el comando `access-list` seguido del número de la lista de acceso y de la condición.

- `protected void Button_Eliminar_actionPerformed(ActionEvent e)`. Tras una confirmación por parte del usuario, elimina la lista de acceso seleccionada mediante el comando `no` seguido del número de la lista de acceso.

4.5.3 Interfaz Interfaz

Esta interfaz contiene métodos usados comunes a todas las clases que editan listas (`Lista_Estandar`, `Lista_Extendida_IP`, `Lista_Extendida_UDP` y `Lista_Extendida_TCP`) por la clase Lista. Estos métodos son los siguientes:

- `String getCondicion()`. Una vez que se ha pulsado el botón de aceptar y se ha generado la cadena correspondiente, mediante este método se obtiene dicha cadena, o `null` si el usuario ha cerrado la ventana sin terminar la especificación de la lista.

- `void hazClick()`. La forma en que el *router* muestra algunas condiciones de la lista y la forma en la que se deben enviar son distintas. Por eso, es necesario poder

procesar las instrucciones devueltas por el *router* al mostrar la configuración al modo en que se deben enviar sin tener que mostrar la ventana y sin intervención por parte del usuario. Este método simula la pulsación del botón de aceptar, para que se genere el comando correspondiente, que después será obtenido mediante el método `getCondicion()`.

- `void setVisible(boolean vis)`. Permite que todas las clases posean un método para ocultar o mostrar la ventana.

4.5.4 Clase Lista_Estandar

Muestra las opciones necesarias para generar una condición para una lista estándar IP:

- Acción a cumplir: permitir o denegar
- Dirección: cualquier dirección, dirección concreta de un *host* o dirección de una red, con posibilidad de especificar la máscara.
- Hacer un *log*.

El constructor de la clase, que hereda de `SemiModalDialog`, admite los mismos parámetros que un diálogo además de la condición que se quiere editar.

Los métodos más importantes implementados en esta clase son los siguientes:

- `public void hazClick()`. Especificado por la interfaz `Interfaz`. Hace una llamada al método ejecutado cuando se pulsa el botón de aceptar.
- `private void editar(String s)`. Si lo que se quiere es editar una condición, el constructor llamará a este método pasando como parámetro la condición que se quiere editar. Este método selecciona en la interfaz gráfica aquellos elementos acorde con la condición, y en su caso, rellena los huecos de las direcciones IP y de la máscara con sus correspondientes valores.
- `public String getCondicion()`. Una vez que se ha pulsado el botón de aceptar, la clase `Lista` llamará a este método para obtener el comando generado a partir de las opciones seleccionadas por el usuario.
- `protected void Button_Aceptar_actionPerformed(ActionEvent e)`. Construye el comando que se debería enviar al *router* de acuerdo con las opciones seleccionadas en la interfaz gráfica, mostrando un mensaje de error si falta alguna opción o alguna es incorrecta.

Además de los métodos descritos, existen otros métodos relacionados con componentes de la interfaz gráfica que según estén seleccionados o no, hacen que otros elementos relacionados con ellos puedan ser seleccionados.

4.5.5 Clase Lista_Extendida_IP

Esta clase es similar a la anterior, añadiendo a la interfaz gráfica opciones tales como:

- Direcciones origen y de destino.
- *Logs* extendidos.
- Opciones de valores de coincidencia relacionados con servicios diferenciados: IP DSCP, fragments, IP Precedence, Time-Range y TOS.

4.5.6 Clase Lista_Extendida_TCP_Puertos

La clase `Lista_Extendida_TCP_Puertos` permite generar cadenas de coincidencia de puertos para añadir a las condiciones de las listas de control de acceso. La clase extiende a `SemiModalDialog` y en su constructor acepta los mismos parámetros que un diálogo normal, aparte de una cadena correspondiente a la condición que se quiere editar; si no se quiere editar ninguna cadena, sino que se quiere añadir una, este parámetro estará a `null`.

La interfaz permite seleccionar el tipo de comparación: igual, mayor que, menor que y distinto; y el puerto bien sea por número o por nombre.

Los métodos más importantes que posee esta clase son los descritos a continuación:

- `private void inicializarCosas()`. Añade los posibles valores a los menús desplegables (condiciones, números de puertos, nombres de puertos).
- `private void editar(String condicion)`. En caso de que se quiera editar una condición de puertos, este método selecciona en la interfaz gráfica las opciones de acuerdo con la condición que se le pasa como parámetro.
- `public String getCondicion()`. Devuelve la condición generada de acuerdo con las opciones seleccionadas en la interfaz gráfica.
- `protected void Button_Aceptar_actionPerformed(ActionEvent e)`. Genera la condición que se añadirá a la condición de la lista de control de acceso de acuerdo con las opciones seleccionadas en la interfaz gráfica.

4.5.7 Clase Lista_Extendida_TCP

Esta clase es similar a la clase `Lista_Extendida_IP`, añadiendo a la interfaz gráfica opciones de servicios diferenciados tales como:

- Ack
- Established
- Fin
- PSH
- Range (rango de puertos)
- Rst
- Syn
- Urg
- Lista de coincidencias con puertos.

4.5.8 Clase Lista_Extendida_UDP_Puertos

Esta clase es similar a la clase `Lista_Extendida_TCP_Puertos`, cambiando los puertos asociados a TCP por los puertos usados en UDP.

4.5.9 Clase Lista_Extendida_UDP

Esta clase es similar a la clase `Lista_Extendida_TCP`, teniendo como valores para coincidencia las siguientes opciones:

- IP DSCP

- Fragments
- IP Precedence
- Range (rango de puertos)
- Time-Range
- TOS
- Lista de coincidencias con puertos.

4.6 Paquete cisco.AsocPolicyInterfaz

Este paquete contiene una única clase, `ConfigurarAsociacionPolicyMap-Interfaz`, encargada de asociar los mapas de políticas creados a las interfaces del *router*, de tal forma que aplique dichas políticas a los paquetes que entren o salgan.

4.6.1 Clase ConfigurarAsociacionPolicyMapInterfaz

Esta clase muestra un menú desplegable con la lista de interfaces disponibles en el *router*, y dos opciones de selección, una de entrada y otra de salida, con todos los mapas de políticas creados. La clase muestra una ventana, y como la mayoría de las clases, extiende de SemiModalDialog. Su único constructor acepta los mismos parámetros que un JDialog además de un objeto DobleSalida para escribir y leer datos por el puerto serie y un objeto EscuchadorEventos para agregar los eventos que sucedan internamente en la clase al programa.

Los métodos que posee esta clase son similares a los de otras:

- private void inicializarCosas(). Rellena las listas de interfaces y mapas de políticas con las que contiene el *router*.
- protected void ComboBox_interfaz_actionPerformed(ActionEvent e). Una vez que se ha seleccionado una interfaz, se deben mostrar las políticas de entrada y salida asociadas a dicha interfaz, si las tuviera. Dichas políticas se obtienen con los siguientes comandos:

  ```
  MiRouter>show policy-map interface ethernet 0/0 | include Service-policy input
  ```

  ```
  MiRouter>show policy-map interface ethernet 0/0 | include Service-policy output
  ```

  ```
    Service-policy output: nueva_policymap (1105)
  ```

  ```
  MiRouter>
  ```

  Como se puede observar, si no hay ninguna política asociada, el *router* no da ninguna salida, mientras que si hay alguna política asociada devuelve una cadena que contiene dicha política. La política es extraída de la respuesta obteniendo el texto encerrado entre los dos puntos y el espacio (": ") y el siguiente espacio.

  Una vez que se han obtenido las políticas, se seleccionará dicha política en la lista de políticas bien de entrada o de salida, recorriendo todos los elementos del menú desplegable y comprobando si son iguales.

- `protected void Button_Aceptar_actionPerformed(ActionEvent e).` Este método genera el comando que se debe enviar al *router* para establecer, cambiar o eliminar los mapas de políticas asociados a la interfaz de acuerdo con las opciones del usuario. El siguiente diagrama de flujo muestra el protocolo seguido para dicha operación (en este caso, para la política de entrada; un proceso análogo se realizará para la política de salida):

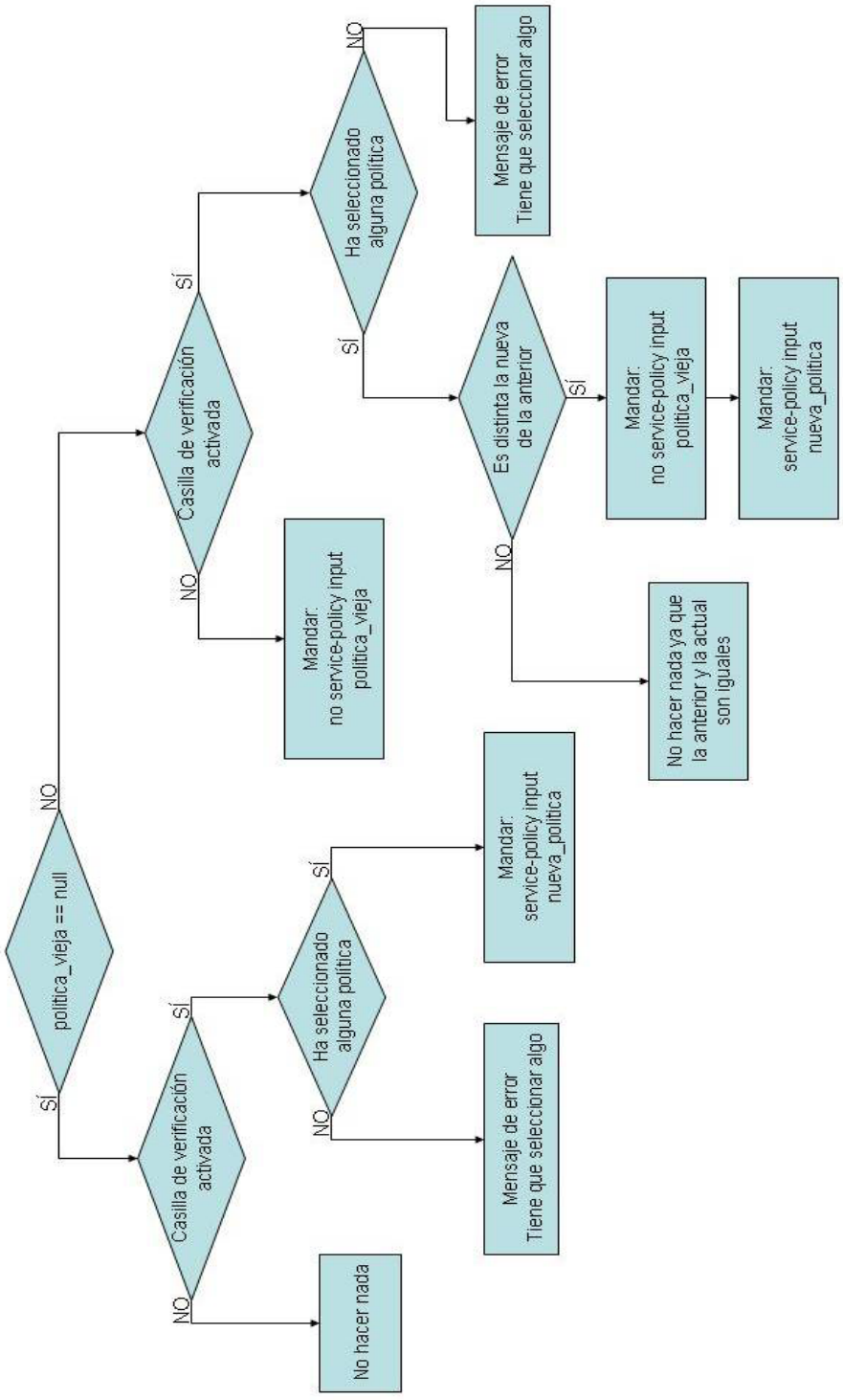


Figura 12 - Diagrama de flujo para asociar mapas de políticas a interfaces

4.7 Paquete cisco.ClassMaps

Este paquete contiene dos clases para manejar, editar, añadir y eliminar los mapas de clases configurados en el *router*. La primera de ellas que se describirá a continuación muestra los mapas de clases que se encuentran actualmente configurados en el *router*, con la opción

de añadir más mapas, editar los ya existentes, y eliminarlos, así como realizar las mismas operaciones con sus condiciones. La segunda clase descrita es una interfaz gráfica que permite generar las condiciones asociadas a los mapas de políticas de acuerdo don criterios como: que pertenezca a otro mapa de clases, a un grupo de acceso, a un valor de clase de servicio, etc.

4.7.1 Clase ClassMap

La clase `ClassMap` muestra un menú desplegable con los mapas de clases configurados en el *router*. Para cada uno de estos mapas, es posible eliminarlos, renombrarlos o cambiar el modo en que se cumplen las condiciones. Así mismo, también es posible añadir nuevos mapas de clases. Una vez se ha seleccionado un mapa de clases, en la lista inferior se muestran las condiciones asociadas a dicho mapa de clases.

La clase extiende de `SemiModalDialog`, teniendo su constructor como parámetros los mismos que los de un diálogo, además de un objeto de la clase `DobleSalida` y otro de la clase `EscuchadorEventos`.

Los métodos más importantes de esta clase se describen a continuación:

- `private void inicializarCosas()`. Rellena el menú desplegable con los mapas de clases disponibles en el *router*, obtenidos con el comando `getClassMaps()` de la clase `DobleSalida`.
- `private void actualizaCondiciones()`. Método ejecutado cuando se selecciona una nueva clase en el menú desplegable, mediante el comando show `class-map` seguido del nombre de la clase. El formato de salida de este comando es el siguiente:

  ```
  MiRouter>show class-map nueva_clase
   Class Map match-all nueva_clase (id 4)
     Match ip  rtp 2000 50
     Match ip  dscp 20  24  48


  MiRouter>
  ```

  Así, la lista inferior se rellenará con las líneas de la salida, eliminando las dos primeras y las dos últimas. Con la segunda línea se puede obtener el modo de cumplimiento de las clases (`match-any` o `match-all`).
- `protected void ComboBox_Lista_Clases_actionPerformed(ActionEvent e)`. Método ejecutado cuando se selecciona una nueva clase en el menú desplegable. Simplemente llama al método `actualizaCondiciones()`.
- `protected void Button_Anadir_Clase_actionPerformed(ActionEvent e)`. Solicita al usuario el nombre de la nueva clase, y la crea en el *router* mediante el comando `class-map` seguido del nombre de la clase a crear.
- `protected void Button_Eliminar_Clase_actionPerformed(ActionEvent e)`. Tras una confirmación para el usuario, elimina la clase seleccionada mediante el comando `no class-map` seguido de la clase seleccionada.
- `protected void Button_Renombrar_actionPerformed(ActionEvent e)`. Renombra la clase seleccionada con el nombre escrito en la caja de texto. En primer lugar se comprueba si el usuario ha escrito algo en la caja de texto. A continuación, mediante el comando `rename` seguido del nuevo nombre, dentro de la configuración del mapa de clases, la clase es renombrada.

- `protected void Button_Cumplir_actionPerformed(ActionEvent e)`. Cambia el modo en que se deben cumplir las condiciones de la lista de acceso. Para ello, manda el comando `class-map`, seguido del modo de cumplimiento de las condiciones (`match-any` o `match-all`) y tras ello el nombre de la clase.

- `protected void Button_Anadir_Condicion_actionPerformed( ActionEvent e)`. Crea una nueva instancia de la clase `ClassMap_Condicion`. Después de que el usuario halla pulsado el botón de aceptar, se obtiene la condición seleccionada y se envía al *router* estando dentro de la configuración de la clase:

  ```
  MiRouter#configure terminal
  Enter configuration commands, one per line.  End with CNTL/Z.
  MiRouter(config)#class-map nueva_clase
  MiRouter(config-cmap)#match any
  MiRouter(config-cmap)#
  ```

- `protected void Button_Editar_Condicion_actionPerformed( ActionEvent e)`. Crea una nueva instancia de la clase `ClassMap_Condicion`, pasando como uno de los parámetros la condición a editar. Después de que el usuario halla pulsado el botón de aceptar, se elimina la condición seleccionada y se manda la nueva condición:

  ```
  MiRouter#configure terminal
  Enter configuration commands, one per line.  End with CNTL/Z.
  MiRouter(config)#class-map nueva_clase
  MiRouter(config-cmap)#no match any
  MiRouter(config-cmap)#match ip rtp 6768 200
  ```

  Tras ello, se actualiza la interfaz gráfica con los nuevos valores llamando al método `actualizaCondiciones()`.

- `protected void Button_Eliminar_Condicion_actionPerformed( ActionEvent e)`. Tras una confirmación elimina la condición seleccionada mediante el comando `no` seguido de la condición, dentro de la configuración del mapa de clases.

4.7.2 Clase ClassMap_Condicion

Esta clase muestra en una ventana todas las opciones que se pueden seleccionar para añadir una condición a un mapa de clases. La clase extiende de `SemiModalDialog` (explicado en el paquete `cisco.Dialogos`). Su constructor admite los mismos parámetros que la clase explicada anteriormente, además de dos cadenas de caracteres que identifican la clase (necesaria para la opción `match class-map` en la que se pueden seleccionar todos los mapas de clases excepto el que se está configurando) y la condición que se quiere editar (estando esta última variable a `null` en caso de querer añadir una nueva condición).

Los métodos implementados en esta clase son los siguientes:

- `private void editar (String condicion)`. Si se ha elegido la opción de editar una condición, este método seleccionará en la interfaz gráfica las opciones acorde con la condición que se pasa como parámetro.

- `private void inicializarCosas()`. Rellena los menús desplegables con los valores de IP DSCP, IP Precedence, interfaces, mapas de clases, protocolos, etc.

- `private void resetear()`. Resetea toda la interfaz gráfica de forma tal que no haya ninguna opción seleccionada.

- `private boolean esvalidohex(String s)`. Comprueba si la cadena que se pasa como parámetro es una cadena válida para las direcciones MAC.
- `public String getCondicion()`. Devuelve el comando que se debería enviar al *router* de acuerdo con las opciones seleccionadas en la interfaz gráfica.
- `protected void Button_Aceptar_actionPerformed(ActionEvent e)`. Construye el comando que se debería enviar al *router* de acuerdo con las opciones seleccionadas en la interfaz gráfica. Si alguna opción es incorrecta o falta algún parámetro, el usuario es avisado mediante un mensaje.

4.8 Paquete cisco.Dialogos

Este paquete contiene los cuadros de diálogo que se usan para informar al usuario de situaciones excepcionales del programa tales como excepciones que se producen durante la ejecución de la aplicación, carga del programa, comunicación con el *router*, etc.

4.8.1 Clase ComunicandoConRouter

Esta clase, que hereda de `SemiModalDialog`, muestra una ventana avisando al usuario indicándole que se están mandando o recibiendo datos. La clase es usada por los métodos de lectura y escritura de la clase `DobleSalida`. Su constructor recibe como parámetros los mismos que recibiría un diálogo normal, además de un posible mensaje en particular especificando la causa que se muestra en la ventana (si se desea utilizar el texto por defecto, esta variable valdrá `null`), y un objeto que implemente la interfaz `CanelarComunicacion`, a cuyo método `cancelarComunicacion()` se llamará si el usuario pulsa el botón para cancelar la comunicación. La clase posee dos constructores distintos, bien sea su creador un `JFrame` o un `JDialog`. En ambos constructores se llamará al método `crear()`, que llama a su vez al método `jbInit()` (para inicializar la interfaz gráfica) y centra el diálogo en la pantalla.

El único método importante del cual se describirá su función es el método `protected void Button_Cancelar_actionPerformed(ActionEvent e)`. Este método llama al método `cancelarComunicacion()` de la clase que implementa la interfaz `CancelarComunicacion` que se pasó como parámetro al constructor. Tras ello, oculta este diálogo.

4.8.2 Clase InformarExcepcion

Esta clase muestra un cuadro de diálogo con información detallada cuando se lanza una excepción. Posee dos pestañas. En la primera de ellas se muestra información general de la misma; en la segunda pestaña se muestra la pila de clases y métodos por los cuales se ha propagado la excepción, indicando el archivo fuente y el número de línea.

La clase extiende de `JDialog`, y cuando se muestra esta ventana no es posible acceder a ninguna de las otras ventanas del programa (navegador de la ayuda y terminal auxiliar), debido a los problemas generados cuando los diálogos son modales (ver apartado 4.8.4 sobre diálogos semimodales). Al igual que la clase anterior, esta clase posee dos constructores según sea un `JDialog` o un `JFrame` quien cree la ventana. Estos dos métodos llaman en primer lugar al constructor de la clase padre, y tras ello al método `crear()`, quien a su vez llama al método `jbInit()` para inicializar la interfaz gráfica, fija el tamaño de la ventana y la centra en la pantalla.

El método `CambiarInfo()` es llamado también por el método `crear()`. Acepta dos parámetros:

- `String s`, indicando una breve descripción de dónde, cómo y por qué se ha producido la excepción.
- `Exception e`, que representa al objeto de la excepción generado. Gracias a él, se podrá obtener la pila de métodos y clases que propagaron esta excepción, y rellenar así el contenido de la segunda pestaña.

4.8.3 Clase MensajeDeEspera

Esta clase muestra una ventana informando al usuario que se está cargando la aplicación, Cuando la ventana principal ha sido creada y es mostrada, esta ventana de aviso es ocultada.

Posee un único constructor que recibe los mismos parámetros que un diálogo, llamando en primera instancia al constructor de la clase padre (`JDialog`). Tras ello, llama al método `jbInit()` para que se inicialice la interfaz gráfica, para a continuación ajustar el tamaño del diálogo a su contenido (método `pack()`) y centrarlo en la pantalla.

Esta clase posee tan solo dos métodos: el constructor descrito anteriormente, y el método `jbInit()` que como ya hemos dicho, inicializa la interfaz gráfica.

4.8.4 Clase SemiModalDialog

Java permite crear básicamente dos tipos de ventanas: los diálogos (`JDialog`) y los marcos (`JFrame`). Los diálogos son una clase especial de ventanas que pueden ser modales o no. En caso de ser modales, hacen que cuando un diálogo sea visible no se pueda acceder a ninguna otra ventana de la aplicación, ya sean otros marcos u otros diálogos. En esta aplicación se tienen principalmente tres marcos: uno para la ventana principal, otro para el navegador de la ayuda y otro para la ventana independiente del terminal. Todas las ventanas que se muestran para la configuración de opciones del *router* y el menú de opciones, son diálogos modales; por ello, cuando se muestra, por ejemplo, la ventana para configurar los mapas de clases, no es posible acceder a la ventana de ayuda, por lo que no se puede obtener ayuda acerca de esta clase.

Una primera solución sería que los diálogos no fueran modales. En el caso particular de esta aplicación no sería recomendable, ya que, por ejemplo, si abrimos el diálogo para configurar los mapas de políticas y estamos configurando una clase, si teniendo abierto ese diálogo abrimos el diálogo para modificar los mapas de clases y eliminamos la clase sobre la que estamos trabajando en el mapa de políticas, se producirán errores en la ejecución del programa.

La otra opción es conseguir que los diálogos sólo bloqueen el acceso a la ventana que los creó. De esta forma, cuando se abre un diálogo, se bloquearía el acceso única y exclusivamente a la ventana principal, permitiendo el acceso al navegador de la ayuda y a la ventana de terminal. Esto se ha conseguido gracias a Internet, donde se ha encontrado la clase `SemiModalDialog`, que hereda de la clase `JDialog`. Cualquier diálogo que quiere tener la propiedad de semimodal, debe heredar de esta clase, obteniendo los resultados deseados descritos anteriormente.

4.9 Paquete cisco.Interfaces

Este paquete contiene la mayoría de las interfaces que se utilizan en la aplicación para el manejo de eventos, utilización de temporizadores, comunicación con el puerto serie, etc.

4.9.1 Interfaz CancelarComunicacion

Este interfaz es implementado por la clase `DobleSalida`, necesaria para que la ventana de aviso que indica que se está comunicando con el *router* (clase `ComunicandoConRouter`) pueda indicar a la clase `DobleSalida` que el usuario quiere cancelar la comunicación. Este interfaz posee un único método:

- `public void cancelarComunicacion()`. Método al cual llama la clase `ComunicandoConRouter` y que debe ser implementado por la clase `DobleSalida`. Es ejecutado cuando el usuario pulsa el botón para cancelar la comunicación.

4.9.2 Interfaz DobleSalida_Escuchador

Este interfaz es implementado por la clase `Ventana_Principal` para recibir los datos que se reciben por el puerto serie. El único método que provee este interfaz, `void recibe(String s)`, es llamado por la clase `DobleSalida` cada vez que se reciben datos por el puerto serie.

4.9.3 Interfaz EscuchadorEventos

Este interfaz es implementado por la clase `Ventana_Principal`. Posee dos métodos:

- `void anadir_evento(String que)`. Este método añade un evento a la tabla de eventos de la ventana principal. Es llamado por todas aquellas clases que generan eventos internos.
- `boolean isAdministrador()`. El método es llamado por aquellas clases que quieren modificar la configuración del *router*. Antes de poder enviar comandos de configuración, la clase debe comprobar si se está en modo administración para poder cambiar la configuración; en caso contrario si se envían comandos, estos darán un mensaje de error.

4.9.4 Interfaz EscuchadorEventosTimer

Este interfaz es utilizado por la clase `Ventana_Principal` y por la clase `Timer`. Gracias a él, es posible que el temporizador avise a la ventana principal cuando expira, de forma que es posible detectar si hay un *router* conectado al puerto serie, si da respuesta antes de un tiempo determinado, tal y como se explica en la clase `Ventana_Principal`. El único método que posee esta interfaz es `void finTimer()`.

4.10 Paquete cisco.IPs

Este método contiene una única clase que gestiona las interfaces y sus direcciones: dirección estática y dinámica, modo de obtener la dirección dinámica, etc.

4.10.1 Clase ConfigurarIPs

Esta clase muestra en la parte superior una lista con todas las interfaces que posee el *router*. Cuando se selecciona una interfaz se seleccionan en la interfaz gráfica las opciones configuradas para esa interfaz.

La clase deriva de la clase `SemiModalDialog`, por las razones explicadas en la sección 4.8.4. Posee un único constructor, que como otras clases, posee los mismos parámetros que un diálogo aparte de un objeto `DobleSalida` y un objeto que implemente la interfaz `EscuchadorEventos`.

Los métodos más importantes implementados en esta clase son los siguientes:

- `private void inicializarCosas()`. Rellena los menús desplegables con las interfaces serie y *Ethernet* que tiene el *router*.
- `private void resetear()`. Resetea los elementos de la interfaz gráfica para que no haya seleccionado nada al iniciar el programa y cuando se cambia de interfaz.
- `protected void ComboBox_Lista_Interfaces_actionPerformed( ActionEvent e)`. Este método es ejecutado cuando se selecciona alguna interfaz en el menú desplegable. Selecciona en la interfaz gráfica las opciones de acuerdo con la configuración del *router* sobre la interfaz seleccionada:
  - Mediante el comando `show interfaces` seguido de la interfaz selecciona se obtiene la siguiente respuesta:

    ```
    Ethernet0/0 is up, line protocol is up
      Hardware is AmdP2, address is 0007.853b.2f00 (bia 0007.853b.2f00)
      Internet address is 192.168.2.254/24
    […]
    ```

    Mediante la primera línea de la salida del comando se puede saber si la interfaz está activada (si antes de la coma está la cadena *up* o *down*), y si la interfaz tiene algún problema (la segunda parte de la cadena contiene *up* o *down*). En el ejemplo anterior, la interfaz está activada y no tiene problemas.
  - Mediante el comando `show interfaces` *`interfaz`* `| begin Internet`, seguido de la interfaz selecciona se obtiene la siguiente respuesta:

    ```
    MiRouter>show interfaces Ethernet 0/0 | begin Internet
      Internet address is 192.168.2.254/24 […]
    ```

    Esta salida indica que la interfaz está configurada mediante una dirección IP estática, indicando la dirección IP y el número de *bits* a 1 de la máscara. Con estos datos se seleccionarán en la interfaz gráfica las opciones correspondientes.

    ```
    MiRouter#show interfaces Ethernet 0/1 | begin Internet
      Internet address will be negotiated using DHCP […]
    ```

    En este caso, la dirección de la interfaz se configurará mediante DHCP.

    Si se obtiene alguna salida con formato distinto al anterior, se mostrará un mensaje indicando que la interfaz está configurada de un modo que el programa no puede manejar.
- `protected void Button_Aceptar_actionPerformed(ActionEvent e)`. El método es ejecutado cuando se pulsa el botón de aceptar de la ventana. Genera la cadena que se debe enviar al *router* para aplicar las opciones seleccionadas por el usuario a la interfaz sobre la que se está trabajando. Además manda el comando `shutdown` (precedido de `no`, si es necesario) si la casilla para activar la interfaz está seleccionada.

4.11 Paquete cisco.PolicyMaps

Este paquete contiene las clases necesarias para trabajar con mapas de políticas: añadir mapas, editarlos, eliminarlos, añadirles clases, editarlas, eliminarlas, y realizar estas mismas operaciones con las políticas asociadas a cada clase.

4.11.1 Clase ConfigurarPolicyMap

La clase `ConfigurarPolicyMap` muestra una ventana con los mapas de políticas en un menú desplegable, botones para añadir o eliminar mapas, una caja de texto y un botón para cambiarles el nombre, y añadir, editar y eliminar clases asociadas a ese mapa de políticas.

La clase hereda de `SemiModalDialog` y su constructor recibe los mismos parámetros que un diálogo, además de un objeto de la clase `DobleSalida` y un objeto cuya clase implemente la interfaz `EscuchadorEventos`.

Cabe destacar los siguientes métodos implementados en la clase:

- `private void inicializar_mapas_politicas()`. Rellena el menú desplegable con los mapas de políticas configurados en el *router*. Usa el método de la clase `DobleSalida getPolicyMaps()`.
- `protected void Button_Renombrar_actionPerformed(ActionEvent e)`. Cambia el nombre al mapa de políticas seleccionado. Para ello usa el comando `rename` dentro de la configuración del mapa de políticas, comprobando previamente que el usuario ha escrito un nombre de política válido en la casilla para tal fin.
- `protected void Button_Anadir_PolicyMap_actionPerformed( ActionEvent e)`. Añade un nuevo mapa de políticas. Para ello solicita al usuario el nombre del nuevo mapa mediante un objeto de la clase `JOptionPane`. Si el usuario ha introducido un nombre válido, se crea el nuevo mapa mediante el comando `policy-map`, seguido del nombre de la nueva política.
- `protected void Button_Eliminar_PolicyMap_actionPerformed( ActionEvent e)`. Elimina el mapa de políticas seleccionado. Para ello envía el comando `no policy-map` seguido de la política seleccionada, dentro de la configuración de terminal.
- `protected void ComboBox_Lista_PolicyMap_actionPerformed( ActionEvent e)`. Método ejecutado cuando se selecciona un nuevo mapa de políticas del menú desplegable. Mediante el comando `show policy-map` *`policymap`* `| include Class`, cuya salida es la siguiente:

  ```
  MiRouter#show policy-map mapa_politicas | include Class
      Class mapa_clases
  MiRouter#
  ```

  Se obtienen las clases asociadas al mapa de políticas seleccionado. Los nombres de los mapas de clases se obtienen eliminado los espacios del inicio de cada línea de la respuesta (método `trim()` de la clase `String`), y tras ello seleccionando el texto desde el primer espacio hasta el final de la línea.
- `protected void Button_Anadir_Clase_actionPerformed(ActionEvent e)`. Añade una nueva clase al mapa de políticas sobre el que se está trabajando. Para ello obtiene una lista con las clases mediante el método `getClassMaps()` de la clase `DobleSalida`, y muestra un mensaje al usuario para que seleccione la clase que desea añadir. Tras ello, añade la clase a la lista inferior.
- `protected void Button_Editar_Clase_actionPerformed(ActionEvent e)`. Crea un nuevo objeto de la clase ConfigurarPolicyMap_Clase, pasándole entre

otros parámetros, el mapa de políticas y la clase sobre los que se están trabajando. Tras ello, muestra la nueva ventana.

- `protected void Button_Eliminar_Clase_actionPerformed( ActionEvent e)`. Elimina la clase del mapa de políticas seleccionado mediante el comando `no`, seguido de la clase seleccionada, dentro de la configuración del mapa de políticas.

4.11.2 Clase ConfigurarPolicyMap_Clase

Esta clase muestra las condiciones asociadas a una clase dentro de un mapa de políticas. Da la opción de añadir, editar y eliminar las políticas mostradas en la lista.

Como la clase anterior, hereda de la clase `SemiModalDialog`, y su constructor posee, además de los mismos parámetros que la anterior clase, dos cadenas que representan el mapa de políticas sobre el que se está trabajando y el mapa de clases que se está editando.

Posee los siguientes métodos:

- `private void inicializarPoliticas()`. Rellena la lista con las condiciones asociadas a la clase que se está editando. Mediante el comando `show policy-map` *`politica`* `class` *`clase`*, se obtienen las políticas de esa clase. La respuesta del *router* ante tal comando es interpretada posteriormente llamando al comando `interpretar()`.
- `private void interpretar(String s)`. El *router*, ante el comando anterior, no devuelve las instrucciones que se supone que se mandaron para configurar las políticas de la clase, sino que los muestra en un estilo más descriptivo. Por ello, se ha tenido que hacer un proceso inverso, viendo qué salida mostraba ese comando si se enviaba un comando de configuración. En la lista de la ventana se muestran los comandos que se supone que se mandaron para conseguir la configuración que muestra la salida del comando. Por ejemplo, para la siguiente salida:

```
MiRouter#show policy-map mapa_politicas class mapa_clases
       Class mapa_clases
      set ip dscp 0
      set qos-group 0
      Traffic Shaping
         Peak Rate Traffic Shaping
                 CIR 8000 (bps) Max. Buffers Limit 1000 (Packets)
           Bc 256                                  Be 0
      Weighted Fair Queueing
            Bandwidth 10 (%)
            exponential weight 9
            class    min-threshold    max-threshold    mark-probablity
            ------------------------------------------------------------

            0        -                -                1/10
            1        -                -                1/10
            2        -                -                1/10
            3        -                -                1/10
            4        -                -                1/10
            5        -                -                1/10
```

```
6        -                -                1/10
7        -                -                1/10
rsvp      128              256              1/512

MiRouter#
```

El programa mostraría en la lista los siguientes comandos que se tuvieron que mandar para conseguir tal configuración:

- `set ip dscp 0`
- `set qos-group 0`
- `shape peak 8000 1000 256 0`
- `bandwith percent 10`
- `random-detect`
- `random-detect exponential-weighting-constant 9`
- `random-detect prec-based`
- `random-detect precedence rsvp 128 256 512`

Esta interpretación la realiza este método. Para añadir un método a la lista, se utiliza una llamada al método `anadirPolitica()`, descrito a continuación.

- `private void anadirPolitica(String s)`. Este método añade a la lista la cadena que se pasa como parámetro, comprobando previamente si esa cadena ya se encuentra en la lista.
- `protected void Button_Anadir_Condicion_actionPerformed(ActionEvent e)`. Añade una nueva política a la lista. Para ello crea una nueva instancia de la clase `ConfigurarPolicyMap_Politica`, y la hace visible. Después de que el usuario pulse el botón de aceptar, se obtiene la nueva política mediante el comando `getPolitica()` y lo envía al *router*, estando dentro de la configuración de la clase.
- `protected void Button_Editar_Condicion_actionPerformed(ActionEvent e)`. Edita una política seleccionada. Para ello, crea una nueva instancia de la clase `ConfigurarPolicyMap_Politica`, pasándole entre otros parámetros la política que se quiere editar. Tras ello, envía el comando `no` para eliminar la política seleccionada y envía la nueva política estando dentro de la configuración de la clase.
- `protected void Button_Eliminar_Condicion_actionPerformed(ActionEvent e)`. Elimina la política de la clase utilizando el comando `no` seguido de la política dentro de la configuración de la clase.

4.11.3 Clase ConfigurarPolicyMap_Politica

Esta clase se encarga de generar los comandos necesarios para añadir y editar políticas dentro de una clase. Contiene todas las opciones que se pueden enviar al estar dentro de la configuración de una clase perteneciente a un mapa de políticas. Estas opciones son las siguientes:

- Bandwith
- Priority
- Queue-limit
- Random-detect

- Service-policy
- Shape
- Police
- Set

La clase hereda de `SemiModalDialog`, y su constructor posee, además de los parámetros típicos en un diálogo normal, los siguientes:

- Un objeto de la clase `DobleSalida` para comunicación con el *router*.
- Un objeto cuya clase implemente la interfaz `EscuchadorEventos` para añadir eventos generados internamente por la clase.
- Una cadena correspondiente al mapa de políticas que se está configurando.
- Una cadena correspondiente al mapa de clases.
- Una cadena que contiene la política que se quiere evaluar. En caso de que no se quiera editar ninguna política, sino que lo que se quiere es añadir una nueva, este parámetro valdrá `null`.

A continuación se explican los métodos más importantes que se han implementado:

- `private void inicializarCosas()`. Añade los valores que correspondan a los menús desplegables de *IP DSCP*, *Precedence*, acciones conformes, en exceso y de violación, etc.
- `public String getPolitica()`. Después de que el usuario pulse el botón de aceptar, la clase `PolicyMap_Clase` llamará a este método para obtener el comando generado de acuerdo con las opciones seleccionadas en la interfaz gráfica. Si el usuario canceló el diálogo o cerró la ventana sin más, este método devolverá `null`.
- `protected void Button_Aceptar_actionPerformed(ActionEvent e)`. Genera el comando adecuado para establecer la política en el *router* de acuerdo con las opciones seleccionadas en la interfaz gráfica. Si algún parámetro es incorrecto o falta alguna opción por seleccionar el usuario es avisado.

4.12 Paquete cisco.Tablas

Este paquete contiene clases necesarias para trabajar con tablas (`JTable`, del paquete `javax.swing`) y modificar su apariencia de acuerdo con las necesidades de cada aplicación [J9].

4.12.1 Clase Table_Cisco

Esta clase extiende a la clase `JTable` y modifica el comportamiento de algunos aspectos de la tabla:

- `public void showHorScroll(boolean show)`. Indica si se quiere o no que se muestren las barras de desplazamiento horizontales. Si se quiere que se muestren, no se ajustará el ancho de las columnas para que quepan en el ancho de la tabla. Por el contrario, si se quiere que se muestren, se desactivará el redimensionado automático de las columnas.
- `public void setColumnWidth(int pColumn, int pWidth)`. Cambia el ancho de la columna especificada por la variable `pColumn` al nuevo ancho `pWidth`.

- `public void setResizable(int pColumn, boolean pIsResize)`. Establece que se pueda o no cambiar el ancho a una columna concreta de la tabla.
- `public void setSelect(int col, boolean select)`. Indica si se puede o no seleccionar una columna en concreto.
- `public boolean isCellSelected(int row, int column) throws IllegalArgumentException`. Devuelve `true` si la celda indicada por la fila y columna pasadas como parámetros está seleccionada. Si los parámetros no son válidos, se lanza la excepción indicada.
- `public void setHeaderSize(int pColumn)`. Establece el ancho de la columna al ancho generado por el título de la misma columna.

4.12.2 Clase Table_JLabelTableCellRenderer

Esta clase, que implementa la interfaz `TableCellRenderer`, sirve para modificar la forma en la que se muestran los datos en las tablas. Por defecto, las tablas sólo puedan manejar cadenas de texto como contenido para las celdas. Implementando la interfaz anterior, es posible mostrar cualquier tipo de componente gráfico (clase `Component`, del paquete `java.awt`). Por ejemplo, es posible mostrar un menú desplegable, un botón, etc. y hacer que al seleccionar o cambiar de estado estos componentes, cambie de comportamiento la tabla. El único método que posee la interfaz es `public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected, boolean hasFocus, int row, int column)`. La implementación de este método que las celdas se muestren como etiquetas (`JLabel`) pudiendo cambiar el fondo de la misma y su alineación.

4.12.3 Clase Table_Model_Cisco

Esta clase hereda de `DefaultTableModel`. Sobrescribe el método `public boolean isCellEditable(int row, int column)`, de tal forma que ninguna celda sea editable, devolviendo `false`.

4.13 Instalación de herramientas

Como se ha visto anteriormente, la aplicación está programada en el lenguaje Java. En el CD-ROM que se adjunta a este proyecto, se incluye el código compilado de la aplicación. Pero para usarlo es necesario tener instalado el entorno en tiempo de ejecución de Java (JRE, Java Runtime Environment), que también se incluye en el CD-ROM, además de otras librerías.

A continuación se pasa a explicar el contenido del CD-ROM, así como los pasos para instalar las herramientas necesarias y ejecutar el programa.

4.13.1 Contenido del CD-ROM

El CD-ROM posee los siguientes directorios, cuyo contenido se explica a continuación:

- JavaComm. Incluye las librerías de Java necesarias para la comunicación con el puerto serie, tanto para Windows como para Linux.
- JRE. Se incluyen los programas de instalación del entorno en tiempo de ejecución de Java, en versiones tanto para Linux como para Windows.

- Linux. Incluye el directorio completo con el JRE, con las librerías necesarias ya instaladas, así como las clases necesarias para su ejecución directa desde el CD-ROM. Aunque este método es posible, no es recomendable ya que no se tendrá acceso para leer y escribir los archivos de historiales de *logs* y eventos.
- Windows. Incluye el directorio completo con el JRE, con las librerías necesarias ya instaladas, así como las clases necesarias para su ejecución directa desde el CD-ROM. Aunque este método es posible, no es recomendable ya que no se tendrá acceso para leer y escribir los archivos de historiales de *logs* y eventos.
- PFC. Contiene este documento en formado PDF.
- Programa. Contiene las clases compiladas de la aplicación, así como los archivos auxiliares necesarios (imágenes, ayuda, etc.).
- JavaDoc. Contiene la documentación generada por la herramienta `javadoc` a partir de los comentarios de las clases.

4.13.2 Conexión del router

Antes de pasar a instalar nada, comprobaremos que el *router* está correctamente conectado al puerto serie, anotando dicho puerto (COM1, COM2, /dev/ttyS0, /dev/ttyS1, …). A continuación deberemos saber los parámetros que tiene configurados el *router* para trabajar con el puerto serie. Si no se han modificado estos parámetros conservará los valores por defecto: 9600 bps, 8 bits de datos, sin paridad y 1 bit de parada. Si hemos cambiado previamente estos parámetros, deberemos cambiarlos bien en el *router* antes de ejecutar el programa o cambiarlos en las opciones del programa.

4.13.3 Instalación de Java en Windows

Usaremos la versión 1.4.2_04 del entorno de ejecución de Java, incluido en el CD-ROM en el directorio \JRE\Windows. Ejecutaremos el único archivo contenido en ese directorio y seguiremos las instrucciones de instalación, aceptando el directorio por defecto de instalación.

4.13.4 Instalación de JavaComm en Windows

Una vez instalado el entorno de ejecución de Java, necesitaremos instalar las librerías de comunicaciones. Para ello, se seguirán los siguientes pasos:

1. Copiar el archivo del CD-ROM \JavaComm\Windows\win32com.dll al directorio bin donde hallamos instalado el JRE.
2. Copiar los archivos contenidos en el CD-ROM \JavaComm\Windows\comm.jar y \JavaComm\Windows\javax.comm.properties al directorio lib del directorio de instalación del JRE.

4.13.5 Ejecución de la herramienta en Windows

Una vez que hayamos realizado esta operación deberemos copiar el directorio "Programa" del CD-ROM a algún lugar del disco duro, para después ejecutar, desde el mismo directorio donde lo hemos copiado, el siguiente comando:

```
> C:\ruta_al_jre\bin\java -cp "C:\ruta_al_jre\lib\comm.jar;
.\classes\;.\lib\jbcl.jar" cisco.Cisco
```

También es posible ejecutar la herramienta directamente desde el CD-ROM ejecutando el *script* `Ejecutar.bat` dentro del directorio \Windows del CD-ROM.

4.13.6 Instalación de Java en Linux

Para instalar el entorno en tiempo de ejecución de Java, ejecutaremos desde una consola el archivo contenido en el directorio del CD-ROM \JRE\Linux. Tras ello, deberemos aceptar las condiciones de uso y pasará a instalarse en el directorio local de usuario.

4.13.7 Instalación de JavaComm en Linux

Para Linux, además de la librería estándar de Java para comunicaciones, se necesitarán las librerías RxTx [J7] [J10].

Seguiremos los siguientes pasos:

1. Copiar los archivos libSerial.so y libParallel.so del directorio \JavaComm\Linux\RxTx del CD-ROM al directorio lib\i386 donde se haya instalado el JRE.
2. Copiar el archivo jcl.jar del directorio \JavaComm\Linux\RxTx del CD-ROM al directorio lib\ext donde se haya instalado el JRE.
3. Copiar el archivo comm.jar del directorio \JavaComm\Linux del CD-ROM al directorio lib\ext donde se haya instalado el JRE.
4. Crear un nuevo archivo llamado javax.comm.properties dentro del directorio lib donde se haya instalado el JRE con el siguiente contenido:

   Driver=gnu.io.RXTXCommDriver

4.13.8 Ejecución de la herramienta en Linux

Una vez que hayamos realizado esta operación deberemos copiar el directorio "Programa" del CD-ROM a algún lugar del disco duro, para después ejecutar, desde el mismo directorio donde lo hemos copiado, el siguiente comando:

```
> /ruta_al_jre/bin/java -cp "/ruta_al_jre/lib/ext/comm.jar:
./classes/:./lib/jbcl.jar" cisco.Cisco
```

También es posible ejecutar la herramienta directamente desde el CD-ROM ejecutando el *script* `Ejecutar.sh` dentro del directorio \Linux del CD-ROM.

4.13.9 Posibles problemas en Linux

Es posible que según la gestión de permisos y archivos por parte del sistema operativo, se deban dar permisos al programa para el bloqueo de archivos. Para solventar tal problema, se deberá establecer el grupo propietario del directorio `/var/lock` al grupo `uucp`. Así mismo se deberá añadir el usuario que ejecute la aplicación al grupo `uucp` como grupo secundario. Los comandos que se deben ejecutar son los siguientes, entrando al sistema como usuario `root`:

- `chgrp uucp /var/lock`
- `adduser usuario_actual uucp`

Capítulo 5
Conclusiones y líneas futuras

5.1 Conclusiones

Como hemos podido comprobar a lo largo del desarrollo de este proyecto, se ha conseguido desarrollar una herramienta útil para configurar los aspectos relacionados con los servicios diferenciados en los enrutadores Cisco de la serie 2600. Mediante la interfaz gráfica implementada es posible tener en pantalla todas las opciones de configuración para las herramientas ya descritas:

- Configuración de las direcciones IP de las interfaces
- Configuración de listas de control de acceso
- Mapas de clases
- Mapas de políticas
- Establecer las políticas de entrada y salida en las interfaces.

La herramienta, al estar desarrollada en lenguaje Java, es posible ejecutarla en un sinfín de sistemas operativos siempre y cuando haya disponible una máquina virtual de Java para ese sistema; en la actualidad, Sun ha desarrollado las mencionadas máquinas virtuales para la mayoría de máquinas: Windows, Solaris, Macintosh, BeOS, Linux, Unix, etc. El único inconveniente que puede surgir a la hora de exportar la aplicación a otros entornos puede ser la disponibilidad de la librería de comunicaciones de Java para dichas plataformas. En la actualidad existen librerías estándar para los sistemas operativos Windows y Solaris, aunque como hemos podido comprobar, no ha sido difícil exportar la aplicación a un entorno no soportado, como ha sido Linux, gracias a las librerías RxTx, desarrolladas por Keane Jarve. Al igual que existen librerías para este sistema operativo, es probable que también hayan sido desarrolladas librerías para los sistemas operativos mencionados.

Los *routers* Cisco de la serie 2600 tienen una base ampliamente instalada en organizaciones, empresas, compañías de cualquier tamaño, universidades, instituciones públicas, etc. debido a su alto rendimiento, bajo coste y posibilidades de ampliación. Además, con la creciente demanda de acceso a Internet y la aplicación de nuevas tecnologías como Voz sobre IP, que requieren de necesidades de tráfico especiales como un bajo retardo, se hace necesario el uso de los servicios diferenciados para marcar los paquetes y recibir un trato especial en las máquinas que gestionan la red: *routers*, conmutadores, pasarelas, etc. La aplicación desarrollada hace que el trabajo de configuración de los servicios diferenciados sea un trabajo menos arduo y más llevadero gracias a su interfaz gráfica.

Por otro lado, a la herramienta se le puede dar uso en la enseñanza. Al igual que sucede con sistemas de otras marcas, como Teldat, que poseen herramientas gráficas para la configuración de sus dispositivos, esta herramienta puede ayudar a los estudiantes a aprender a aplicar y configurar los servicios diferenciados de acuerdo con las necesidades de la organización en los dispositivos Cisco, que en la actualidad son los de mayor difusión en el mundo empresarial y que soportan la mayoría del tráfico que circula por Internet. Así mismo, al ser una herramienta desarrollada en lenguaje Java, ampliamente difundido en la actualidad y de fácil aprendizaje, es posible ir corrigiendo sobre la marcha aquellos fallos que se posiblemente se encuentren durante la ejecución.

5.2 Líneas futuras

Las posibilidades de expansión de la aplicación sólo tienen límite con las opciones de configuración del *router*: enrutamiento, túneles, redes privadas virtuales, cifrado, comunicaciones serie, RDSI, Voz sobre IP, completar las listas de control de acceso a todos los protocolos soportados, opciones de cortafuegos, etc.

También sería interesante la exportación de la herramienta a otros sistemas operativos, que aunque menos utilizados, también permitirían ejecutar la herramienta gracias a la máquina virtual de Java.

Capítulo 6
Manual de usuario

6.1 Inicio del programa

En primer lugar, se instalará el programa y se iniciará de acuerdo con lo señalado en el apartado 4.13.

Una vez que se inicia el programa mientras éste se carga, se mostrará la siguiente figura indicando el proceso de carga del programa:

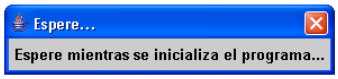


Figura 13 - Mensaje mostrado mientras se inicia el programa

Cuando se abra el puerto, se mostrará un mensaje indicando que se están mandando o recibiendo datos:

Figura 14 - Mensaje mostrado mientras se comunica con el *router*

El mensaje anterior se repetirá a lo largo del programa cada vez que se envíen o reciban datos.

Una vez arrancado el programa, hay tres posibilidades de iniciarlo. La primera de ellas, es que por algún motivo, el puerto no haya podido ser abierto. En este caso, aparecerá una ventana como la siguiente:

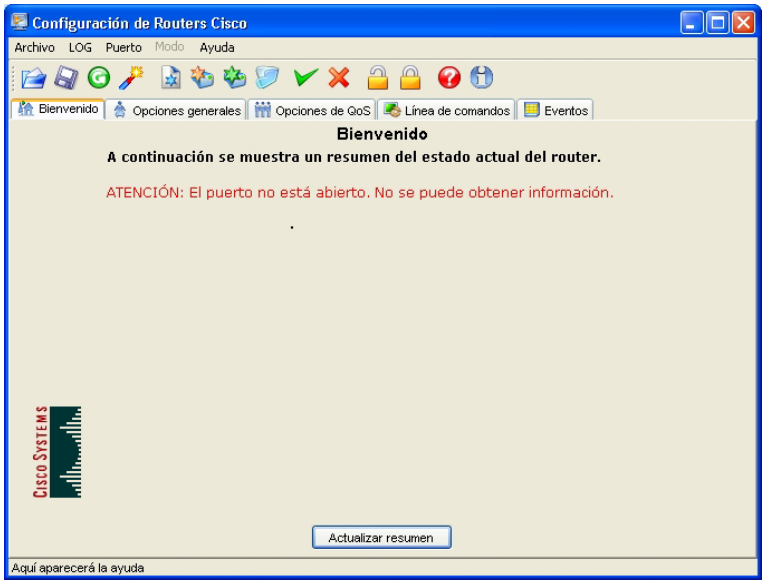


Figura 15 - Inicio del programa con el puerto cerrado

Si por el contrario el puerto se ha abierto, es posible que el *router* se encuentre en modo usuario (lo más normal) o en modo administrador (si ya se ha estado utilizando anteriormente).

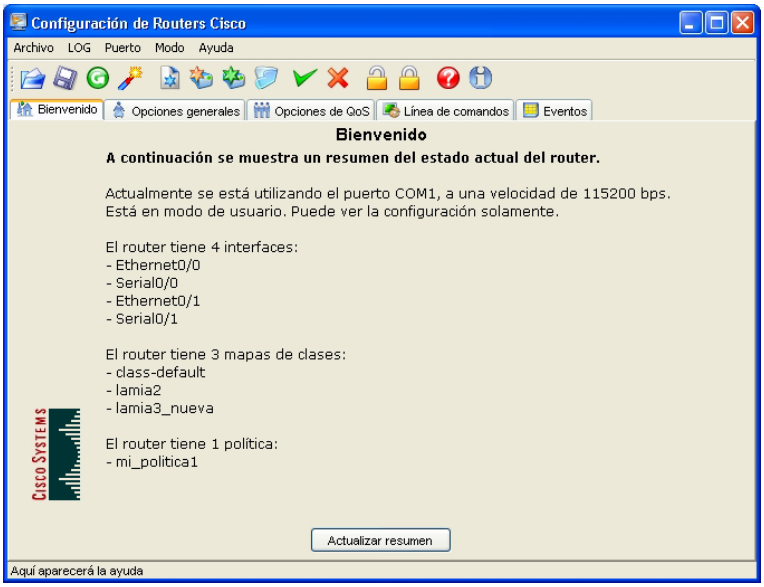


Figura 16 - Inicio del programa en modo usuario

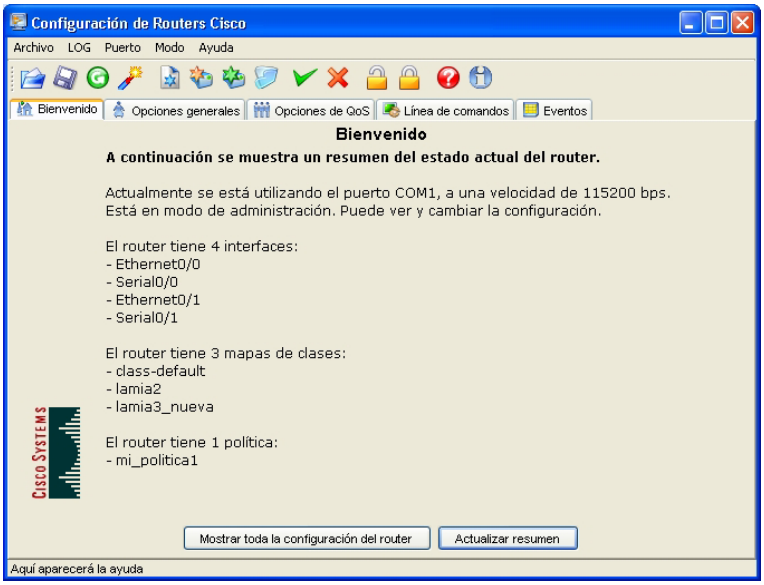


Figura 17 - Inicio del programa en modo administrador

En la figura se pueden observar cuatro zonas claramente diferenciadas:

- La barra de menús contiene comandos para trabajar con las configuraciones del *router* y las opciones del programa (menú Archivo); utilizar los historiales de la ventana del terminal (menú LOG); conmutar entre tener el puerto cerrado o abierto (menú Puerto); una vez que el puerto está abierto es posible cambiar entre el modo usuario (EXEC) y el modo EXEC privilegiado (administrador, menú Modo); el último menú contiene dos opciones para abrir la página principal de la ayuda y otra para mostrar la ventana de información del programa:

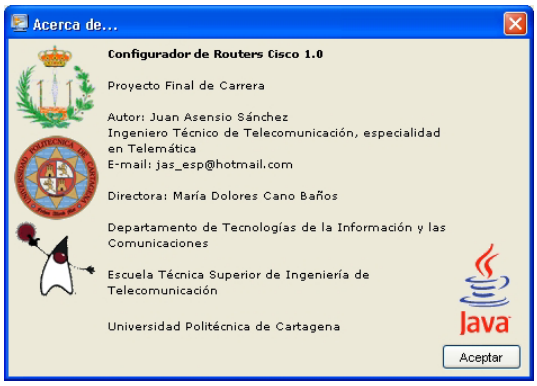


Figura 18 - Ventana Acerca de...

- La barra de herramientas, contiene accesos directos a las opciones de los menús descritas anteriormente.
- Una serie de pestañas que constituyen el cuerpo de la ventana. La primera de ellas muestra un resumen del estado del programa y del *router*: interfaces, mapas de clases y mapas de políticas, configuración del puerto serie, etc.; la segunda pestaña permite configurar opciones generales del *router* como las direcciones de las interfaces; las opciones relacionadas con servicios diferenciados son configuradas desde la tercera pestaña; la cuarta pestaña muestra una ventana de terminal básica, desde la cual se pueden enviar comandos y combinaciones especiales de teclas; por último, la quinta pestaña muestra una tabla con los eventos generados internamente por el programa: interpretación de cadenas, creación de instancias, etc.
- Una barra de estado donde se muestra una pequeña ayuda sobre el programa.

6.2 Descripción de los Menús

En este apartado se decribirá la funcionalidad de cada una de las opciones de los menús.

6.2.1 Menú Archivo

Este menú posee cinco opciones:

- Guardar configuración como inicial. Guarda la configuración actual del *router* para que se utilice como por defecto la próxima vez que se reinicie. Esto debe hacerse cada vez que se hagan modificaciones en la configuración, ya que si se apaga el *router* sin realizar esta operación, las modificaciones realizadas no se tendrán en cuenta la próxima vez que se inicie.
- Cargar configuración inicial. Carga en memoria la última configuración guardada mediante el comando anterior, descartando cualquier modificación que se haya realizado posteriormente.
- Reiniciar *router*. Hace un reinicio del *router*, cargando la configuración inicial.
- Opciones. Muestra el cuadro de diálogo de opciones, explicado ampliamente en el apartado 4 de este mismo capítulo.
- Salir. Sale de la aplicación, cerrando previamente el puerto serie, y guardando los historiales del terminal y los eventos si se ha establecido así en las opciones.

6.2.2 Menú LOG

Este menú permite trabajar con el historial de la ventana de eventos. Las opciones que se muestran son las siguientes:

- Cargar LOG. Muestra un diálogo para seleccionar un archivo de texto, y establece el contenido del archivo como contenido de la ventana del terminal.
- Añadir LOG. Muestra un diálogo para seleccionar un archivo de texto, y añade su contenido al contenido actual de la ventana del terminal.
- Guardar LOG. Muestra una ventana para seleccionar el archivo en el cual se quiere guardar el contenido actual de la ventana de terminal.
- Borrar LOG. Limpia la ventana del terminal.

6.2.3 Menú Puerto

Mediante este menú es posible conmutar entre tener el puerto abierto y cerrar el puerto. Pulsando sobre la opción “Abierto” se abrirá el puerto en caso de que esté cerrado. Por el contrario, pulsando sobre la opción “Cerrado”, se cerrará el puerto siempre y cuando se haya abierto previamente.

6.2.4 Menú Modo

Este menú sólo es accesible cuando el puerto está abierto. Permite intercambiar entre los dos modos de administración del *router*: modo Usuario (EXEC) y modo Administrador (EXEC privilegiado). Para pasar del modo usuario al modo administrador se deberá introducir la contraseña de administrador, ofreciendo la aplicación hasta tres intentos para introducirla correctamente.

6.2.5 Menú Ayuda

Contiene opciones referentes a la ayuda del programa e información acerca de la aplicación. Pulsando sobre la primera opción (Página inicial de la ayuda) se mostrará la página principal de la ayuda en un navegador independiente.

Pulsando sobre la segunda opción del menú, Acerca de, se mostrará una nueva ventana con información acerca de la aplicación y el Proyecto Final de Carrera: autor, directora, etc.

6.3 Estado del Router

Como ya se ha visto, la primera pestaña de la aplicación muestra un resumen sobre el estado general del *router*, indicando si el puerto serie está abierto; en caso afirmativo, el nombre del puerto usado, así como la velocidad a la que está conectado.

También informa del modo en el que se encuentra conectado al *router*: usuario o administrador. Además muestra una lista con las siguientes características del *router*:

- Interfaces de comunicaciones que posee.
- Mapas de clases.
- Mapas de políticas.

6.4 Menú de Opciones

Desde el menú de Opciones se pueden seleccionar los parámetros del puerto serie. Permite establecer el puerto que se usará, la velocidad, los bits de datos, los bits de parada, el tipo de paridad usada, el control de flujo, el tiempo de espera máximo antes de abrir el puerto, el tiempo máximo que se esperará para que el *router* dé una respuesta al iniciar el programa, y el tiempo que se debe esperar desde que se recibe el primer dato de una respuesta hasta que se pasa a leer la respuesta por completo (este tiempo deberá ser mayor conforme disminuye la velocidad del puerto serie). Las opciones descritas anteriormente se encuentran en la primera pestaña del diálogo:

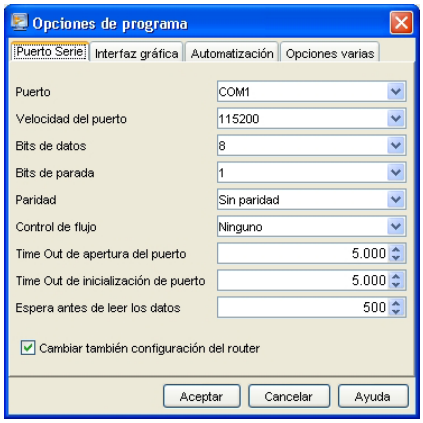


Figura 19 - Opciones del puerto serie

La segunda pestaña del menú de opciones permite establecer el tipo de estilo (*Look And Feel*) que usarán las ventanas de la aplicación. Los cambios que se hagan en este aspecto tendrán efecto para las nuevas ventanas que se creen (no las ya creadas) y para cuando se inicie de nuevo el programa. Las siguientes capturas muestran los tres estilos disponibles en la distribución Java para Windows:

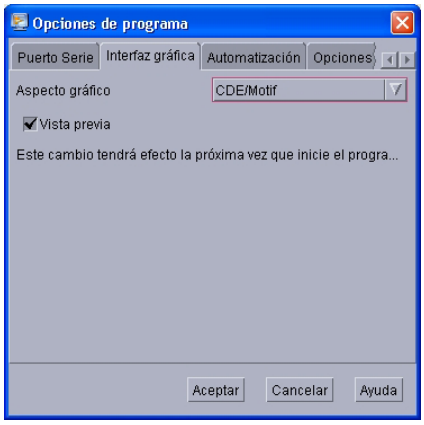


Figura 20 - Estilo CDE/Motif

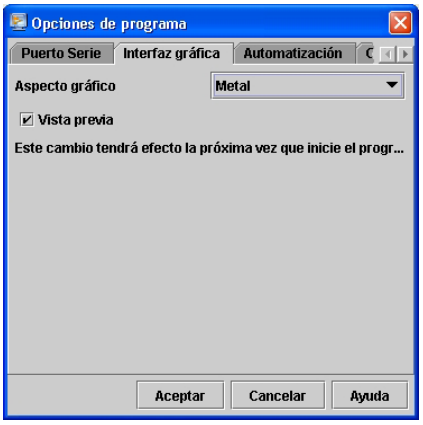


Figura 21 - Estilo Metal

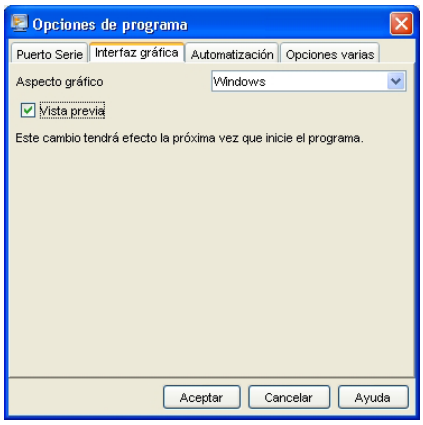


Figura 22 - Estilo Windows

La tercera pestaña de esta ventana permite establecer opciones de automatización del programa. Mediante estas opciones es posible cargar automáticamente un archivo de históricos de comandos (ventana de terminal) y de eventos. Así mismo, es posible guardar esos mismos datos de nuevo al salir del programa, sin necesidad de intervención por parte del usuario. La última opción de la pestaña ("Añadir al contenido anterior del fichero") es útil para reducir la carga del programa, ya que no carga al inicio los datos, pero sí que los guarda al salir añadiéndolos al contenido del fichero, por lo que la carga del programa y su cierre se ejecutarán en menos tiempo.

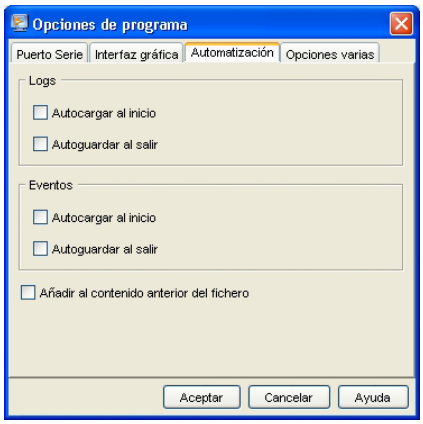


Figura 23 - Opciones de automatización

La última de las pestañas de este cuadro de diálogo permite especificar opciones orientadas a la depuración del programa:

- Mostrar eventos también por la salida estándar. Muestra los eventos que se generan internamente en el programa, además de en la tabla de eventos de la ventana principal, por la salida estándar.
- Mostrar ventana independiente del terminal. Indica si se quiere que se muestre una ventana independiente de la ventana principal donde se muestran los datos que se van y enviando y recibiendo a través del puerto serie.
- Mostrar eventos de entrada/salida (clase `DobleSalida`). Esta opción indica si se deben mostrar los eventos generados por la clase anterior. Puede ser útil desactivar esta opción ya que esta clase genera un gran número de eventos y que no suelen ser necesarios para comprobar el funcionamiento del programa.

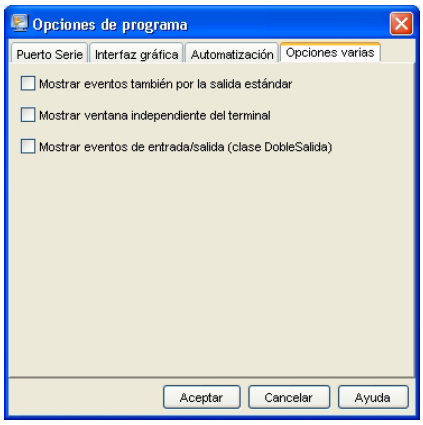


Figura 24 - Opciones varias

6.5 Opciones Generales

En la segunda pestaña de la ventana principal tenemos las opciones generales de configuración del *router*. Desde esta pestaña se puede:

- Establecer y cambiar las direcciones IP a las interfaces del *router*.
- Activar y configurar el enrutamiento RIP.
- Cambiar la contraseña de administración.
- Cambiar el nombre del *router*.

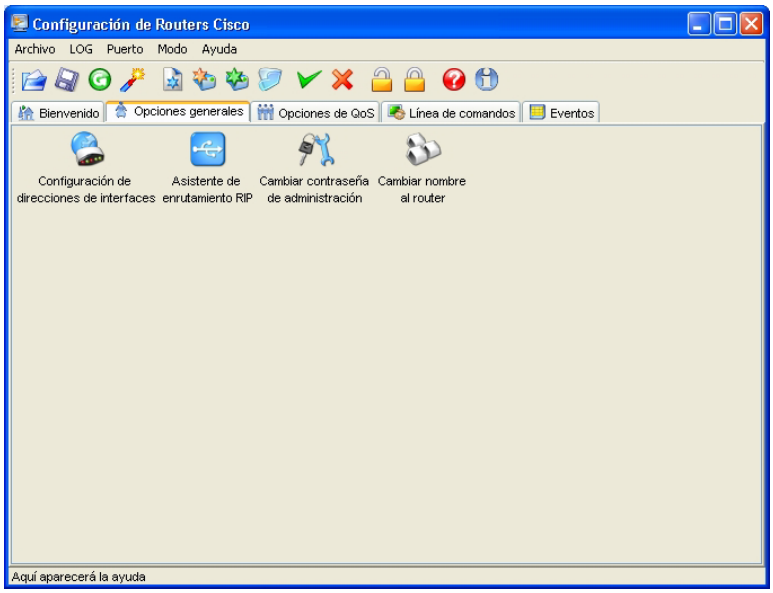


Figura 25 - Pestaña de opciones generales

6.5.1 Configuración de direcciones de interfaces

Desde esta ventana se puede administrar la forma en la que las interfaces obtienen su dirección a través de DHCP o bien establecer sus direcciones directamente a través de direcciones estáticas. En la parte superior de la ventana se muestra una lista con las interfaces de red disponibles en el *router*. Seleccionando una de ellas se seleccionarán en la parte inferior las opciones de acuerdo con su configuración. Si es estática se rellenarán los campos

correspondientes con los valores de las direcciones IP y de las máscaras. Si está configurada para que obtenga su dirección a través de DHCP se seleccionará el valor correspondiente.

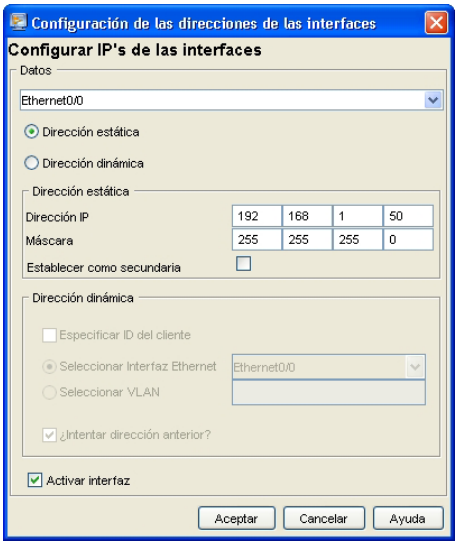


Figura 26 - Interfaz Ethernet con dirección estática

Figura 27 - Interfaz Ethernet con dirección dinámica

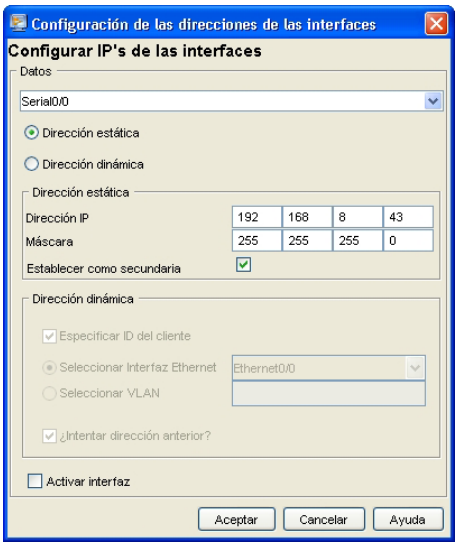


Figura 28 - Interfaz Serie con dirección estática

Figura 29 - Interfaz serie con dirección dinámica

Si la interfaz obtiene su dirección de otro modo (por ejemplo, IPCP) se mostrará un mensaje de aviso diciendo que dicha opción no está implementada en este programa.

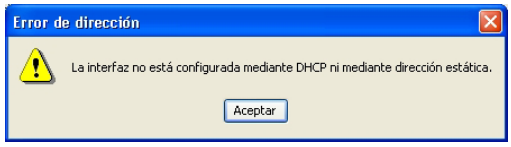


Figura 30 - Mensaje de error

Para activar o desactivar la interfaz se puede usar la casilla de la parte inferior. Así mismo, si la interfaz está activada, pero tiene algún problema como por ejemplo que el cable

está desconectado o que el tipo de encapsulación es distinto en cada extremo, se mostrará un mensaje de error advirtiéndolo.

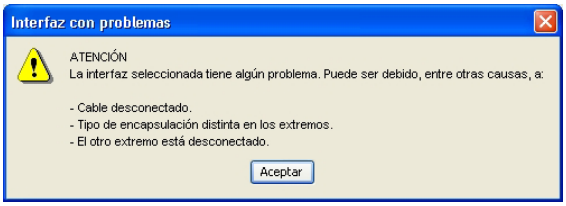


Figura 31 - Mensaje de aviso

6.5.2 Asistente de enrutamiento RIP

Mediante este asistente es posible activar el enrutamiento RIP en el *router*. Una vez pulsado el botón, se le solicitará al usuario todas las redes a las que está conectado el *router*.

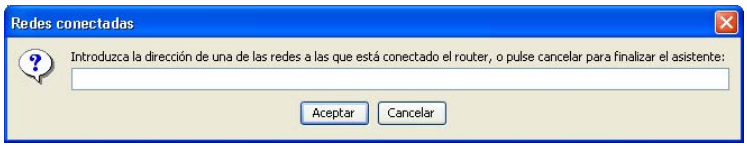


Figura 32 - Asistente de enrutamiento RIP

Una vez que se hayan introducido todas las redes, se debe pulsar el botón cancelar, tras lo cual se mandarán los comandos al *router* para la activación del protocolo con las redes especificadas.

6.5.3 Cambiar contraseña de administración

Este asistente sirve para cambiar la contraseña de administración del *router* (la que provee el acceso al modo EXEC privilegiado). Cuando se pulse el botón se le solicitará al usuario que introduzca la nueva contraseña de administración mediante el siguiente diálogo:

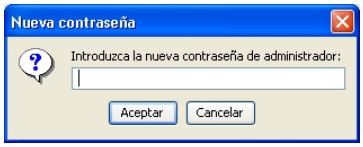


Figura 33 - Asistente para cambiar la contraseña

Cabe recordar que esta contraseña solo tendrá validez mientras se esté usando la configuración actual. Si se reinicia el *router* o se apaga, la contraseña será la anterior a no ser que se guarde la nueva configuración mediante el menú Archivo, Guardar configuración como inicial.

6.5.4 Cambiar nombre al *router*

Tras pulsar el botón para iniciar el asistente para cambiar el nombre al *router*, se mostrará la siguiente ventana, solicitando al usuario que introduzca el nuevo nombre del *router* que se quiere usar:

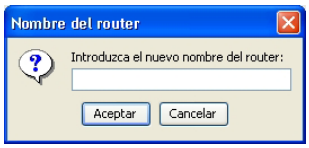


Figura 34 - Asistente para cambiar nombre al *router*

Una vez que se ha pulsado el botón de aceptar se manda el comando al *router* para que se cambie el nombre.

6.6 Opciones de Calidad de Servicio (QoS)

Las opciones relacionadas con calidad de servicio (*Quality of Service*, QoS) se encuentran situadas en esta tercera pestaña. Como se explicó en el capítulo 4, los pasos a seguir para aplicar los servicios diferenciados son los siguientes:

1. Definir una clase de tráfico (comando *class-map*).
2. Crear un mapa de políticas asociando a la clase creada anteriormente una o varias políticas de calidad de servicio (comando *policy-map*).
3. Asociar el mapa de políticas recién creado a una interfaz (*service-policy*).

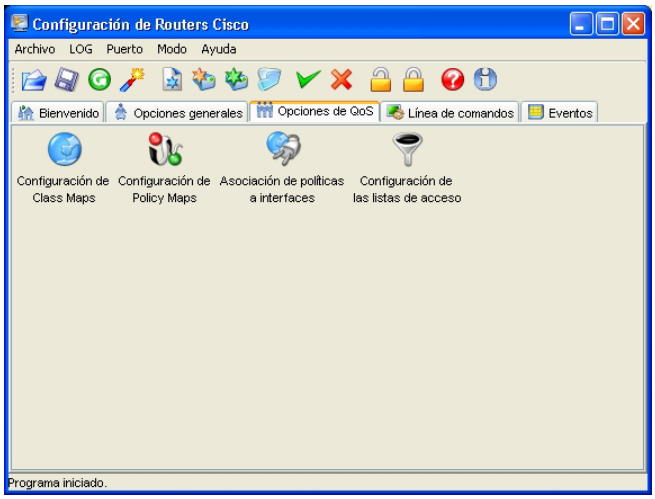


Figura 35 - Opciones de QoS

6.6.1 Listas de acceso

Pulsando sobre el botón de las listas de acceso, se mostrará una ventana con las listas de acceso disponibles.

Figura 36 - Ventana con las listas de acceso

La primera opción que se muestra en la ventana es la de añadir una nueva lista de acceso. Con esta opción se mostrará una ventana donde se puede seleccionar el número de la lista que se quiere usar. En este apartado cabe mencionar que se han eliminado de la lista aquellos números que ya están siendo utilizados por otras listas de acceso. Además se debe tener en cuenta que las listas comprendidas entre el 1 y el 99 son listas de acceso estándares y las comprendidas entre 100 y 199 son listas de acceso extendidas.

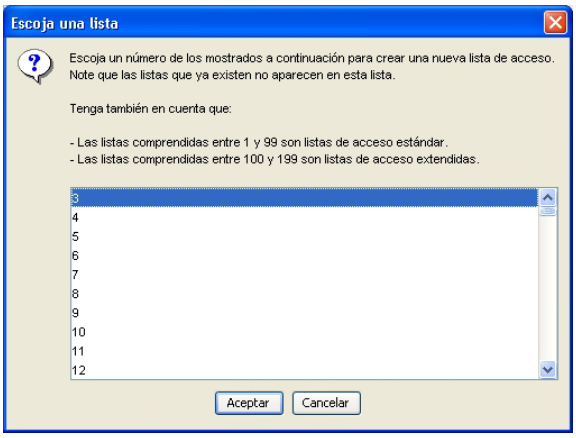


Figura 37 - Añadir una lista de acceso

Una vez se seleccione el número de la lista de acceso que se quiere añadir, se mostrará una nueva ventana con las condiciones que posee la lista de acceso, inicialmente vacía. Este cuadro permite especificar la acción que se desea realizar con el paquete en una ACL estándar (permitirla en la lista o denegarla) y la dirección IP desde la cual procede el paquete. Además permite activar la casilla de *log*, mediante la cual se podrá tener conocimiento de los paquetes que se comparan con la lista de acceso.

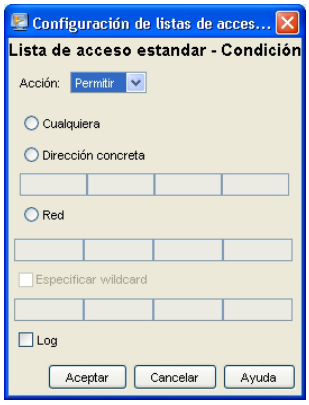


Figura 38 - Añadir condición para la lista estándar

Si por el contrario se ha seleccionado añadir una lista extendida, se mostrará un nuevo cuadro de diálogo donde se debe especificar el tipo de condición que se desea añadir, siendo esta elección de tres tipos: IP, TCP ó UDP. Todas estas listas permiten las mismas opciones que las listas estándar, diferenciando además entre direcciones de origen y de destino, y permitiendo hacer un *log* añadido incluyendo la interfaz de entrada del paquete, además de las opciones específicas de cada clase de lista.

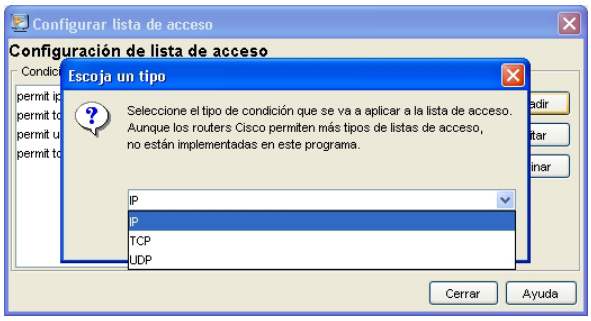


Figura 39 - Añadir condición para lista extendida

La elección IP permite añadir condiciones para paquetes basados en los campos IP (DSCP, *Fragments*, *Precedence*, *Time-Range*, TOS).

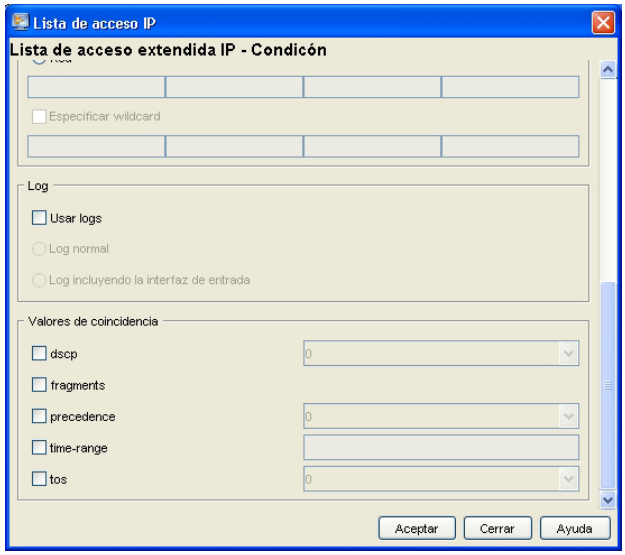


Figura 40 - Condiciones para lista extendida IP

La elección TCP permite añadir condiciones para paquetes basados en los campos TCP (ACK, DSCP, *Established*, *Fin*, *Fragments*, *Precedence*, PSH, Rango de puertos, RST, SYN, *Time-Range*, TOS, URG).

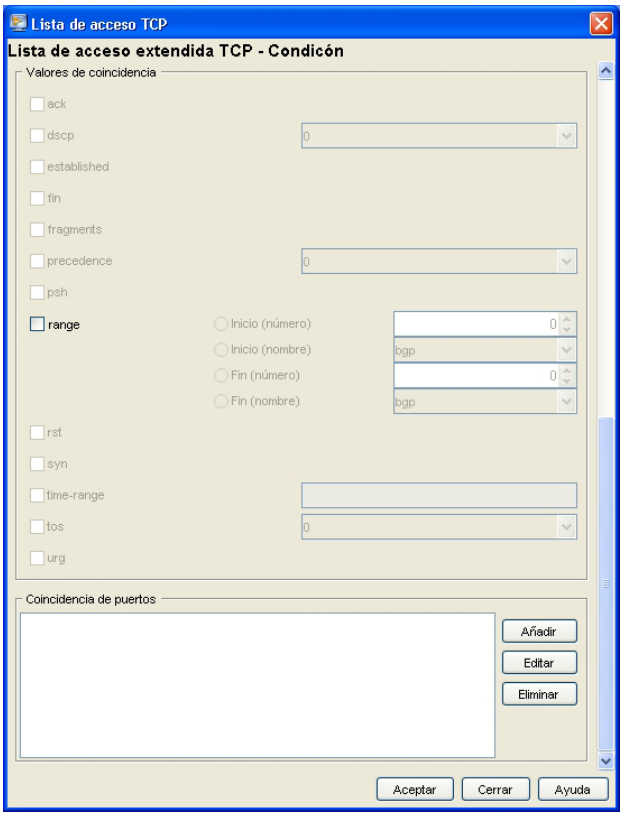


Figura 41 - Condición para lista extendida TCP

La elección UDP permite añadir condiciones para paquetes basados en los campos UDP (DSCP, *Fragments*, *Precedence*, Rango de puertos, *Time-Range*, TOS).

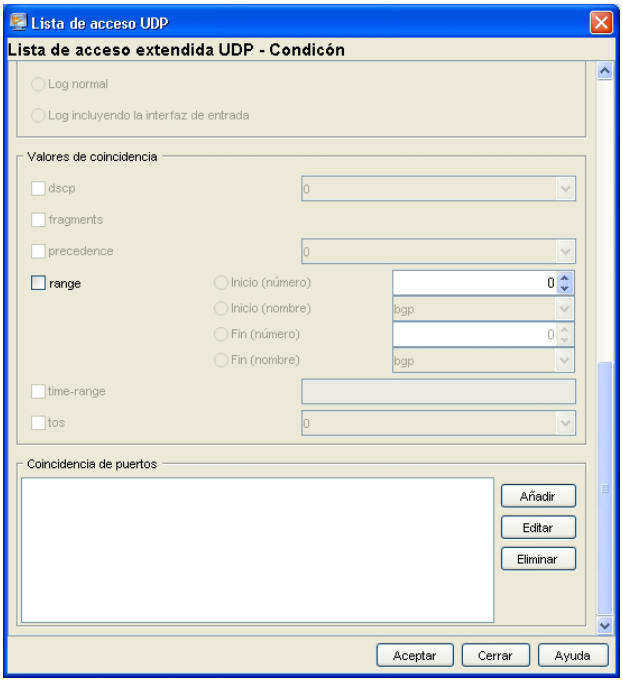


Figura 42 - Condición para lista extendida UDP

Una vez mostradas estas ventanas es posible generar la condición que deben cumplir los paquetes para que pertenezcan a esta lista de control de acceso. Cuando se pulse el botón de aceptar se enviará el comando al *router* y se mostrará en la lista de condiciones. Las

condiciones también pueden ser editadas una vez que se han añadido, seleccionándola y pulsando el botón editar. Según el tipo de lista y el tipo de condición, se abrirá la ventana correspondiente con las opciones de la ventana seleccionadas según la condición seleccionada:

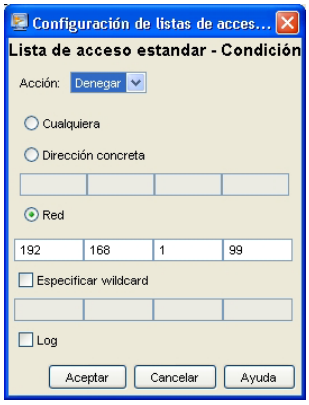


Figura 43 - Editar una condición (de una lista estándar)

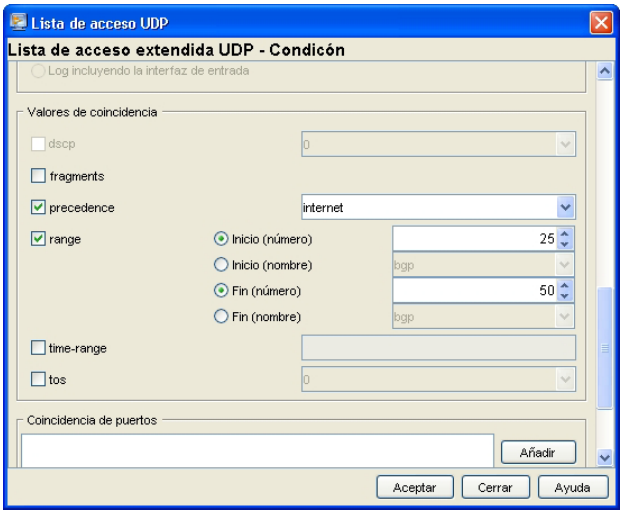


Figura 44 - Editar una condición (de una lista extendida UDP)

También es posible eliminar condiciones de la lista de acceso pulsando sobre el botón de eliminar. La condición se eliminará definitivamente tras aceptar la confirmación que se muestra:

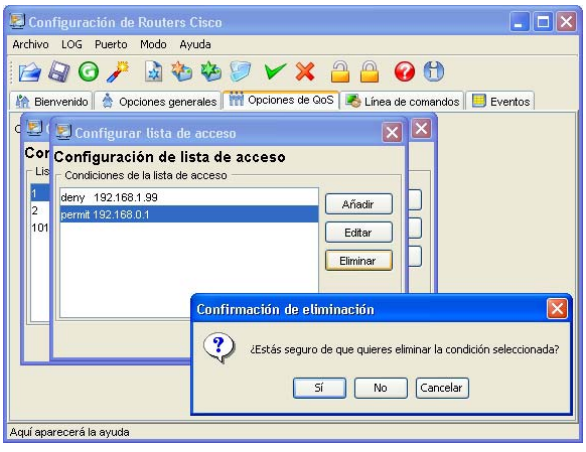


Figura 45 - Eliminar condición

Desde la ventana principal de las listas de acceso también es posible editar y eliminar listas de acceso. Pulsando sobre el botón editar seleccionando previamente una lista de acceso, se mostrará una nueva ventana, donde es posible añadir, editar y eliminar las condiciones de la forma descrita anteriormente.

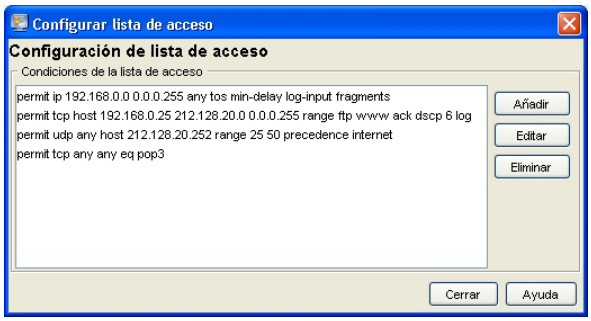


Figura 46 - Editar una lista de acceso

Para eliminar una lista de acceso seleccionaremos primero la lista de acceso que se desea eliminar y tras ello se pulsará el botón de eliminar, que, tras una confirmación, eliminará la lista de acceso.

Figura 47 - Confirmación para eliminar una lista

6.6.2 Mapas de clases

Pulsando sobre el botón para la configuración de los mapas de clases se mostrará una ventana donde en la parte superior se muestra un menú desplegable con los mapas de clases disponibles en el *router*.

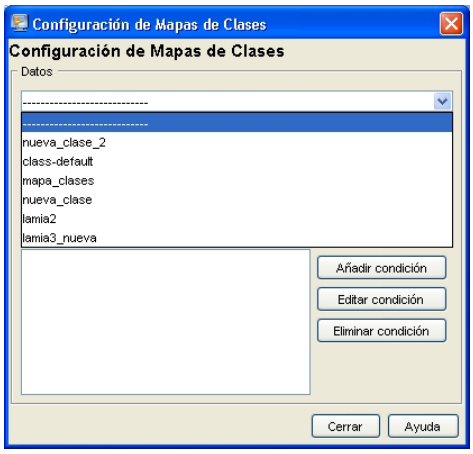


Figura 48 - Ventana de configuración de los mapas de clases

Cuando se seleccione un elemento del menú, en la parte inferior se muestran las condiciones que deben cumplir los paquetes para que pasen a formar parte de la clase.

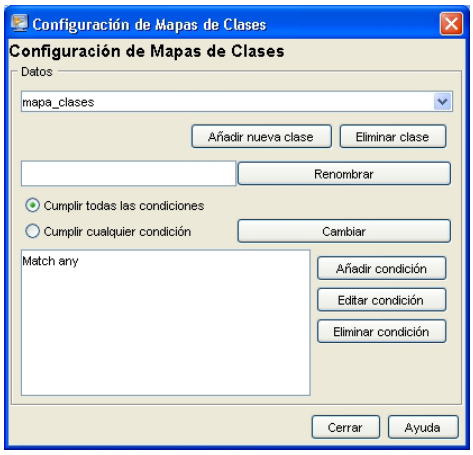


Figura 49 - Condiciones de un mapa de clases

Una vez se ha seleccionado un mapa de clases es posible cambiar el nombre, escribiendo en la casilla de texto para tal efecto el nombre, y pulsando después el botón de renombrar.

También es posible cambiar el modo en que se deben cumplir las condiciones para que un paquete pase a formar parte de la clase. Una vez seleccionado el modo en que se deben cumplir, se debe pulsar el botón de cambiar para que el cambio surja efecto.

Es posible añadir, editar y eliminar las condiciones del mapa de clases. Si se pulsa sobre añadir se mostrará una nueva ventana donde se pueden elegir las opciones para generar la nueva condición. Si se pulsa el botón para editar la condición, se mostrará una ventana similar pero con las opciones ya seleccionadas de acuerdo con la condición.

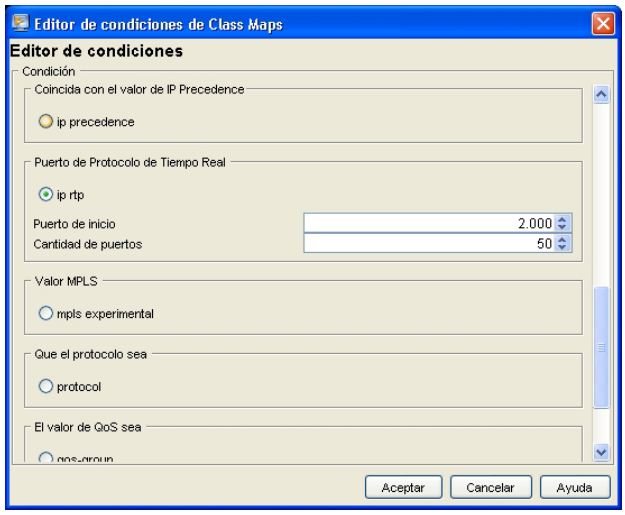


Figura 50 - Editar una condición del mapa de clases

6.6.3 Configurar los mapas de políticas

Mediante esta opción es posible asignar acciones a los paquetes que pertenezcan a las clases anteriormente definidas. En la ventana principal que aparece una vez pulsado el botón para configurar los mapas de políticas, aparece en la parte superior una lista con los nombres de los mapas de políticas ya configurados.

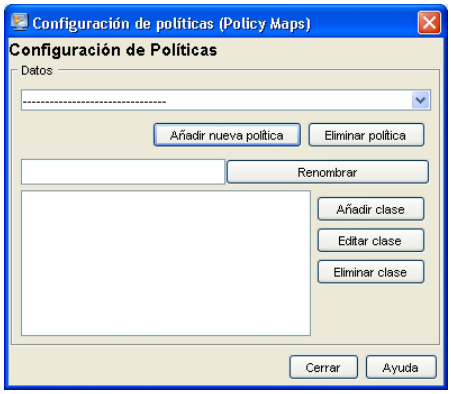


Figura 51 - Pantalla inicial de la configuración de los mapas de clases

En primer lugar es posible añadir nuevos mapas políticas a la configuración del *router*. Para ello, pulsaremos el botón de añadir nueva política, tras lo cual se nos preguntará el nombre de la nueva política. Tras introducirlo se creará, y se mostrará ya en la lista, aunque aún no tendrá ninguna clase asociada.

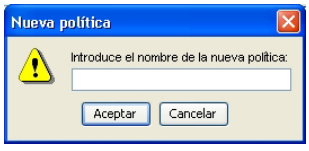


Figura 52 - Añadir un nuevo mapa de políticas

Desde la misma ventana es posible eliminar un mapa de políticas seleccionado. Para ello, seleccionaremos el mapa que queremos eliminar en el menú desplegable, tras lo que se nos mostrará una confirmación de eliminación, Si la aceptamos, se mandarán los mensajes al *router* para eliminar la política.

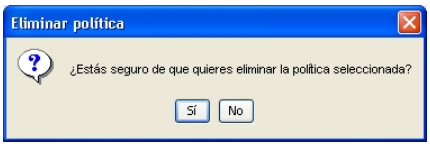


Figura 53 - Eliminar un mapa de políticas

Seleccionando alguna de las opciones, en la parte inferior se mostrarán las clases configuradas dentro de ese mapa de políticas. Es posible tener más de un mapa de clases dentro de la misma política, con políticas (acciones) distintas para cada clase.

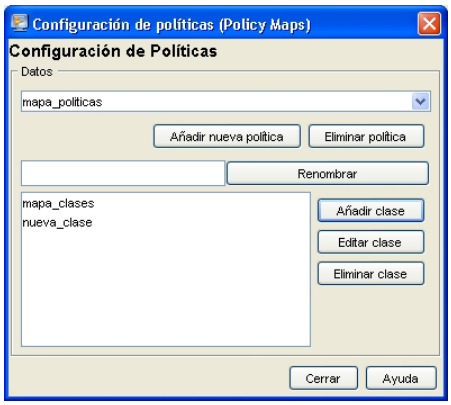


Figura 54 - Clases del mapa de políticas

Desde la pantalla principal es posible renombrar el mapa, escribiendo el nuevo nombre en la casilla correspondiente y pulsando después el botón de renombrar.

Para añadir una nueva clase al mapa de políticas, pulsaremos el botón añadir. Se nos mostrará una nueva ventana donde podemos seleccionar la clase que queremos añadir al mapa de políticas.

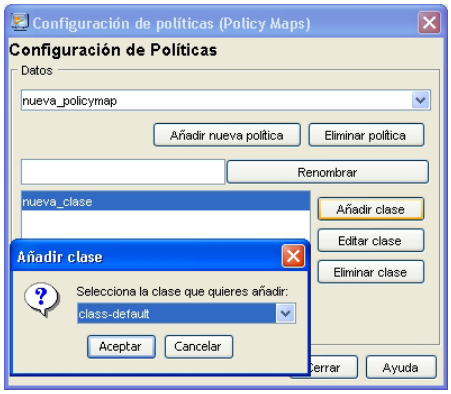


Figura 55 - Añadir nueva clase

También es posible eliminar una clase ya creada mediante el botón de eliminar clase.

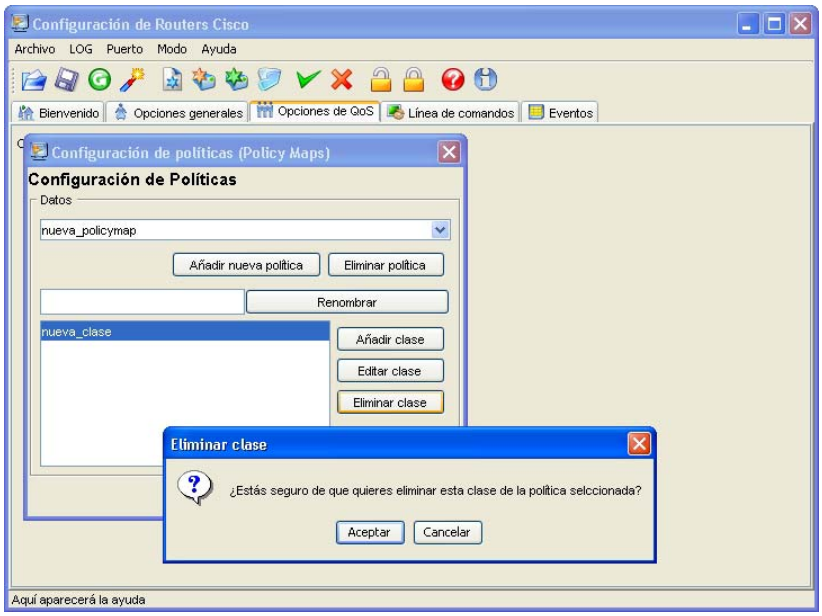


Figura 56 - Eliminar clase

Una vez añadida la clase, deberemos editarla para poder añadir las políticas a realizar con los paquetes que pertenezcan a la clase seleccionada. Si seleccionamos una clase y pulsamos sobre editar se nos mostrará una ventana como la siguiente:

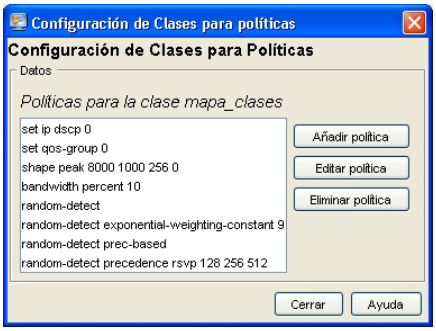


Figura 57 - Editar clase

Si acabamos de añadir una clase, la ventana anterior aparecerá vacía. Podremos añadir políticas, es decir, acciones a realizar, para aplicarlas a los paquetes que cumplan la clase seleccionada. Pulsando el botón de añadir aparecerá la siguiente ventana:

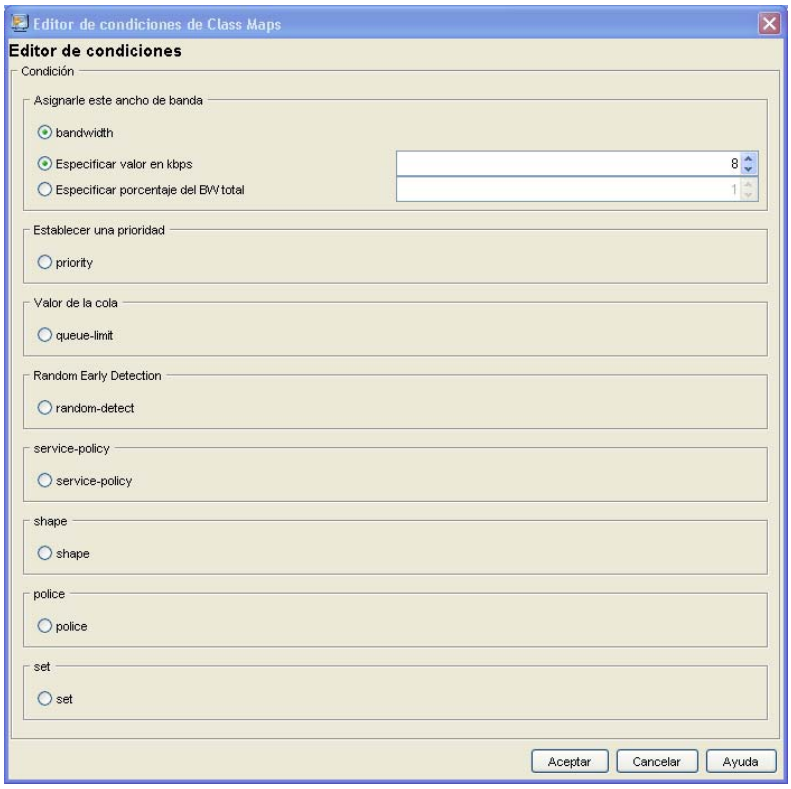


Figura 58 - Añadir nueva política

En dicha ventana podremos seleccionar las siguientes opciones:

- Establecer un ancho de banda mínimo en kilobits por segundo o en porcentaje del ancho de banda total disponible para la interfaz.
- Establecer prioridades en cuanto a velocidad.
- Establecer el valor de la cola.
- Marcar los paquetes con un determinado valor de IP DSCP o IP Precedence.
- Establecer otro mapa de políticas como política.
- Establecer acciones a emprender con los paquetes de acuerdo con umbrales de ráfagas.
- Eliminar paquetes que superen un umbral de recepción.
- Establecer valores de QoS, valores de las celdas ATM, etc.

Si ya creamos una política anteriormente y lo que queremos es editarla, seleccionaremos primero aquella y después pulsaremos sobre el botón de editar. Se nos mostrará una ventana similar a la anterior pero con las condiciones especificadas en la política seleccionadas en la interfaz gráfica.

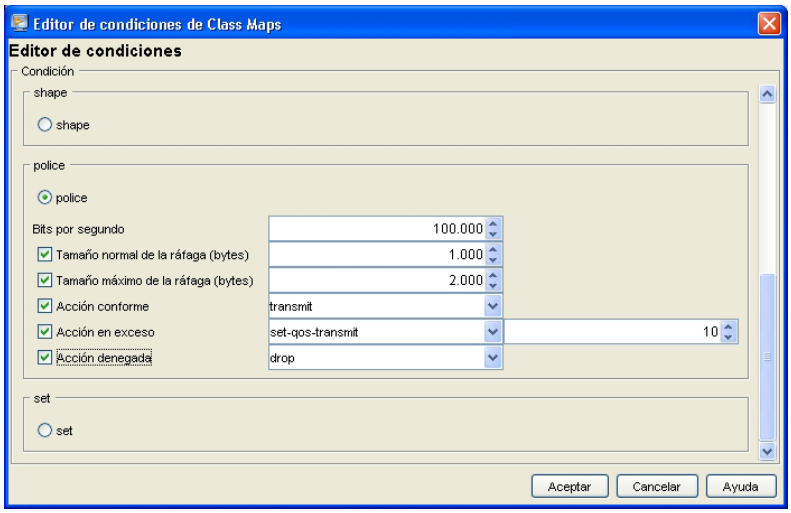


Figura 59 - Editar política

Para eliminar una política de la configuración de la clase, se seguirá el procedimiento actual, seleccionando la política a eliminar y después pulsando sobre el botón de eliminar política.

Figura 60 - Eliminar política

No todas las combinaciones de políticas son posibles, o hay algunas opciones que requieren de otras. En caso de que esto ocurra, el *router* devolverá un mensaje de error, y su respuesta será mostrada por pantalla:

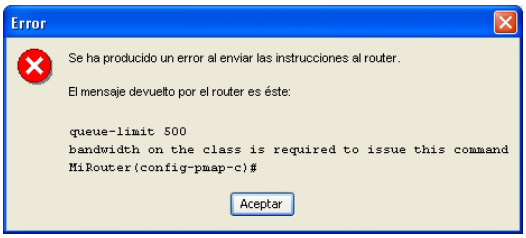


Figura 61 - Error al configurar una política

6.6.4 Asociar políticas a interfaces

Una vez que hemos creado los mapas de políticas que describen las acciones a ejercer sobre los paquetes que pertenezcan a una determinada clase, es necesario que asociemos esos mapas de políticas a las interfaces del *router*. Estas políticas pueden ser aplicadas bien cuando los paquetes entren por una cierta interfaz, bien cuando los paquetes salgan por la interfaz. Por esto, las interfaces pueden tener asociados hasta dos mapas de políticas.

Al pulsar en la ventana principal sobre el botón de asociación de políticas a interfaces, se nos mostrará la siguiente ventana:

Figura 62 - Asociación de políticas a interfaces

Seleccionando sobre el menú desplegable de la parte superior una interfaz, se seleccionarán automáticamente las políticas de entrada y salida que tiene asociadas, si las tuviera.

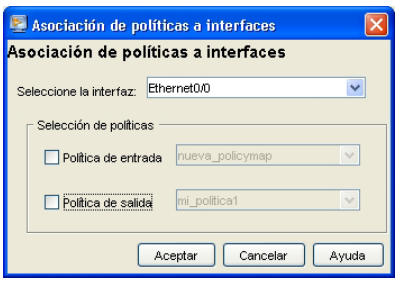


Figura 63 - Interfaz seleccionada

Una vez seleccionada la interfaz es posible desasociar una política asociada anteriormente (mediante la marca de verificación) o cambiarla.

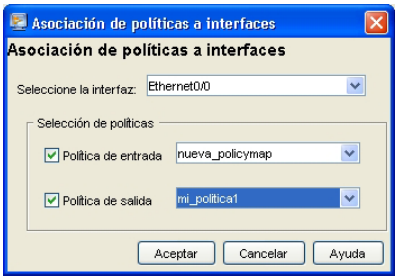


Figura 64 - Políticas asociadas

No todas las políticas son asociables tanto a la entrada como a la salida. Una vez que pulsemos el botón aceptar, se mandarán los comandos necesarios al *router*, pero éste no mostrará ningún mensaje de error inmediatamente. Tras esta operación deberemos observar en la ventana de terminal si aparece un mensaje como el siguiente:

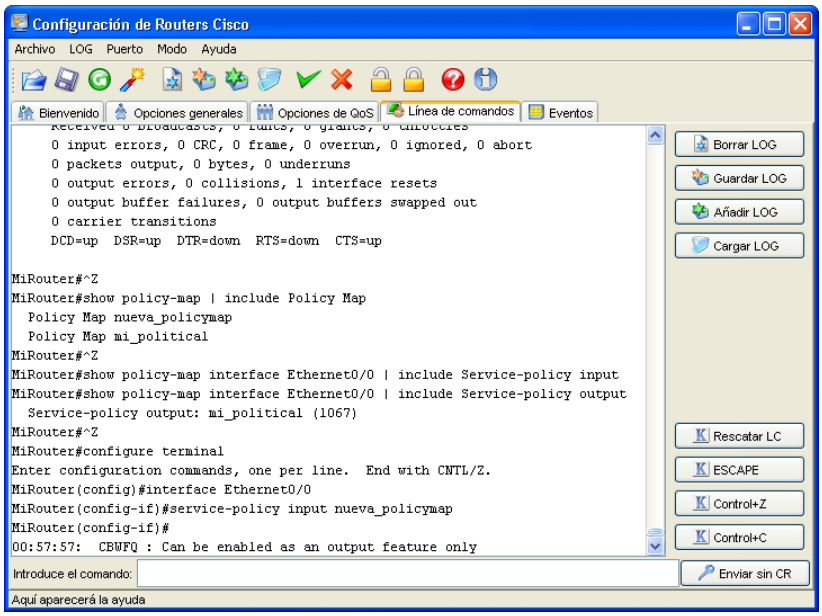


Figura 65 - Error de asociación

En este caso, el *router* nos indica que el mapa de políticas asociado como entrada (cuya configuración es aplicable como *Class Based Weighted Fair Queueing*, CBWFQ, ver apartado 3.6.4) sólo es posible aplicarlo como opción de salida, cuando en realidad lo hemos intentado asociar como política de entrada.

6.7 Ventana del Terminal

En la cuarta pestaña de la ventana principal se encuentra un pequeño terminal desde el cual es posible enviar comandos al *router*, además de secuencias especiales de teclas y poder trabajar con la ventana del historial.

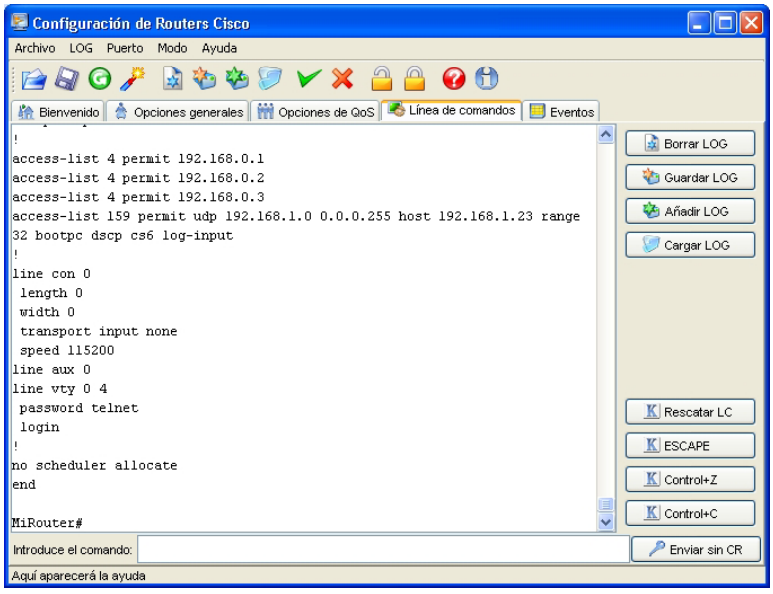


Figura 66 - Terminal

A continuación se describe la funcionalidad de cada uno de los componentes, tanto botones como cajas de texto, de los que se compone esta pestaña:

- Borrar LOG. Limpia la ventana del terminal.

- Guardar LOG. Muestra una ventana para seleccionar el archivo en el cual se quiere guardar el contenido actual de la ventana de terminal.
- Añadir LOG. Muestra un diálogo para seleccionar un archivo de texto, y añade su contenido al contenido actual de la ventana del terminal.
- Cargar LOG. Muestra un diálogo para seleccionar un archivo de texto, y establece el contenido del archivo como contenido de la ventana del terminal.
- Rescatar línea de comandos. En ocasiones el *router* pasa a un estado de reposo en el cual no es posible enviar comandos. Pulsando este botón se cierra el puerto y se vuelve a abrir, obteniendo de nuevo el prompt del *router*.
- Escape. Manda una señal de escape al *router*. Esta combinación de teclas es útil cuando, por ejemplo, se ha mandado un ping a una dirección desconocida y el *router* se queda en espera de una respuesta durante un tiempo prolongado.
- Control+Z. Manda una señal de Control+Z al *router*.
- Control+C. Manda una señal de Control+C al *router*, Interrumpe cualquier proceso que se esté ejecutando en el *router*.

6.8 Tabla de Eventos

La última pestaña del programa muestra una tabla donde se van recogiendo todos los eventos que se generan internamente en el programa.

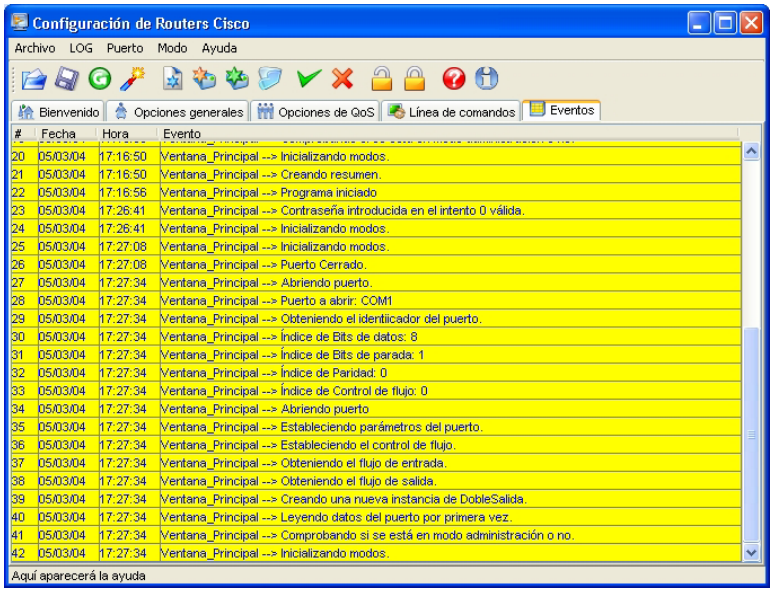


Figura 67 - Tabla de eventos

Puede ser útil a la hora de depurar el programa y encontrar errores en la interpretación de las respuestas del *router* ante los comandos enviados.

6.9 Ejemplo de configuración

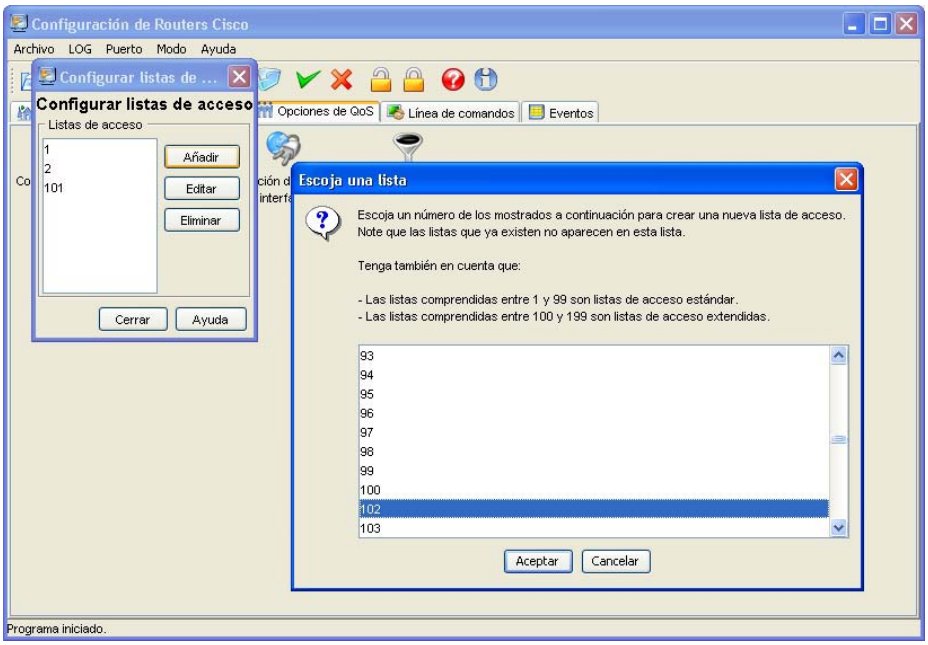
En el siguiente apartado se mostrará como realizar una configuración de ejemplo en el *router*. La configuración reservará un ancho de banda para aquellos paquetes cuyo protocolo sea de tiempo real y que además provengan de la red 192.168.100.0. A dichos paquetes se les reservará, como ya se ha dicho, un ancho de banda mínimo garantizado. El nuevo mapa de

políticas creado, se asociará a la interfaz Ethernet 0/1, que estará conectada a Internet (los paquetes procesados suponemos que entrarán por la interfaz Ethernet 0/0).

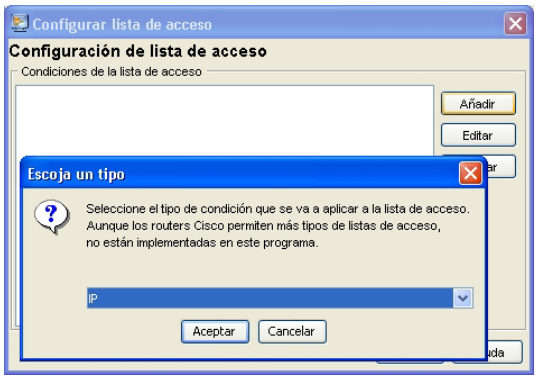
En primer lugar crearemos una nueva lista de acceso extendida IP mediante la cual indicaremos que los paquetes que provengan de la red 192.168.100.0 (con máscara de red 255.255.255.0) y cuyo destino sea cualquiera, pasen a formar parte de ella.

En primer lugar, nos aseguraremos de que el puerto está abierto y nos encontramos en modo de administración, así como que el puerto está bien configurado. Para ello simplemente echaremos un vistazo a la ventana del terminal y comprobaremos que los datos recibidos son coherentes.

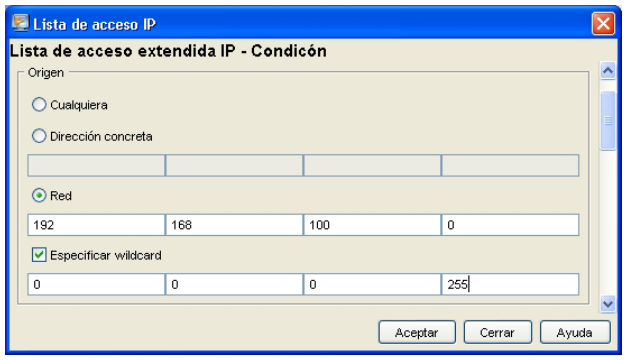
A continuación, crearemos una nueva lista de acceso extendida, pulsando el botón "Configuración de las listas de acceso" de la tercera pestaña. Tras ello, pulsaremos sobre el botón añadir y seleccionaremos un número mayor o igual que 101, correspondiente a las lista de acceso IP extendidas.

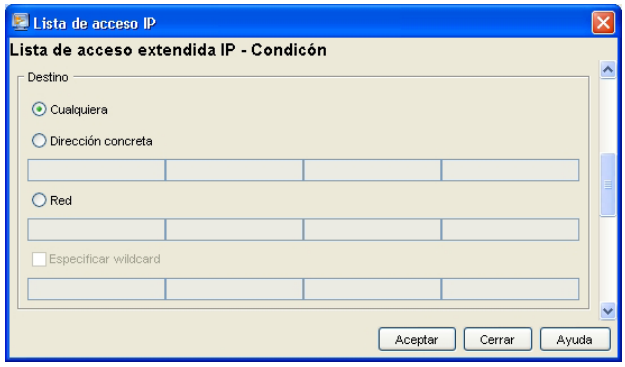


En la nueva ventana que nos aparecerá pulsaremos sobre el botón añadir y seleccionaremos la opción IP en el diálogo que se nos muestra.



En la nueva ventana mostrada seleccionaremos como dirección origen la de la red mencionada anteriormente, y como red de destino seleccionaremos la opción que indica "Cualquiera".



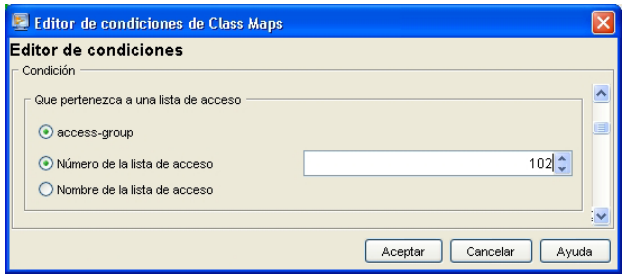


Tras esto, la ventana con las condiciones de la lista de acceso debe quedar como sigue:

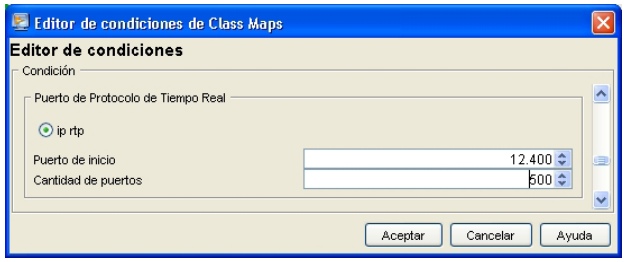
Cerraremos todas las ventanas pulsando el botón de Aceptar o de Cerrar según el caso.

El siguiente paso es crear un mapa de clases donde especifiquemos que sólo pasarán a formar parte de esa clase aquellos paquetes que pertenezcan a la lista de acceso 102 (recién creada) y que además hayan entrado por la interfaz Ethernet 0/0. Para ello, pulsaremos sobre el botón “Configuración de Class Maps”, situado en la pestaña de “Opciones de QoS”. Una vez abierta la ventana, pulsaremos sobre el botón “Añadir nueva clase”, donde escribiremos en el nuevo diálogo el nombre de la clase; en este caso será “clase_voz”.

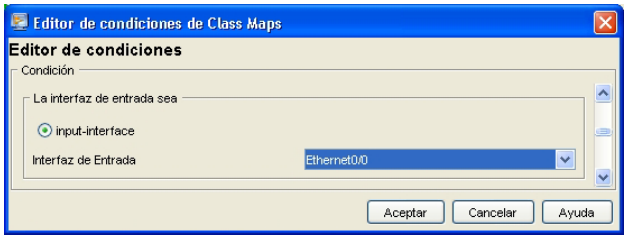
Tras esto, pulsaremos el botón para añadir una condición. La primera de ellas será la referente a la coincidencia con la lista de acceso. Para ello, en la nueva ventana abierta seleccionaremos la opción *access-group*, seleccionando la lista mediante un número y se pulsará después el botón de Aceptar.



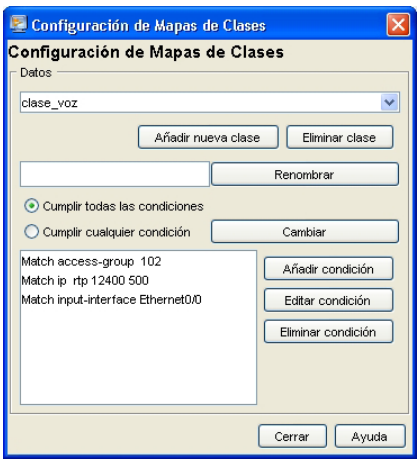
La siguiente condición a añadir es la referente a los puertos del protocolo de tiempo real. La opción a seleccionar en este caso será la indicada por *ip rtp*, indicando como puerto de inicio el 12400 y como cantidad de puertos 500 (estos datos son ficticios); de este modo, cualquier paquete del protocolo RTP cuyo puerto esté comprendido entre el 12400 y el 12900 pasará a formar parte de la clase si cumple también las otras condiciones.



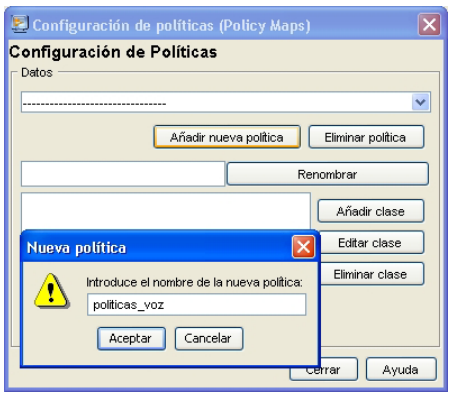
Tras ello, pulsaremos el botón de aceptar, y añadiremos una nueva condición, en este caso, la referente a la interfaz de entrada. Para ello seleccionaremos la opción *input-interface*, seleccionando la interfaz Ethernet 0/0.



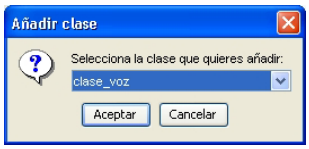
Una vez añadidas todas las opciones, la lista de condiciones debería quedar como sigue.



El siguiente paso será especificar las acciones a realizar con los paquetes que pertenezcan a la clase configurada anteriormente. Para ello pulsaremos el botón de la tercera pestaña cuyo texto es “Configuración de Policy Maps”. Una vez abierta la ventana, pulsaremos el botón para añadir una nueva política, donde se nos preguntará el nombre del nuevo mapa de políticas:

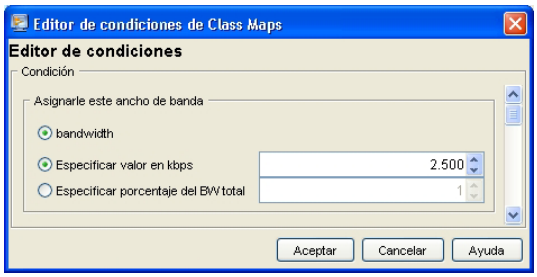


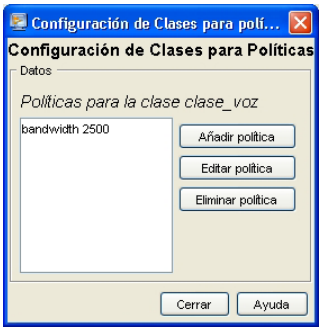
A continuación debemos añadir el mapa de clases creado en el paso anterior para que pase a formar parte del mapa de políticas y poder especificar las acciones a emprender con los paquetes.



Quedando la ventana con este aspecto:

A continuación seleccionaremos la clase recién añadida y pulsaremos el botón Editar. Una vez abierta la nueva ventana pulsaremos sobre Añadir política. En este punto seleccionaremos la opción bandwith, precisando un ancho de banda de 2500 kbps. A continuación pulsaremos el botón de Aceptar de la ventana de políticas, y el botón cerrar de la ventana de la configuración de mapas de políticas.





El último paso consiste en asociar el mapa de políticas a una interfaz en concreto. Para ello pulsaremos la opción “Asociación de políticas a interfaces”, situado en la ventana principal del programa.

En esta ventana seleccionaremos la interfaz Ethernet 0/1, y especificaremos como política de salida el mapa de políticas “politicas_voz”, que acabamos de crear, y se pulsará el botón de Aceptar.

Para concluir, si queremos que la configuración se mantenga tras un reinicio del *router*, pincharemos sobre el menú Archivo y a continuación sobre la opción Guardar configuración como inicial.

Bibliografía

Java

[J1] Página oficial de Java
http://java.sun.com

[J2] Java™ 2 Platform, Standard Edition, v 1.4.2 API Specification
http://java.sun.com/j2se/1.4.2/docs/api/

[J3] Java Communications API
http://java.sun.com/products/javacomm/

[J4] Guía de iniciación al lenguaje Java
http://pisuerga.inf.ubu.es/lsi/Invest/Java/Tuto/Index.htm

[J5] Borland JBuilder 9, Personal Edition
http://www.borland.es/jbuilder/index.html

[J6] Fundamentos del lenguaje Java
http://ants.dif.um.es/~humberto/asignaturas/cursojava/fundamentos/fundamentos.html

[J7] RXTX serial and parallel I-O libraries supporting Sun's CommAPI
http://www.rxtx.org/

[J8] Java Tutorial
http://java.sun.com/docs/books/tutorial/index.html

[J9] JavaWorld - Set your table options -- at runtime!
http://www.javaworld.com/javaworld/javatips/jw-javatip116.html

[J10] Java Comm Serial API How-To for Linux
http://wass.homelinux.net/howtos/Comm_How-To.shtml

Cisco

[C1] Cisco Company Overview
http://newsroom.cisco.com/dlls/company_overview.html

[C2] Router multiservicio modular de la serie Cisco 2600 – Hoja de datos

http://www.cisco.com/warp/public/cc/pd/rt/2600/prodlit/_sp_2600_ds.htm

[C3] Academia de networking de Cisco Systems. Guía del primer año

Pearson Educación, S.A. ISBN: 84-205-3296-7

[C4] Cisco IOS QoS Technology

http://www.cisco.com/en/US/products/sw/iosswrel/ios_abcs_ios_networking_the_enterprise0900aecd800a4df9.html

[C5] Modular Quality of Service Command-Line Interface Overview-Cisco IOS Software Releases 12.2

http://www.cisco.com/en/US/products/sw/iosswrel/ps1835/products_configuration_guide_chapter09186a00800bd908.html

[C6] Class-Based Weighted Fair Queueing -Cisco IOS Software Releases 12.0 T - Cisco Systems

http://www.cisco.com/en/US/products/sw/iosswrel/ps1830/products_feature_guide09186a0080087a84.html

[C7] Configuring Class-Based Packet Marking-Cisco IOS Software Releases 12.2 Mainline - Cisco Systems

http://www.cisco.com/en/US/products/sw/iosswrel/ps1835/products_configuration_guide_chapter09186a00800c75cf.html

[C8] Configuring Class-Based Shaping-Cisco IOS Software Releases 12.2 Mainline - Cisco Systems

http://www.cisco.com/en/US/products/sw/iosswrel/ps1835/products_configuration_guide_chapter09186a00800bd8f0.html

[C9] Configuring Traffic Policing-Cisco IOS Software Releases 12.2 Mainline - Cisco Systems

http://www.cisco.com/en/US/products/sw/iosswrel/ps1835/products_configuration_guide_chapter09186a00800bd8ee.html

[C10] Configuring Weighted Fair Queueing-Cisco IOS Software Releases 12.2 Mainline - Cisco Systems

http://www.cisco.com/en/US/products/sw/iosswrel/ps1835/products_configuration_guide_chapter09186a00800b75af.html

[C11] DiffServ Compliant Weighted Random Early Detection-Cisco IOS Software Releases 12.1 T - Cisco Systems

http://www.cisco.com/en/US/products/sw/iosswrel/ps1834/products_feature_guide09186a008008043f.html

[C12] Low Latency Queueing-Cisco IOS Software Releases 12.0 T - Cisco Systems

http://www.cisco.com/en/US/products/sw/iosswrel/ps1830/products_feature_guide09186a0080087b13.html

[C13] CiscoWorks CiscoView Dataste 6.0

http://www.cisco.com/warp/public/cc/pd/wr2k/view/prodlit/cvv50_ds.htm

[C14] Cisco ConfigMaker 2.6
http://www.cisco.com/en/US/products/sw/netmgtsw/ps754/index.html

[C15] Interconexión de dispositivos de red Cisco
Steve McQuerry; Cisco, D.L. 2001; ISBN: 84-205-3189-8

[C16] Manual de CISCO
Tom Shaughnessy y Toby Velte; Mc Graw Hill , 2000; 84-481-2727-7