HERRAMIENTA SOFTWARE PARA LA SIMULACIÓN DE PROTOCOLOS DE ENRUTAMIENTO EN REDES IP

ARNOL ANDRÉS CHILITO CASTRO
VÍCTOR HUGO PAZ OCAMPO

Anexos A, B , C y D

Director: Mag. Ing. Francisco Javier Terán

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Popayán
2003

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

ANEXO A. DIRECCIONAMIENTO IP

Una dirección IP es un número jerárquico de dos niveles y 32 bits. Es jerárquico porque la primera parte de la dirección representa la red, mientras que la segunda representa el nodo (host).
Los 32 bits se encuentran agrupados en cuatro octetos con 8 bits por octeto. El valor de cada octeto va de 0 a 255 decimal, o de 00000000 a 11111111 en binario. Las direcciones IP suelen estar escritas con notación decimal de puntos (cada uno de los cuatro octetos está escrito con notación decimal, y los puntos se colocan entre los octetos). La figura A.1 ilustra cómo se convierte un octeto de una dirección IP binaria a notación decimal.

Valor de cada bit

1	1	1	1	1	1	1	1	
128	64	32	16	8	4	2	1	=255

Conversión de binario a decimal

0	1	0	0	0	0	0	1	
128	64	32	16	8	4	2	1	
0	64	0	0	0	0	0	1	=65

Figura A.1 Cómo convertir un octeto de una dirección IP de binario a decimal

1. Clases de direcciones IP

Para adaptar redes grandes y pequeñas, el Centro de información de red (NIC) segrego la dirección IP de 32 bits en clases de la A a la E. Los primeros bits del primer octeto determinan la clase de una dirección; y a su vez, determina cuántos bits de red y bits de host se encuentran en la dirección. Esto se ilustra en las direcciones de Clases A, B y C de la figura A.2. En consecuencia, cada clase permite que haya un número determinado de direcciones de red y un número determinado de direcciones de host en una red.

Figura A.2 Clases de direcciones IP

Clase A

Cuando está escrito en formato binario, el primer bit (el bit que está ubicado más a la izquierda) de la dirección de Clase A siempre es 0. Un ejemplo de una dirección IP de clase A es 124.95.44.15. Una manera fácil de reconocer si un dispositivo forma parte de una red de Clase A es verificar el primer octeto de su dirección IP, cuyo valor debe estar entre 0 y 126. (127 *comienza* con un bit 0, pero está reservado para fines especiales).
Todas las direcciones IP de Clase A utilizan solamente los primeros 8 bits para identificar la parte de la red de la dirección. Los tres octetos restantes se pueden utilizar para la parte del host de la dirección. A cada una de las redes que utilizan una dirección IP de Clase A se les pueden asignar hasta 2 elevado a la 24 potencia (2^{24}) (menos 2), o 16.777.214 direcciones IP posibles para los dispositivos que están conectados a la red.

Clase B

Los primeros 2 bits de una dirección de Clase B siempre son 10. Un ejemplo de una dirección IP de Clase B es 151.10.13.28. Las direcciones IP de Clase B siempre tienen valores que van del 128 al 191 en su primer octeto.
Todas las direcciones IP de Clase B utilizan los primeros 16 bits para identificar la parte de la red de la dirección. Los dos octetos restantes de la dirección IP se encuentran reservados para la porción del host de la dirección. Cada red que usa un esquema de direccionamiento IP de Clase B puede tener asignadas hasta 2^{16} (menos 2), o 65.534 direcciones IP posibles a dispositivos conectados a su red.

Clase C

Los 3 primeros bits de una dirección de Clase C siempre son 110. Un ejemplo de dirección IP de Clase C es 201.110.213.28. Las direcciones IP de Clase C siempre tienen valores que van del 192 al 223 en su primer octeto.
Todas las direcciones IP de Clase C utilizan los primeros 24 bits para identificar la porción de red de la dirección. Sólo se puede utilizar el último octeto de una dirección IP de Clase C para la parte de la dirección que corresponde al host. A cada una de las redes que utilizan una dirección IP de Clase C se les pueden asignar hasta 2^8 (menos 2), o 254, direcciones IP posibles para los dispositivos que están conectados a la red.

Clase D

Están definidas como las redes con los cuatro primeros bits que comiencen con 1110**.** Estas redes no representan una máquina sino una colección que forma parte de un grupo

denominado multicast IP. Comprende las direcciones de red desde la 224.0.0.0 hasta la 239.255.255.255.

Clase E

Las redes clase E, comienzan con sus cinco primeros bits en 11110 y están compuestas por las redes comprendidas desde la 240.0.0.0 hasta la 254.255.255.255. Estas direcciones de red están reservadas para uso futuro
La Tabla 1 muestra el intervalo de direcciones, el número de redes y el número de host de cada una de las Clases (observe que las direcciones de Clase D y E se utilizan para otros fines, no para dirigirse a los hosts).

Clase	Intervalo de dirección	Número de redes	Número de Host
Clase A	1.0.0.0 a 126.0.0.0	$126(2^7-2)$	$16.777.214\ (2^{24} - 2)$
Clase B	128.0.0.0 a 191.255.0.0	$16.382\ (2^{14} - 2)$	$65.534\ (2^{16} - 2)$
Clase C	192.0.0.0 a 223.255.255.255	$2.097.150\ (2^{21} - 2)$	$254\ (2^{8} - 2)$
Clase D	224.0.0.0 a 239.255.255.255	Reservado para direcciones de multidifusión	
Clase E	240.0.0.0 a 254.255.255.255	Reservado para usos futuros	

Tabla A.1 Intervalo de direcciones IP

2. Identificadores de Red y direcciones Broadcast

Una dirección IP que termina en 0 binarios en todos los bits de host se reserva para la dirección de red. Por ejemplo la dirección de clase B 176.10.0.0 es una dirección de red, dado que los últimos dos octetos que corresponden a los números de host tienen 0, y se utilizan para los dispositivos que están conectados a la red.
La dirección de red nunca se usará como dirección para un dispositivo conectado a ella y es usada por los enrutadores para enviar datos en Internet.

Si necesita enviar datos a todos los dispositivos de la red necesitará una dirección de broadcast. Esta dirección terminan con unos binarios en toda la parte de la dirección que corresponde al host. Por ejemplo para la red 172.16.0.0 la dirección de broadcast sería 172.10.255.255 (ya que 255 es el valor decimal de un octeto que contiene 11111111)

3. Máscaras de Subred

Para afrontar el problema del agotamiento de direcciones IP se introdujo un procedimiento, denominado enmascaramiento de subred para dividir las direcciones de Clase A B y C en partes más pequeñas, incrementando así el numero de redes posibles. Una máscara de subred es un valor de 32 bits que identifica qué bits de una dirección representan los bits de red y cuáles representan los bits de host.
En otras palabras, el router no determina la parte de red de la dirección examinando el valor del primer octeto; examina la máscara de subred asociada a la dirección. De esta forma, la máscara de subred permite ampliar el uso de una dirección IP. Esta es una forma de hacer de una dirección IP una jerarquía de tres niveles, como se ve en la figura A.3.

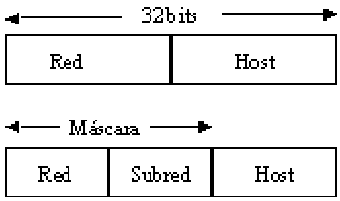

Figura A.3 Mascaras de subred

Para crear una máscara de subred en una dirección, utilice un 1 en cada bit que quiera representar una parte de red o subred de la dirección, y utilice un 0 en cada bit que quiera representar la parte de host de la dirección. Observe que los números de las marcas son contiguos. Las máscaras de subred predeterminadas de las clases A, B y C se muestran en la tabla A.2.

Clase	Máscara predeterminada en binario	Máscara predeterminada en decimal
Clase A	11111111.00000000. 00000000. 00000000	255.0.0.0
Clase B	11111111. 11111111. 00000000. 00000000	255.255.0.0
Clase C	11111111. 11111111. 11111111. 00000000	255.255.255.0

Tabla A.2 Mascara de red para las direcciones IP

Añadir bits a la parte de red de una dirección reduce el número de bits en la parte de host. Por consiguiente, la creación de redes adicionales (subredes) se hace a costa del número de dispositivos host que pueden ocupar cada segmento de red.
El número de subredes que se crea se calcula con la fórmula $2^n - 2$, donde n es el número de bits con los que se ha ampliado la máscara predeterminada y el –2 se debe a las direcciones reservadas de red y de broadcast.
Las direcciones de host se seleccionan a partir de los bits restantes de la máscara y deben ser numéricamente distintos de todos los demás host de la subred. El número de host que se

crea se calcula con la fórmula $2^n - 2$, donde n es el número de bits disponibles en la parte del host.
Dado que las máscaras de subred amplían el número de direcciones de red que se pueden usar empleando bits de la parte del host, no conviene que decida al azar el número adicional de bits que va usar en la parte de red. Para establecer la máscara de subred se debe tener en cuenta:

- Determinar el número de redes necesarias.
- Determinar el número de host por red (subred).
- Determinar futuros requisitos de red y de host.

Ejemplo de subredes

Suponga que tiene asignada la dirección clase C 200.133.175.0 y necesita dividirla en 14 subredes para diferentes grupos dentro de una organización.
Por tal motivo necesitará por lo menos 4 bits de la parte de host para crear un prefijo de red para cada una de las subredes y los otros 4 bits restantes serán utilizados para identificar los hosts.
La máscara de una dirección clase C es 255.255.255.0 y aumentando 3 bits más quedaría 255.255.255.240 (11111111.11111111.11111111.11110000).
Las subredes creadas se muestran en la tabla A.3.

Bits de subred	Número de red	Direcciones de host	Direccioned de broadcast
0000	200.133.175.0	Reservada	
0001	200.133.175.16	.17 hasta .30	200.133.175.31
0010	200.133.175.32	.33 hasta .46	200.133.175.47
0011	200.133.175.48	.49 hasta .62	200.133.175.63
0100	200.133.175.64	.65 hasta .78	200.133.175.79
0101	200.133.175.80	.81 hasta .94	200.133.175.95
0110	200.133.175.96	.97 hasta .110	200.133.175.111
0111	200.133.175.112	.113 hasta .126	200.133.175.127
1000	200.133.175.128	.129 hasta .142	200.133.175.143
1001	200.133.175.144	.145 hasta .158	200.133.175.159
1010	200.133.175.160	.161 hasta .174	200.133.175.175
1011	200.133.175.176	.177 hasta .190	200.133.175.191

1100	200.133.175.192	.193 hasta .206	200.133.175.207
1101	200.133.175.208	.209 hasta .222	200.133.175.223
1110	200.133.175.224	.225 hasta .238	200.133.175.239
1111	200.133.175.240	Reservada	

Tabla A.3 Ejemplo de subredes

4. Operación booleana AND

Para enrutar un paquete de datos, el router primero debe determinar la dirección de subred/red destino ejecutando una operación AND lógica utilizando la dirección IP del host destino y la máscara de subred para esa red. El resultado será la dirección de subred/red, que es la que utiliza el router para determinar cómo debe enviar el paquete.
Supongamos que tiene una red de Clase B, con el número de red 172.16.0.0. Después de analizar las necesidades de la red, decide pedir prestados 8 bits para crear subredes. Si pide prestados 8 bits en una red de Clase B, la máscara de subred es 255.255.255.0.
Alguien, desde fuera de la red, envía datos a la dirección IP 172.16.2.120. A fin de determinar dónde enviar los datos, el router realiza la operación AND con esta dirección y la máscara de subred. Cuando se ha realizado la operación AND de los dos números, la porción del host del resultado siempre es 0. Lo que resta es el número de red, incluyendo la subred. De este modo, los datos se envían a la subred 172.16.2.0 y solo el último router se da cuenta de que el paquete debe enviarse hacia el host 120 de esa subred.

ANEXO B. DISEÑO DE LA HERRAMIENTA

1. Modelado de los Protocolos usando CPN

1.1 Modelado del protocolo RIP usando las CPN

En la figura B.1 se presenta el diagrama básico del protocolo RIP, el cual presenta dos estados, Transmisor y Receptor. El primero de ellos modela el protocolo para la transmisión de datos hacia otros enrutadores que estén corriendo RIP, el segundo estado modela la recepción y el tratamiento de dichos datos. Ambos estados acceden a un buffer que contiene la información necesaria para la realización de sus funciones.

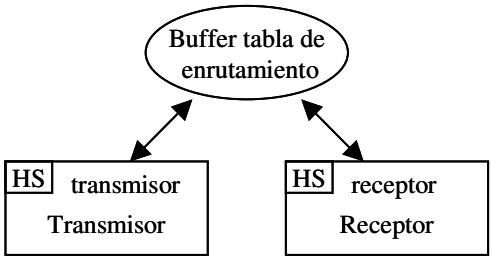

Figura B.1 Modelo básico

El estado Transmisor el cual de acuerdo a la terminología de CPN es una transición sustituta (*sustitution transition)*, esta detallado en la subpágina transmisor que se presenta en la figura B.2.

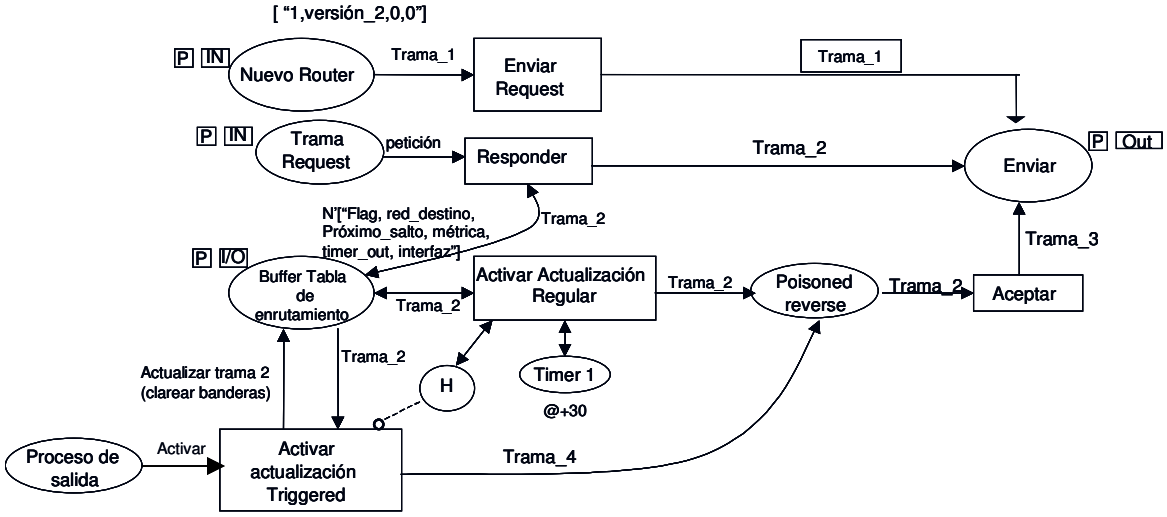

Figura B.2 Subpágina transmisor

El transmisor esta modelado por 7 estados y 5 transiciones. Los cuales se describirán a continuación:

Estado	Nuevo Enrutador
Transición	Enviar Request
Nuevo estado	Enviar
Descripción	Una vez se ponga en marcha el protocolo por parte del usuario un token del tipo tramaRequest aparecerá en el estado *Nuevo Enrutador* habilitando la transición *Enviar Request*. Esta transición estará encargada de enviar tramas request con el fin de solicitar la tabla de enrutamiento completa de los vecinos y poder actualizar su propia tabla de enrutamiento. Este proceso se realiza solamente una vez durante la inicialización del protocolo.

Estado	Trama Request
Transición	Responder
Nuevo estado	Enviar
Descripción	Cuando llega una trama Request al enrutador, este debe responder con su tabla de enrutamiento completa en una trama Response, este proceso es modelado por la transición *Responder* la cual accede al buffer *tabla de enrutamiento* para obtener los datos requeridos, arma la trama y la pone en el estado *Enviar*.

Estado	Timer 1
Transición	Actualización Regular
Nuevo estado	Poisoned Reverse
Descripción	El protocolo Rip cada 30 segundos debe enviar una trama de actualización que contiene toda la tabla de enrutamiento. Este procedimiento esta modelado por el estado *Timer 1* el cual activa la transición *Actualización Regular* que accede al buffer de la tabla de enrutamiento para obtener los datos requeridos y pasarlos al estado *Poisoned Reversed*.

Estado	Poisoned Reverse
Transición	Aceptar
Nuevo estado	Enviar
Descripción	Para evitar las mallas de enrutamiento, el protocolo Rip implementa el poisoned Reverse el cual consiste en no enviar la información aprendida de un vecino al mismo vecino. Este proceso es llevado a cabo en la transición *Aceptar* y los resultados son llevados al estado

	Enviar.

Estado	Proceso de salida
Transición	Actualización Triggered
Nuevo estado	Poisoned Reverse
Descripción	Otro mecanismo utilizado por Rip para evitar las mallas de enrutamiento son las actualizaciones activadas o triggered, así una vez llega una actualización, un token aparecerá en el estado *Proceso de Salida* activando la transición *Activar* A*ctualización Triggered* enviando los datos hacia el estado *Poisoned Reverse*.

El Receptor esta modelado en la subpágina receptor que se presenta en la figura B.3. Este consta de 8 estados y 5 transiciones, dos de las cuales son transiciones sustitutas: Actualizar Buffer y Temporizar, y están detalladas en las subpáginas actualizar buffer y temporizar respectivamente.

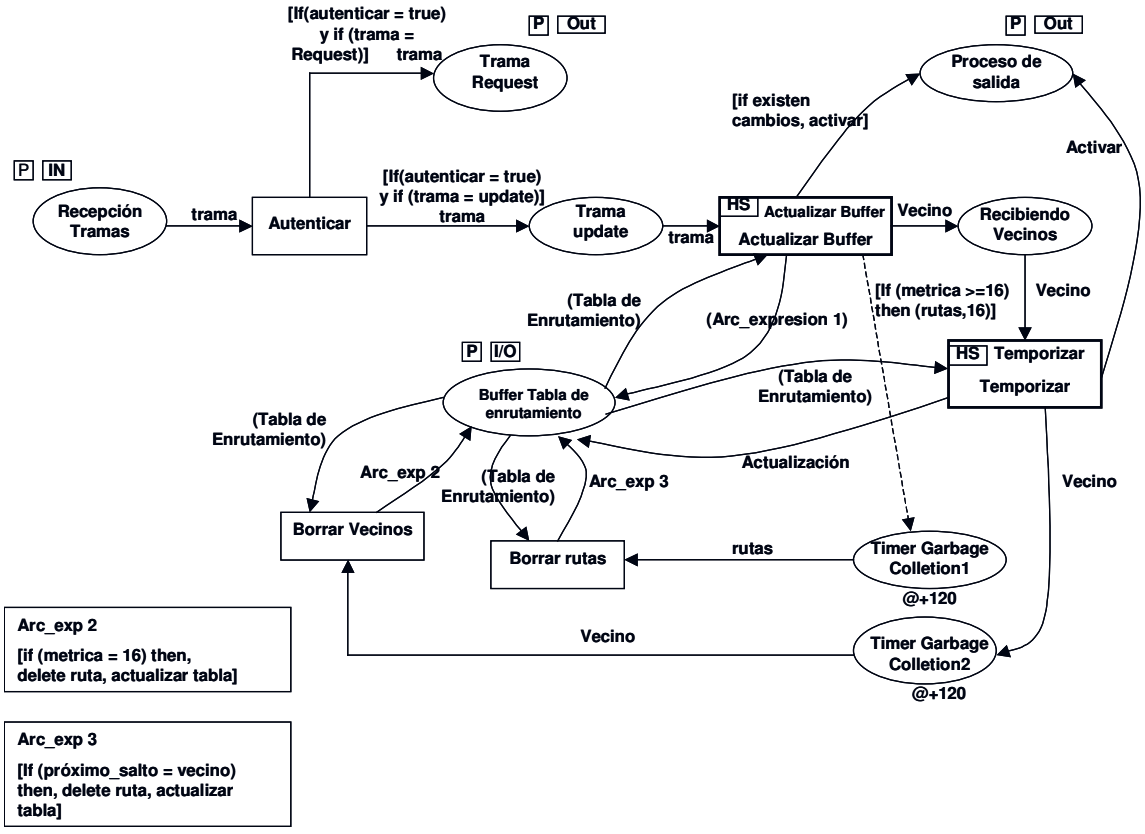

Figura B.3 Subpágina Receptor

Estado	Recepción tramas
Transición	Autenticar

Nuevo estado	Trama request o trama update
Descripción	Cuando llega una trama, un token aparece en el estado *Recepción Tramas* habilitando la transición *Autenticar*, la cual se encarga de autenticar y detectar el tipo de trama que llega, para así habilitar cualquiera de los dos estados *Trama Request* o *Trama Update*, el primer estado es un puerto de salida y fue modelado en la subpágina transmisor.

Estado	Trama update
Transición	Actualizar buffer
Nuevo estado	Proceso de salida, recibiendo vecinos, buffer tabla de enrutamiento, timer garbage colletion 1.
Descripción	La recepción de una trama update activa la transición *Actualizar Buffer* que será detallada en la subpágina del mismo nombre mas adelante.

Estado	Recibiendo vecinos
Transición	Temporizar
Nuevo estado	Timer garbage colletion 2
Descripción	La transición *Actualizar Buffer* envía un token del tipo vecino a la transición *Temporizar* con el cual la activa. Dicha transición se encarga de contar el tiempo transcurrido entre actualizaciones sucesivas de un vecino para percatarse cuando esta fuera de servicio y enviar un token al estado *Timer Garbage Colletion 2* con la información del vecino que salió del servicio.

Estado	Timer garbage colletion 2
Transición	Borrar vecinos
Nuevo estado	Buffer tabla de enrutamiento
Descripción	Una vez llaga un token con la información de vecino que ha salido del servicio al estado *Timer Garbage Colletion 2*, este temporiza 120 segundos, pasados los cuales activa la transición *Borrar Vecinos* que borra del buffer tabla de enrutamiento todas las rutas aprendidas del vecino que salió del servicio.

Estado	Timer garbage colletion 1
Transición	Borrar ruta
Nuevo estado	Buffer tabla de enrutamiento
Descripción	Una vez llega un token con la información de las rutas que tienen una

	métrica igual a 16 o superior al estado *Timer Garbage Colletion 1*, este temporiza 120 segundos, pasados los cuales activa la transición *Borrar Rutas*, que borra del buffer *tabla de enrutamiento* las rutas con dicha métrica.

La figura B.4 muestra la subpágina actualizar buffer que modela la transición sustituta Actualizar Buffer. Esta red CPN esta formada por 6 estados y 2 transiciones que se explican a continuación.

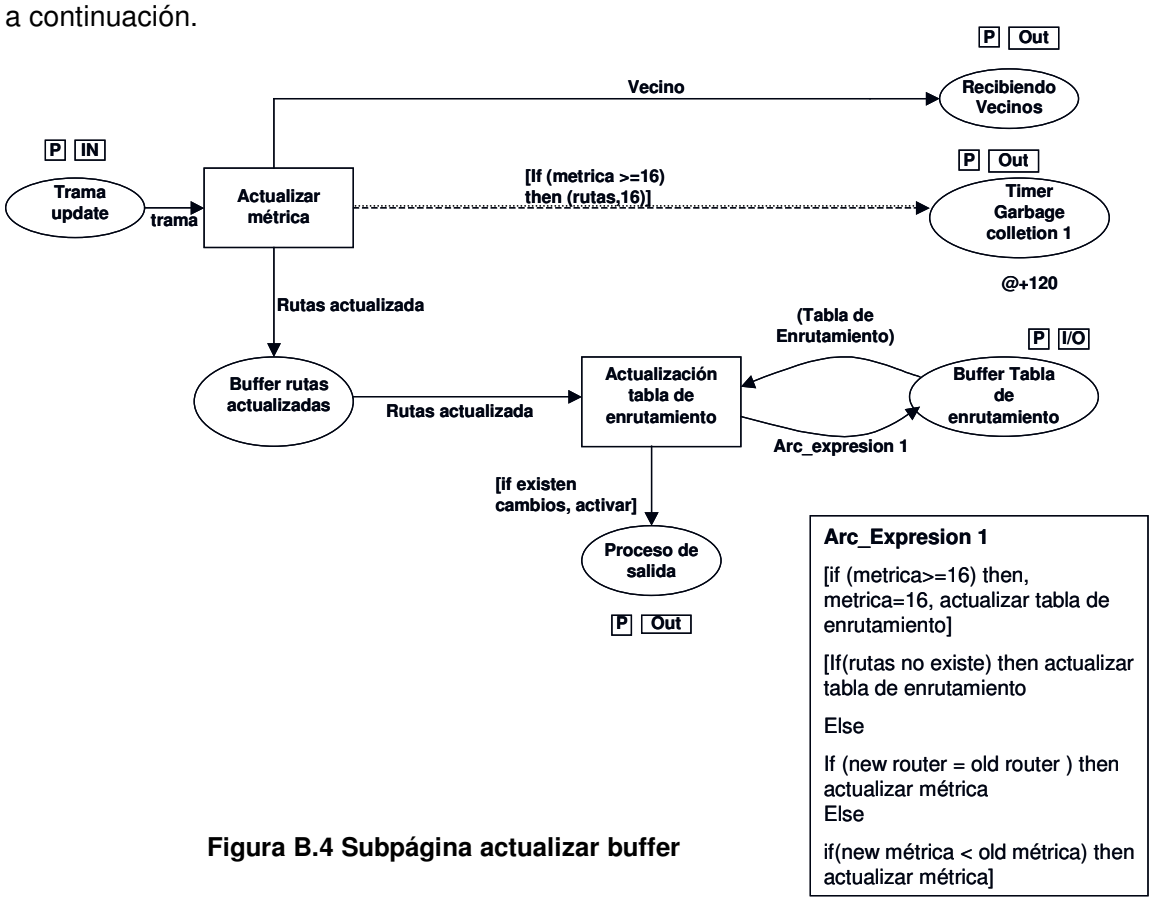

Figura B.4 Subpágina actualizar buffer

Estado	Trama update
Transición	Actualizar métrica
Nuevo estado	Buffer rutas actualizadas y recibiendo vecinos
Descripción	La presencia de una trama update en el estado *Trama Update* activa la transición *Actualizar Métrica*, la cual se encarga de incrementar en 1 la métrica de las rutas. Una vez actualizada la métrica esta transición pasa un token a dos estados: *Recibiendo Vecinos*, que fue explicado anteriormente, y *Buffer Rutas Actualizadas*, el cual guardará las rutas aprendidas con la nueva métrica.

Estado	Trama update
Transición	Actualizar métrica
Nuevo estado	Timer garbage colletion 1.
Descripción	Si una vez actualizada la métrica el resultado es igual a 16 o mayor, la transición enviará un token al estado *Garbage Colletion 1* para iniciar el proceso de borrado de la ruta que se explicó anteriormente.

Estado	Buffer rutas actualizadas
Transición	Actualización tabla de enrutamiento
Nuevo estado	Buffer tabla de enrutamiento y proceso de salida
Descripción	El estado *Buffer Rutas Actualizadas* activará la transición *Actualización Tabla de Enrutamiento*, la cual se encarga de guardar en la tabla de enrutamiento las nuevas redes aprendidas, los cambios en redes ya existentes y de pasar un token al estado *Proceso de Salida* para activar las transiciones estudiadas en la subpágina transmisor.

La figura B.5 muestra la subpágina temporizar que modela la transición sustituta Temporizar. Esta red CPN esta formada por 5 estados y 2 transiciones que se explica a continuación.

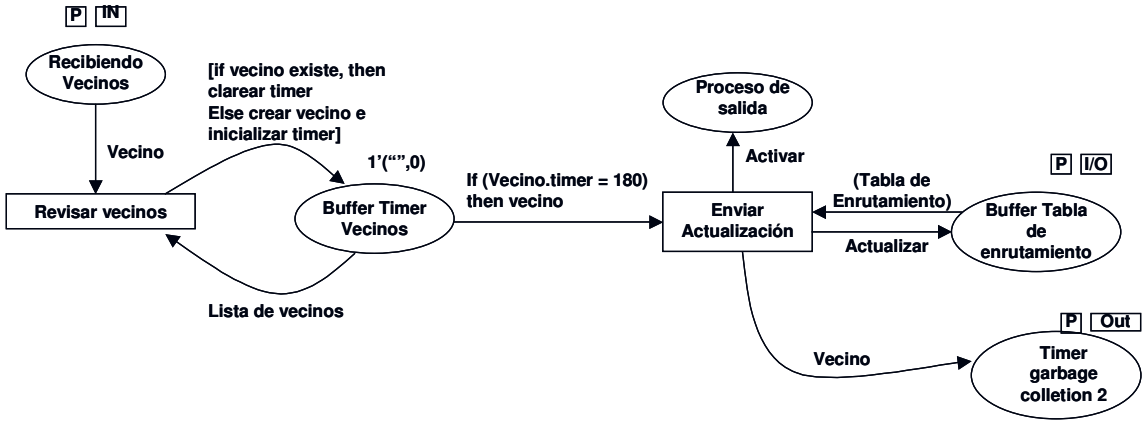

Figura B.5 Subpágina temporizar

Estado	Recibiendo vecinos
Transición	Revisar vecinos
Nuevo estado	Buffer timer vecinos
Descripción	El estado *Recibiendo Vecinos* habilita la transición *Revisar Vecinos*, la cual se encarga de mirar en el *Buffer Timer Vecinos* si ya se han recibido mensajes, en cuyo caso la transición clarea el timer; en caso contrario, la transición crea el timer para el nuevo vecino.

Estado	Buffer timer vecinos
Transición	Enviar actualización
Nuevo estado	Proceso de salida, buffer tabla de enrutamiento, timer garbage colletion 2
Descripción	El estado *Buffer Timer Vecinos* temporiza la llegada de paquetes de los vecinos, y cuando no recibe mensajes por un periodo de 180 segundos de un determinado vecino habilita la transición *Enviar Actualización*, la cual se encarga de enviar tokens a los estados *Proceso de Salida* y *Timer Garbage Colletion*, además de actualizar el buffer de la tabla de enrutamiento indicando las rutas que van a ser eliminadas.

1.2 Modelado del protocolo OSPF

El diagrama muestra la descripción general del modelo realizado para el protocolo OSPF. Esta dividido en siete transiciones, seis de la cuales son transiciones sustitutas que se detallan en las subpáginas con el mismo nombre de la transición y siete estados.

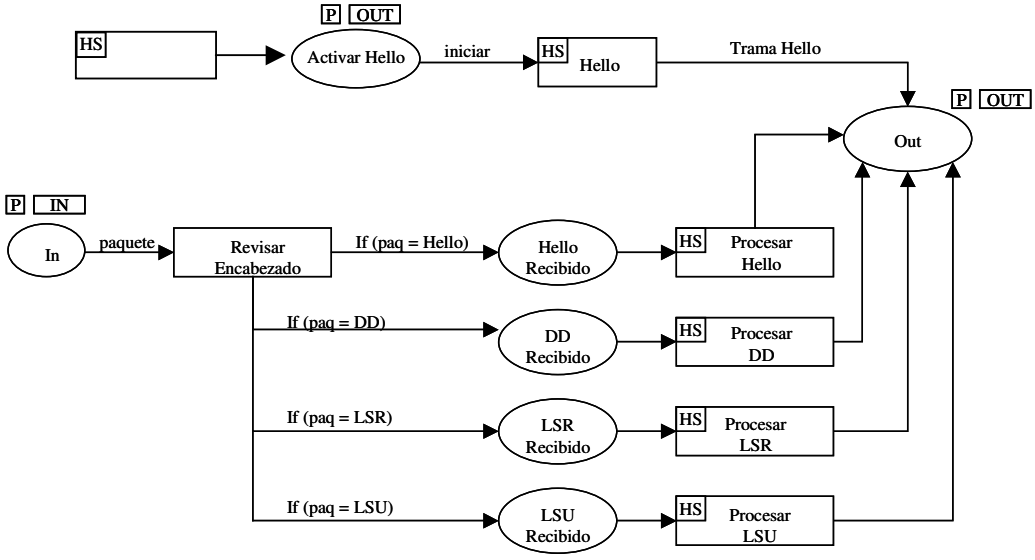

Figura B.6 Modelo básico protocolo OSPF

Estado	In
Transición	Revisar encabezado
Nuevo estado	Hello Recibido, DD Recibido, LSR Recibido, LSU recibido. (Depende del tipo de trama entrante) .
Descripción	El estado *In* recibe las tramas del medio de transmisión y activa la

	transición *Revisar Encabezado* que discrimina el tipo de trama entrante para poder enviarla al siguiente estado y realizar su procesamiento.

Estado	Activar Hello
Transición	Hello
Nuevo estado	Out
Descripción	Cuando el usuario activa una interfaz, el primer proceso que debe realizar el enrutador es el envío de paquetes Hello cada 10 segundos a través del medio de transmisión. Esta función esta representada en la transición sustituta *Hello*.

Los restantes estados y transiciones se explicarán en detalle cuando se describan las respectivas transiciones sustitutas.

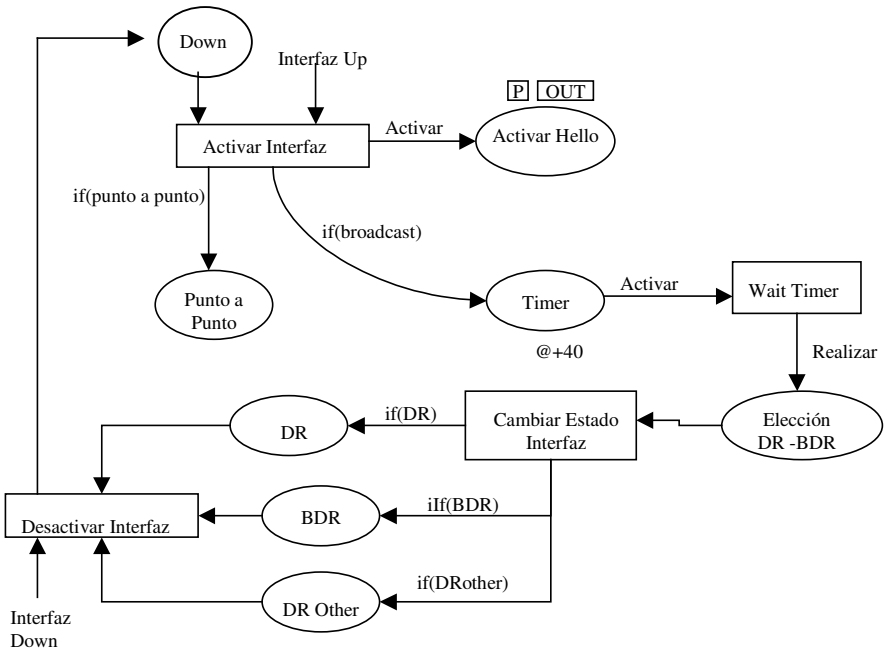

Figura B.7 Subpágina interfaz

Estado	Down
Transición	Activar interfaz
Nuevo estado	Activar hello, punto a punto, Timer
Descripción	Cuando una interfaz no se ha inicializado se encuentra en un estado *Down*. Cuando el usuario la inicialice, la transición *Activar interfaz*

	será la encargada de colocar en funcionamiento el envío de tramas Hello y dependiendo del tipo de interfaz (serial o ethernet) el nuevo estado será *punto a punto* o *Timer*.

Estado	Timer
Transición	WaitTimer
Nuevo estado	Elección DR – BDR
Descripción	Cuando una interfaz ethernet se coloca en funcionamiento debe esperar los paquetes hello de los vecinos para elegir el DR y el BDR. El protocolo especifica que debe esperar 40 segundos antes de realizar esta elección. Cuando se cumple el tiempo, la transición *WaitTimer* pasa el token al estado *Elección DR-BDR*.

Estado	Elección DR –BDR
Transición	Cambiar Estado
Nuevo estado	DR,BDR,DROther
Descripción	Cuando se realiza la selección del DR y el BDR debe cambiar el estado de la interfaz. Si un enrutador no quedo seleccionado como DR y tampoco como BDR su próximo estado será *DROther*.

Estado	DR, BDR, DROther
Transición	Desactivar Interfaz
Nuevo estado	Down
Descripción	Cuando el usuario desactive la interfaz, esta debe pasar nuevamente al estado *Down*.

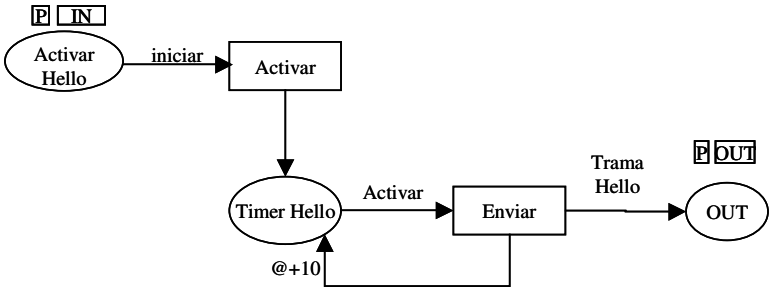

Figura B.8 Subpágina Hello

Estado	Timer Hello
Transición	Enviar
Nuevo estado	Out
Descripción	Cuando se activa la interfaz, periódicamente se deben enviar paquetes Hello. Cuando el timer expira activa la transición *Enviar* que envía el paquete al medio de transmisión. Como se puede analizar en la figura, el token regresa nuevamente al estado *Timer Hello* y el proceso se repite infinitas veces.

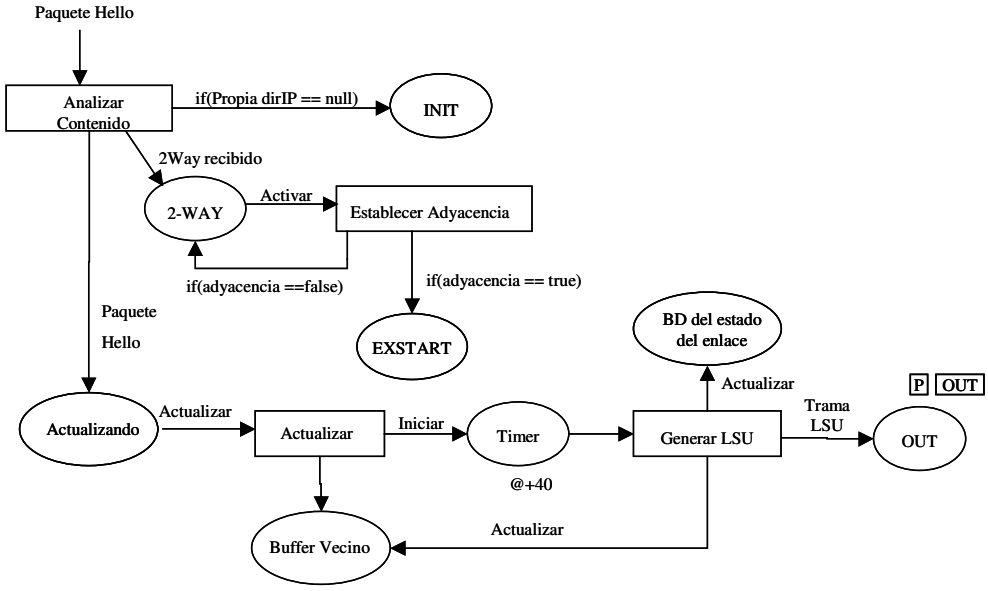

Figura B.9 Subpágina procesar Hello

Estado	Hello recibido
Transición	Analizar Contenido
Nuevo estado	Init, 2Way, Actualizando
Descripción	Cuando llega un paquete Hello el enrutador revisa el contenido, si el enrutador que envió el paquete no se encuentra en la lista de vecinos su estado será *INIT*, si se encuentra su próximo estado será *2WAY*. Finalmente siempre debe actualizar su lista de vecinos.

Estado	2 WAY
Transición	Establecer Adyacencia
Nuevo estado	2WAY o EXTART
Descripción	Cuando una interfaz se encuentre en el estado *2 WAY* debe establecer adyacencia con otro enrutador. En redes Ethernet esta adyacencia se debe realizar con el DR y el BDR, mientras que en

	enlaces seriales la adyacencia se realiza con el único vecino que tiene. Si realiza la adyacencia pasará al estado *EXTART* de lo contrario su estado será *2WAY.*

Estado	Actualizando
Transición	Actualizar
Nuevo estado	Timer
Descripción	Cuando llega un paquete Hello se debe actualizar la tablas de vecinos e iniciar un timer de 40 segundos. Si este timer expira indica que se ha perdido contacto con el vecino y por lo tanto se debe anunciar a las demás interfaces de este suceso por medio de tramas LSU.

En el estado Extart se realiza la decisión entre los Routers adyacentes sobre quien va a realizar las funciones de Maestro y Esclavo. La figura B.10 ilustra este proceso.

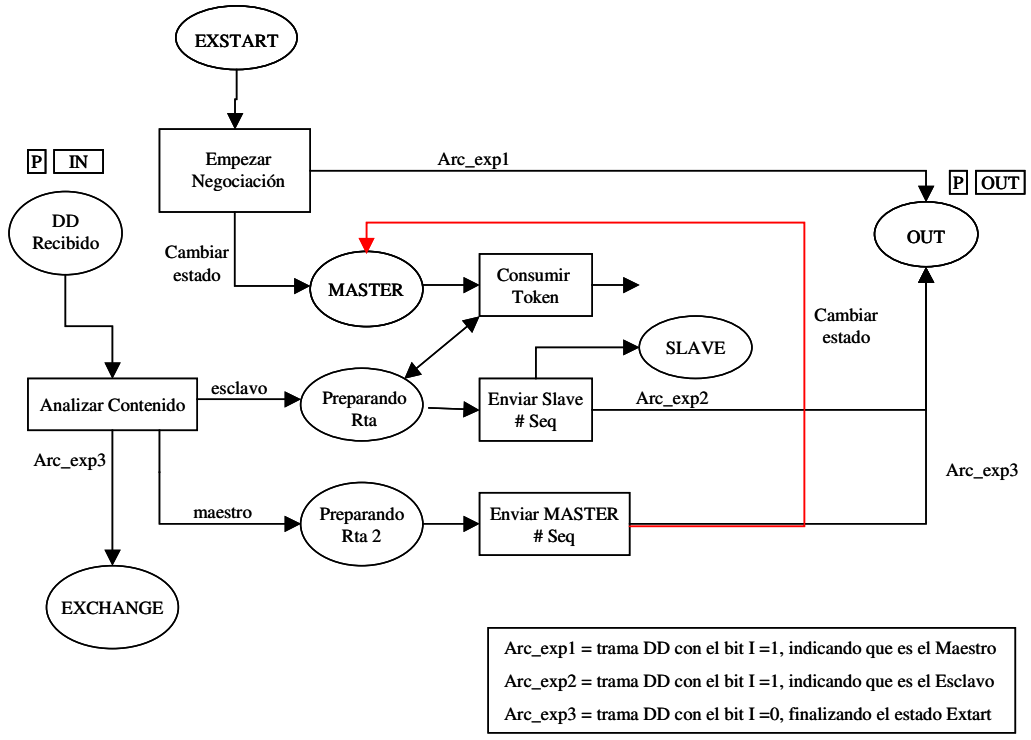

Figura B.10 Subpágina procesar DD primera parte

Estado	Extart
Transición	Empezar negociación

Nuevo estado	Master, OUT
Descripción	El enrutador que inicia el proceso extart se declara así mismo como maestro cambiando su estado a *Maestro* y envía el paquete a través de la red como una trama DD con el bit I en 1 (indicando negociación).

Estado	DD Recibido
Transición	Analizar Contenido
Nuevo estado	Preparando Rta 1 y 2, o EXCHANGE
Descripción	Cuando llega un paquete DD se analiza el contenido. En primer lugar si el bit I es igual a 0, indica que la negociación ya se realizó, y pasara al estado *EXCHANGE*. En caso contrario deberá analizar que rol tomará dentro del intercambio y enviará los paquetes de acuerdo a los resultados obtenidos.

Estado	Preparando Rta
Transición	Enviar Slave
Nuevo estado	Slave y Out
Descripción	Si llega un paquete DD indicando que es el maestro, y el receptor esta de acuerdo en ser el esclavo, colocara un token en el estado *Slave*, y enviará una trama DD indicando este hecho. Existe la posibilidad que un enrutador se haya declarado erróneamente como Maestro, pero cuado llegan los paquetes del enrutador adyacente debe cambiar su estado de maestro a esclavo, por tal motivo es necesario consumir el token del estado *Maestro*.

Estado	Preparando Rta2
Transición	Enviar Master
Nuevo estado	Master y Out
Descripción	Si llega un paquete DD, y el receptor según sus cálculos debe ser el maestro, colocará un token en el estado *Maestro*, y enviará una trama DD indicando este hecho.

En el estado Exchange se intercambian las bases de datos del estado del enlace. La figura B.11 explica este proceso.

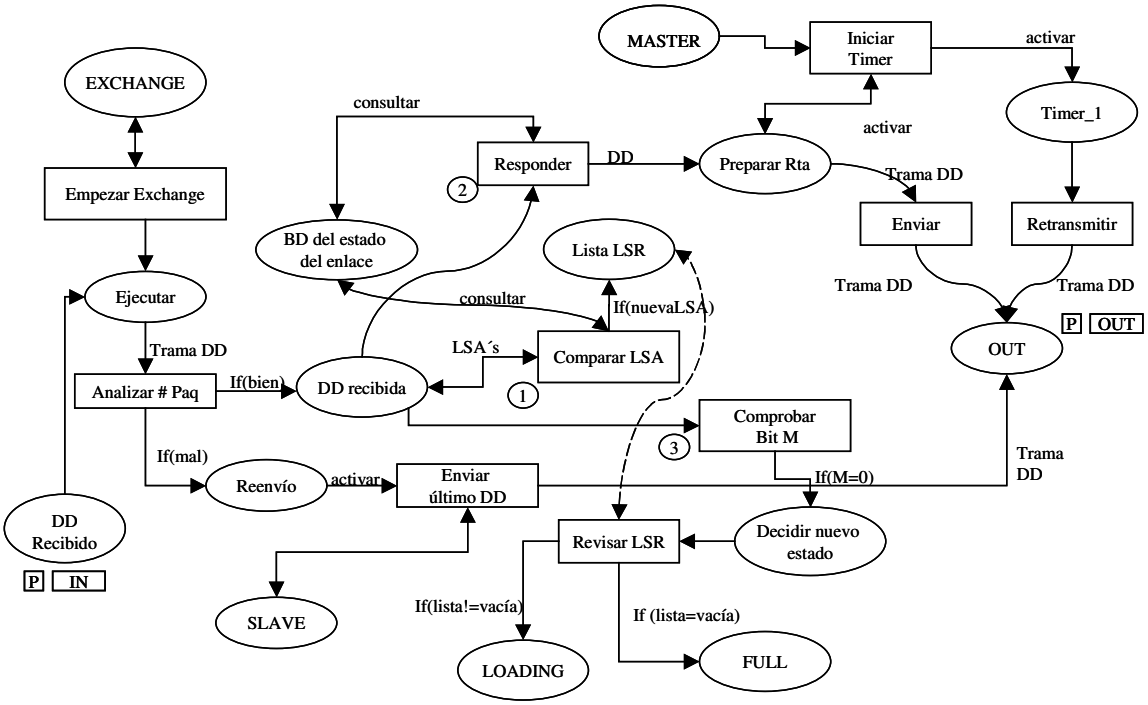

Figura B.11 Subpágina procesar DD segunda parte

Estado	Ejecutar
Transición	Analizar # Paq
Nuevo estado	DDRecibida o Reenvió
Descripción	Para llevar el control de flujo de los paquetes DD se revisa su número de secuencia. Si el análisis arroja un resultado positivo, es decir que el paquete que recibió era el que se esperaba, el token pasará al estado *DD Recibida*. En caso contrario, es decir si ha llegado una trama duplicada, se deberá enviar el último paquete que se recibió de manera adecuada pasando el token al estado *Reenvío* (esta acción es realizada por el esclavo)

Estado	DD Recibida
Transición	Comparar LSA, Responder y Comprobar bit M
Nuevo estado	Depende de la acción que se ejecute
Descripción	▪ Cuando llega el paquete DD se deben revisar las LSAs entrantes con las LSA que se encuentran en la Base de Datos del estado del enlace del enrutador, esta operación es realizada por la transición *CompararLSA*. Si existe alguna LSA que no contiene la coloca en el buffer *Lista LSR*.

	▪ Posteriormente, el enrutador debe responder con una paquete DD con sus propios datos de la Base de Datos del Estado del Enlace. Esta acción es realizada por la transición *Responder* que se encarga de armar el trama para ser enviada y pasa el token al estado *Preparar Rta.* ▪ Finalmente, se debe revisar si el bit M del paquete DD es igual a 0 (indica el último paquete DD), si es así pasa el token al estado *Decidir Nuevo Estado*.

Estado	Preparar Rta
Transición	Enviar
Nuevo estado	Out
Descripción	Envía las tramas DD con las LSA de su Base de datos del Estado del Enlace. Si el enrutador que esta enviando el paquete es el Maestro debe iniciar un temporizador para enviar la misma trama en caso de que no escuche una respuesta del esclavo.
Estado	Decidir Nuevo estado
Transición	Revisar LSR
Nuevo estado	Loading o Full
Descripción	Cuando se han enviado todos los paquetes DD, se debe decidir cual será el próximo estado. Para ello se debe revisar la lista LSR. Si se encuentra vacía pasa al estado *Full* y el enrutador puede en ese momento empezar a realizar sus procesos de enrutamiento. Si no se encuentra vacía, debe pedir mayor información de las nuevas LSAs al enrutador con el cual realizó la adyacencia.

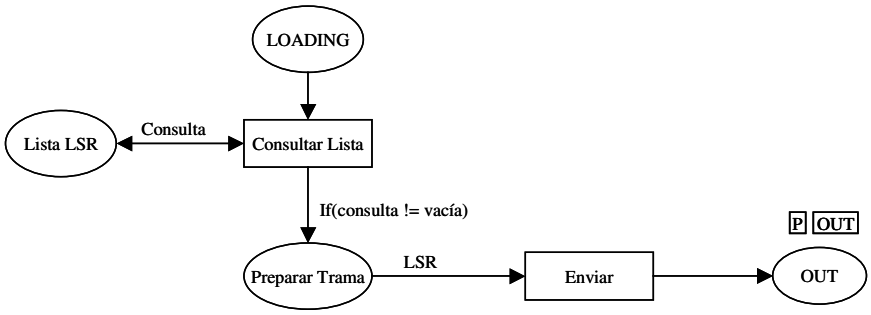

Figura B.12 Subpágina procesar DD tercera parte

Estado	Loading
Transición	Consultar Lista
Nuevo estado	Preparar Trama

Descripción	En el estado Loading se debe pedir mayor información de las nuevas LSAs al enrutador con el cual realizó la adyacencia, para ello se consulta la lista LSR y se pasa al estado *Preparar Trama.*

Estado	Preparar Trama
Transición	Enviar
Nuevo estado	Out
Descripción	Prepara una trama LSR con las nuevas LSA. Estas tramas son usadas para pedir mayor información sobre las LSAs.

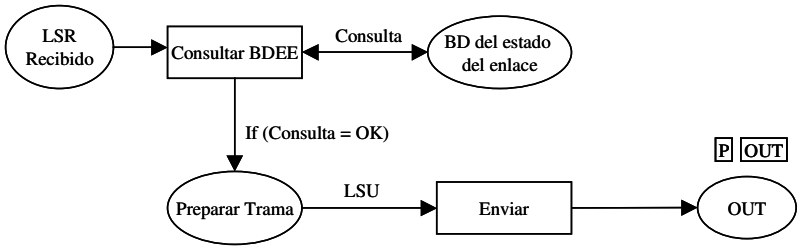

Figura B.13 Subpágina procesar LSR

Estado	LSR recibido
Transición	Consultar BDEE
Nuevo estado	Preparar Trama
Descripción	Cuando llega una trama LSR se debe consultar en la Base de datos del Estado del Enlace para obtener mayor información de las LSAs. Con la nueva información es posible prepara la nueva trama que contendrá toda esta información.

Estado	Preparar Trama
Transición	Enviar
Nuevo estado	Out
Descripción	Prepara una trama LSU que contendrá la información par actualizar la base de datos del estado del enlace.

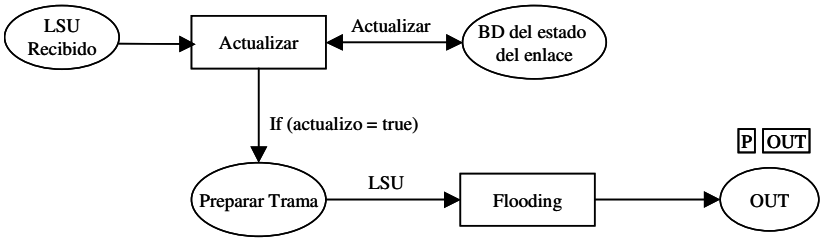

Figura B.14 Subpágina procesar LSU

Estado	LSU recibido
Transición	Actualizar
Nuevo estado	Preparar Trama
Descripción	Cuando llega una trama LSU se actualiza la Base de datos del Estado del Enlace. OSPF especifica que cuando llegue una actualización es necesario realizar un proceso de inundación de esta información por las demás interfaces del enrutador. El estado *Preparar Trama* y la transición *Flooding* realizarán este proceso,

1.3 Modelado del protocolo BGP_4 con redes de Petri

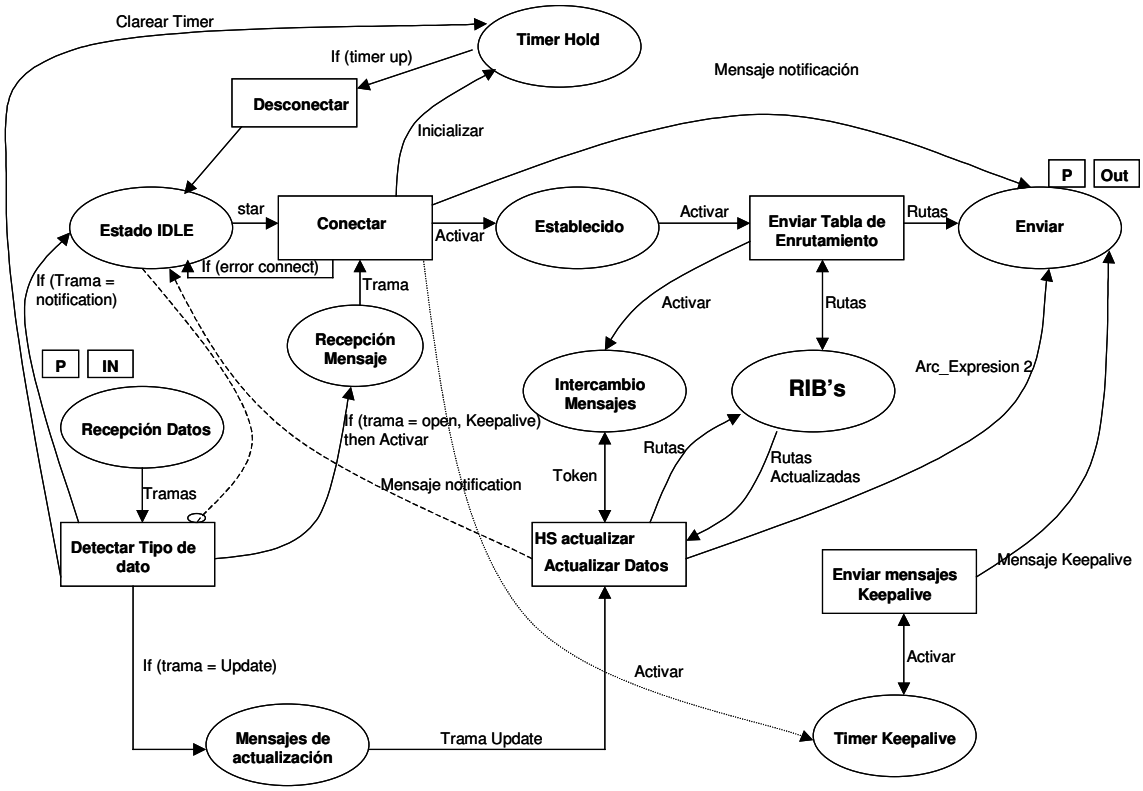

Figura B.15 Protocolo BGP

La figura representa el esquema CPN para el modelado del protocolo BGP_4, el cual consta de 10 estados y 5 transiciones, una de las cuales; Actualizar datos es una transición sustituta, que se especifica en la subpágina actualizar.

Estado	Estado Idle
Transición	Conectar
Nuevo estado	Timer Hold, establecido, enviar, Timer KeepAlive
Descripción	Una vez el sistema es inicializado por el operador, se activa la transición *Conectar*, la cual se encarga de establecer una conexión TCP con un vecino BGP; si este proceso es llevado a cabo de manera exitosa la transición inicializa el timer hold presente en el estado *Timer Hold* y el timer keepalive presente en el estado *Timer KeepAlive* (que se encarga de enviar mensajes keepalive de forma periódica) y pasa al estado *Establecido*, en caso contrario el sistema envía un mensaje de notificación a su vecino y regresa al estado *Idle*.

Estado	Establecido
Transición	Enviar tabla de enrutamiento
Nuevo estado	Intercambio de mensajes
Descripción	Cuando se ha establecido una conexión exitosa con un vecino y el sistema se encuentra en el estado *Establecido*, se habilita la transición *Enviar Tabla de Enrutamiento*, la cual se encarga de transmitir al vecino las rutas que tiene en las RIB's. Una vez la transición realiza su tarea, el sistema pasa al estado *Intercambio de mensajes*, donde se espera recibir mensajes de actualización, keepalive y notificación,

Estado	Recepción de datos
Transición	Detectar tipo de datos
Nuevo estado	Mensajes de actualización
Descripción	En el estado *Recepción de datos* se recogen todos los datos provenientes de los vecinos BGP, es así como cuando llega un dato se habilita la transición *Detectar tipo de datos*, la cual se encarga de mirar el tipo de mensaje que llegó, inicializar el timer hold y llevar al sistema al estado *Mensajes de Actualización* (update), donde se activa la transición *Actualizar datos* que se especifica mas adelante.

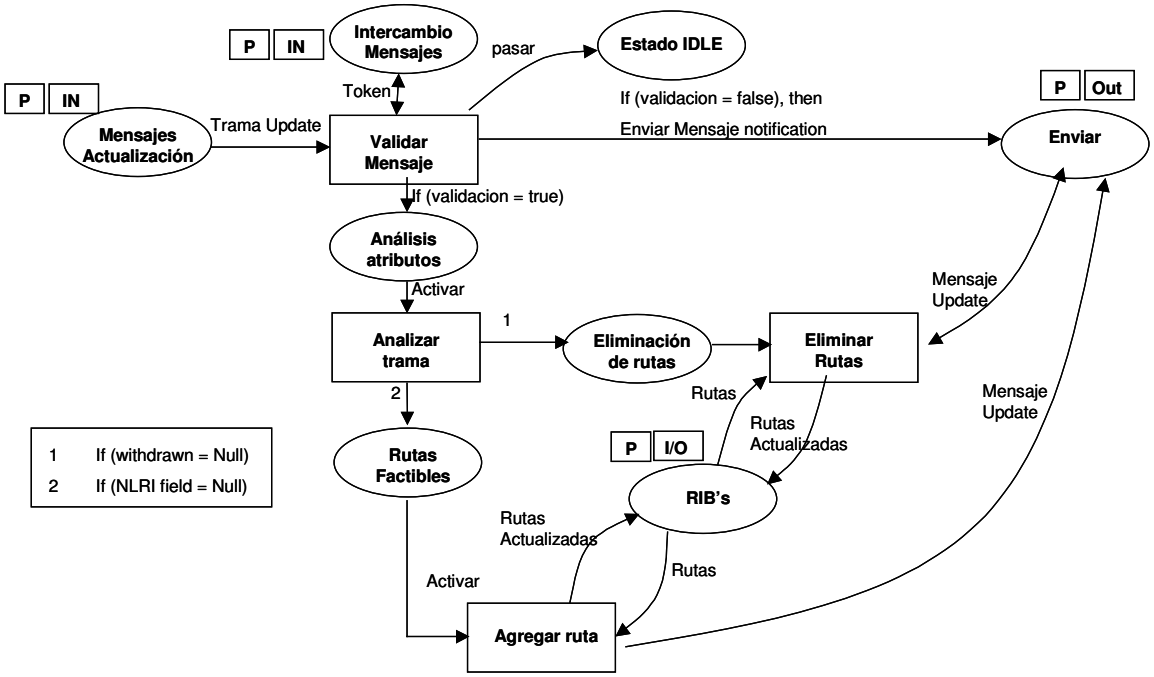

Figura B.16 Subpágina actualizar

Estado	Mensajes de actualización
Transición	Validar mensaje
Nuevo estado	Analizar atributos
Descripción	Una vez llega un mensaje de actualización se activa la transición *Validar Mensaje*, que se encarga de revisar que el mensaje provenga de un vecino conocido y que los datos contenidos en el tengan valores correctos, si no se encuentran errores el sistema pasa al estado *Analizar Trama*, en caso contrario el sistema pasa al estado *Idle* y envía un mensaje de Notificación al vecino para cerrar la conexión TCP.

Estado	Análisis atributos
Transición	Analizar trama
Nuevo estado	Eliminación de rutas o rutas factibles
Descripción	En el estado *Análisis atributos* se guarda la trama de actualización, la cual ha sido previamente validada, es aquí donde se habilita la transición *Analizar trama* que se encarga de ver si el paquete trae rutas que deban ser eliminadas o si por el contrario trae una ruta para ser agregada a las RIB's. En el primer caso el sistema pasa al estado *Eliminación de rutas*, en el segundo caso el sistema pasa al estado *Rutas factibles*.

Estado	Eliminación de rutas
Transición	Eliminar rutas
Nuevo estado	Enviar
Descripción	El estado *Eliminación de rutas* activa la transición *Eliminar rutas* que se encarga de borrar de las RIB's las rutas que han sido anunciadas en el paquete de actualización para ser sacadas del servicio y manda un mensaje de actualización con las rutas eliminadas a los demás vecinos por medio del estado *Enviar,* o con rutas de reemplazo si estas existen.

Estado	Rutas factibles
Transición	Agregar ruta
Nuevo estado	Enviar
Descripción	El estado *Rutas factibles* activa la transicion *Agregar ruta* que se encarga de agregar a las RIB's las rutas que han sido aprendidas y envía un mensaje de actualización a los demás vecinos por medio del estado *Enviar*.

1.4 Modelado del protocolo IGRP con redes de Petri

Debido a la similitud que presenta el protocolo IGRP con el protocolo RIP se utilizó el modelo de este último protocolo para implementar IGRP.

2 Modelado de la herramienta con RUP

2.1 Construcción del árbol de funciones

A continuación se identifican las funciones del sistema y se establece una jerarquía donde las funciones más generales agrupan las mas específicas, la siguiente tabla muestra el árbol de funciones para el sistema:

1. Dibujar Figura (hace referencia a los dispositivos de red que se podrán manipular con la herramienta, estos dispositivos son: enrutadores, hubs y hosts).
    1.1 Seleccionar figura.
    1.2 Copiar figura.
    1.3 Pegar figura.
    1.4 Eliminar figura

1.5 Conectar figura.
1.6 Mover figura.

2. Manipular Archivos (hacen referencia a los archivos generados por la herramienta a partir de las redes construidas por el usuario).
   2.1 Guardar archivo.
   2.2 Abrir archivo.
   2.3 Modificar archivo.
   2.4 Imprimir archivo.
   2.5 Crear archivo.

3. Hacer consultas
   3.1 Consultar manual de usuario.
   3.2 Consultar teoría de los protocolos de enrutamiento.
   3.3 Consultar laboratorios.
   3.4 Consultar datos de diseñadores.

4. Ejecutar simulación
   4.1 Seleccionar protocolo a simular.
   4.2 Configurar interfaces del enrutador.
   4.3 Empezar simulación.
   4.4 Mostrar tramas.
   4.5 Detener simulación.
   4.6 Desactivar interfaz del enrutador
   4.7 Activar interfaz del enrutador.
   4.8 Mirar tabla de enrutamiento.
   4.9 Monitorear figura enrutador.

Tabla B.1 Funciones del Sistema

2.2 Diagrama de casos de uso

El sistema cuenta con un solo autor que se relaciona de manera directa con el sistema, y de ahora en adelante lo llamaremos usuario.
Los casos de usos se desprenden del árbol de funciones, y se presentan en la figura B.17

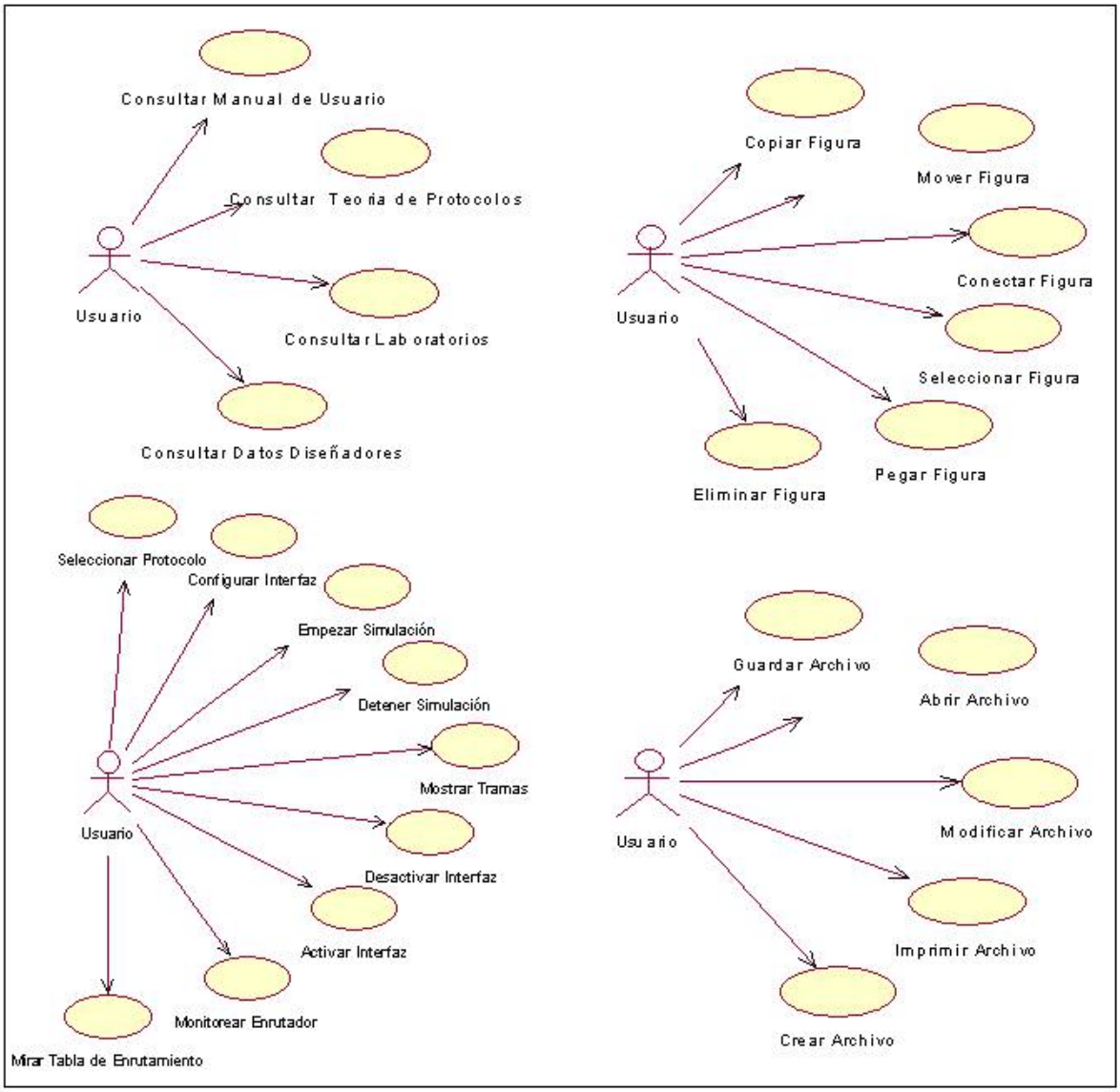

Figura B.17 Diagrama de casos de uso

Descripción de los casos de uso

Caso de uso	Dibujar figuras
Actor	Usuario
Tipo	Primario
Descripción	El usuario selecciona la figura (enrutador, hub o host) que desea colocar en el área de dibujo. El sistema guarda la información de su elección. El usuario da clic sobre el área de dibujo. El sistema dibuja la figura.

Caso de uso	Copiar
Actor	Usuario
Tipo	Secundario
Descripción	El usuario selecciona dentro del área de dibujo la figura que desea copiar. El usuario da la orden al sistema de copiar el dispositivo. El sistema guarda la figura que el usuario copio.

Caso de uso	Pegar
Actor	Usuario
Tipo	Secundario
Descripción	El usuario solicita al sistema pegar una figura. El sistema dibuja la figura que se había copiado anteriormente en el área de dibujo.

Caso de uso	Eliminar
Actor	Usuario
Tipo	Secundario
Descripción	El usuario selecciona una figura en el área de dibujo y solicita al sistema borrarlo. El sistema borrará el dispositivo del área de dibujo y si tiene conexiones con otros dispositivos también se quitarán dichas conexiones.

Caso de uso	Conectar figuras
Actor	Usuario
Tipo	Primario
Descripción	El usuario da clic con el botón derecho del mouse sobre una de las figuras que coloco en el área de dibujo. El sistema mostrará las interfaces del dispositivo. El usuario selecciona cualquier interfaz. Esta operación se repite en la figura con la cual se quiere realizar la conexión. El sistema dibujará una línea indicando que se ha realizado la conexión.

Caso de uso	Seleccionar
Actor	Usuario

Tipo	Primario
Descripción	El usuario da clic sobre una figura que se encuentra en el área de dibujo. El sistema dibuja unos pequeños cuadros negros alrededor del dispositivo que selecciono.

Caso de uso	Mover
Actor	Usuario
Tipo	Primario
Descripción	El usuario da clic sobre una figura que se encuentra en el área de dibujo dejando el botón del mouse presionado. A medida que mueva el mouse, la figura se moverá con el. El sistema captura las posiciones del cursor y dibuja la figura en la nueva posición.

Caso de uso	Guardar
Actor	Usuario
Tipo	Primario
Descripción	El usuario solicita al sistema que guarde la configuración de red realizada. El sistema despliega un formulario para que se especifique el directorio y el nombre del archivo. El usuario debe escoger el directorio y el nombre del archivo. El sistema guardará la configuración de red en un archivo.

Caso de uso	Abrir
Actor	Usuario
Tipo	Primario
Descripción	El usuario da clic sobre botón abrir. El sistema despliega un formulario para que el usuario escoja el archivo que desea abrir. El sistema presentará la información guardada anteriormente en el archivo.

Caso de uso	Imprimir
Actor	Usuario
Tipo	Secundario
Descripción	El usuario solicita al sistema imprimir. El sistema imprimirá la topología de red que se encuentra en el área

	de dibujo.

Caso de uso	Manual de usuario
Actor	Usuario
Tipo	Secundario
Descripción	El usuario solicita ayuda. El sistema despliega una página web que contiene la ayuda para el manejo de la herramienta.

Caso de uso	Consultar Protocolos
Actor	Usuario
Tipo	Secundario
Descripción	El usuario solicita información de los protocolos. El sistema desplegará una página web que contiene información teórica sobre los protocolos de enrutamiento.

Caso de uso	Consultar Laboratorios
Actor	Usuario
Tipo	Secundario
Descripción	El usuario solicita consultar laboratorios. El sistema desplegará una página web que contiene los laboratorios que se diseñaron para el manejo de la herramienta.

Caso de uso:	Consultar datos de los diseñadores
Actores:	Usuario
Tipo:	Secundario
Descripción:	El usuario solicita al sistema ver la información de los diseñadores e implementadores de la herramienta. El sistema despliega en pantalla la información solicitada. El usuario recibe la información.

Caso de uso:	Seleccionar protocolo
Actores:	Usuario
Tipo:	Primario
Descripción:	El usuario escoge el protocolo con el que desea hacer la simulación. El sistema deshabilita la selección de los otros protocolos y muestra al usuario de manera permanente y hasta que dure la simulación el tipo de tramas que implementa el protocolo seleccionado y los campos de la tabla de enrutamiento con la que trabaja dicho

	protocolo.

Caso de uso:	Configurar interfaces
Actores:	Usuario
Tipo:	Primario
Descripción:	El usuario selecciona un enrutador y solicita al sistema configurar las interfaces de dicho componente. El sistema verifica que la operación puede efectuarse y despliega las interfaces del componente y los campos necesarios para la correcta configuración de cada una de estas. El usuario ingresa la información (dirección IP, mascara, etc) para la configuración. El sistema modifica el estado y los parámetros de la interfaz.

Caso de uso:	Empezar simulación
Actores:	Usuario
Tipo:	Primario
Descripción:	El usuario solicita al sistema iniciar la simulación de uno de los protocolos de enrutamiento que este soporta. El sistema presenta al usuario de manera gráfica el intercambio de paquetes entre los enrutadores y los cambios en la tablas de enrutamiento de estos.

Caso de uso:	Mostrar tramas
Actores:	Usuario
Tipo:	Secundario
Descripción:	El usuario solicita al sistema información del contenido de las tramas que se presentan en la simulación. El sistema muestra al usuario la información solicitada.

Caso de uso:	Detener simulación
Actores:	Usuario
Tipo:	Primario
Descripción:	El usuario solicita al sistema detener un proceso de simulación que ya había sido iniciado. El sistema libera todos los recursos asignados a la simulación.

Caso de uso:	Desactivar interfaz de los componentes
Actores:	Usuario

Tipo:	Primario
Descripción:	El usuario selecciona un componente, escoge una interfaz y solicita al sistema desactivarla. El sistema desactiva la interfaz y cambia la apariencia del formulario.

Caso de uso:	Activar interfaz de los componentes
Actores:	Usuario
Tipo:	Primario
Descripción:	El usuario selecciona un componente, escoge una interfaz y solicita al sistema activarla. El sistema activa la interfaz y cambia la apariencia física del formulario.

Caso de uso:	Mirar tabla de enrutamiento
Actores:	Usuario
Tipo:	Primario
Descripción:	El usuario selecciona un enrutador y el sistema le responde con la tabla de enrutamiento del componente seleccionado.

Caso de uso:	Monitorear enrutador
Actores:	Usuario
Tipo:	Primario
Descripción:	El usuario le solicita al sistema monitorear el trafico enviado y recibido por un enrutador. El sistema presenta al usuario el tipo de paquetes que puede monitorear. El usuario selecciona el tipo de paquete a monitorear. El sistema presenta al usuario todos los paquetes entrantes y salientes del tipo seleccionado.

2.3 Casos De Uso Extendidos

Dibujar figuras

Caso de uso:	Dibujar figuras
Actores:	Usuario (iniciador)
Tipo:	Primario
Resumen:	El usuario dibuja una figura (enrutador, hub o host) en el área de dibujo.

Referencias cruzadas:	Funciones: 1.1

Precondiciones

No existe ninguna precondición para iniciar este caso de uso.

Flujo Principal

- Este caso de uso empieza cuando el usuario selecciona un dispositivo del panel componentes.
- El sistema guarda la selección del usuario.
- El usuario deberá pulsar clic en el área de dibujo.
- El sistema dibuja en el área de dibujo el dispositivo seleccionado con su respectivo nombre.

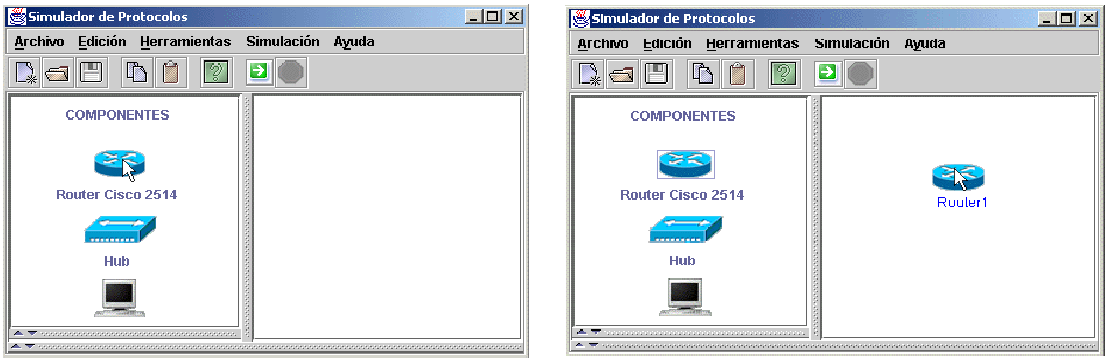

Figura B.18 Formulario principal del sistema

Conectar figuras

Caso de uso:	Conectar figuras
Actores:	Usuario (iniciador)
Tipo:	Primario
Resumen:	Se realiza la conexión entre dos dispositivos en el área de dibujo
Referencias cruzadas:	Funciones:1.6 Caso de uso: Dibujar Figura

Precondición

- Se debe haber dibujado por lo menos dos figuras en el área de dibujo

Flujo Principal

- El usuario pulsa clic derecho sobre una de las figuras.
- El sistema presenta las interfaces (puertos) del dispositivo.

- El usuario selecciona una interfaz libre.
- El sistema guarda la elección de la interfaz y sabe que va a realizar una conexión, por tal razón a medida que el usuario mueva el mouse dentro del área de dibujo una línea se dibujará indicando que esta realizando una conexión.
- El usuario selecciona otra figura con la cual realizará la conexión siguiendo el procedimiento descrito anteriormente (E1).
- El sistema realiza la conexión entre las dos figuras (E2). En caso de ser así se dibujará una línea entre los dos dispositivos, representando la conexión, y modificará los interfaces (puertos) de los dispositivos.

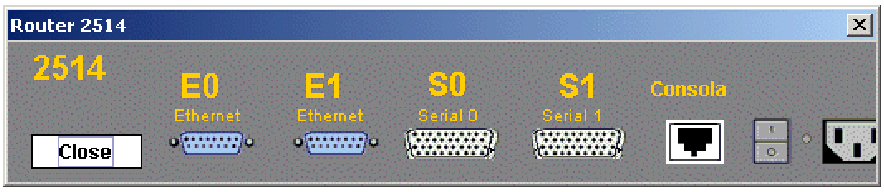

Figura B.19 Formulario interfaces de los dispositivos

Flujos de excepción

E1: El sistema despliega un mensaje informando que no se ha seleccionado ningún dispositivo.

- El caso de uso termina sin realizar la conexión.

E2: El sistema despliega un mensaje indicando que no es posible realizar la conexión.

- El caso de uso termina sin realizar la conexión. Hay que tener en cuenta que no todas las conexiones se pueden realizar. Existen algunas restricciones:

1. Solo se pueden conectar interfaces Ethernet con interfaces Ethernet, y seriales con seriales
2. No se pueden conectar dos interfaces de un dispositivo con dos interfaces de otro dispositivo
3. No se puede conectar una interfaz de un dispositivo con otra interfaz del mismo dispositivo
4. No se pueden conectar 2 hubs (Esta restricción se tomo debido a problemas en el tiempo de transporte de los paquetes a través de la red)

Seleccionar figura

Caso de uso:	Seleccionar figura
Actores:	Usuario (iniciador)
Tipo:	Primario
Resumen:	Se selecciona un dispositivo dentro del área de dibujo
Referencias	Funciones: 1.2, 1.3, 1.4, 1.5, 1.6, 1.7

cruzadas:	Caso de uso: dibujar figura

Precondición

- Se debe haber dibujado por lo menos un dispositivo en el área de dibujo

Flujo Principal

- El usuario da clic sobre uno de los dispositivos del área de dibujo.
- El sistema guarda la posición donde se dio el clic y la compara con la posición de los dispositivos dentro del área de dibujo.
- Si se encuentran en la misma posición, el sistema dibuja unos pequeños cuadros negros alrededor del dispositivo que selecciono.

Copiar figura

Caso de uso:	Copiar
Actores:	Usuario (iniciador)
Tipo:	Secundario
Resumen:	Copia un dispositivo del área de dibujo
Referencias cruzadas:	Funciones: 1.3 Caso de uso: Dibujar figura, Seleccionar figura

Precondición

- Se debe haber dibujado por lo menos un dispositivo en el área de dibujo.
- Se debe haber seleccionado un dispositivo (caso de uso seleccionar figura).

Flujo Principal

- Este caso de uso empieza cuando el usuario pulsa clic sobre el botón Copiar de la barra de herramientas de la interfaz.
- El sistema guarda el tipo de dispositivo.

Flujos de excepción

E1 : El sistema despliega un mensaje indicando que debe seleccionar una figura.

- Este caso de uso no empieza si no se ha seleccionado una figura previamente.

Pegar figura

Caso de uso:	Pegar figura
Actores:	Usuario (iniciador)
Tipo:	Secundario
Resumen:	Dibuja una figura en el área de dibujo
Referencias cruzadas:	Funciones: 1.4 Caso de uso: Copiar figura

Precondición

- Se debe haber copiado una figura previamente.

Flujo Principal

- Este caso de uso empieza cuando el usuario pulsa clic sobre el botón Pegar de la barra de herramientas.
- El sistema dibuja en el extremo superior izquierdo del área de dibujo una figura igual al que se había copiado, pero con diferente nombre.

Eliminar figura

Caso de uso:	Eliminar figura
Actores:	Usuario (iniciador)
Tipo:	Secundario
Resumen:	Elimina una figura del área de dibujo
Referencias cruzadas:	Funciones: 1.5 Caso de uso: Dibujar figura, Seleccionar figura

Precondición

- Se debe haber dibujado por lo menos una figura en el área de dibujo.
- Se debe haber seleccionado una figura (caso de uso seleccionar figura).

Flujo Principal

- Este caso de uso empieza cuando el usuario pulsa clic sobre el botón Borrar del menú Edición.
- El sistema borra el dispositivo del área de dibujo (E1) y si esta conectado a otros dispositivos se modifican las imágenes de sus interfaces (puertos).

Flujos de excepción

E1 : El sistema despliega un mensaje indicando que no se puede eliminar un componente.

- Existen algunas restricciones para eliminar un dispositivo. Si se esta simulando un protocolo y el dispositivo que se quiere borrar esta corriendo, el sistema evita que se elimine.

Guardar archivo

Caso de uso:	Guardar archivo
Actores:	Usuario (iniciador)
Tipo:	Primario
Resumen:	Guarda en un archivo la topología de red creada.
Referencias	Funciones: 2.1

cruzadas:	

Precondición

- No se debe haber ejecutado el caso de uso empezar simulación.

Flujo Principal

- Este caso de uso empieza cuando el usuario pulsa clic sobre el botón Guardar de la barra de herramientas.
- El sistema despliega un formulario para que especifique la dirección y el nombre del archivo.
- El usuario debe escoger la posición, el nombre del archivo y selecciona una de las opciones del final del formulario Guardar y Cancelar.
- Si elige la opción guardar el sistema guardará la configuración de red en un archivo.
- Si elige la opción cancelar no se hará nada y el caso de uso terminará.

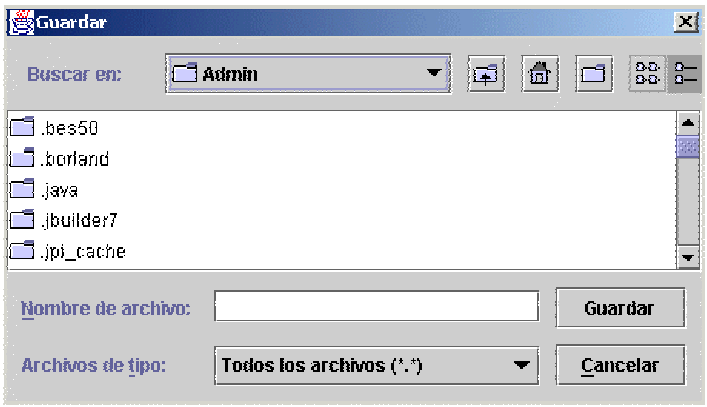

Figura B.20 Formulario guardar archivo

Abrir archivo

Caso de uso:	Abrir archivo
Actores:	Usuario (iniciador)
Tipo:	Primario
Resumen:	Abre un archivo.
Referencias cruzadas:	Funciones: 2.2

Precondición

- No existe ninguna precondición para este caso de uso.

Flujo Principal

- Este caso de uso empieza cuando el usuario pulsa clic sobre el botón Abrir de la barra de herramientas.
- El sistema presentará un cuadro de dialogo para que el usuario seleccione el archivo a abrir.
- El usuario escoge el archivo que desea abrir.
- El sistema presentará la información guardada anteriormente

Consultar Manual de Usuario

Caso de uso:	Consultar manual de usuario
Actores:	Usuario (iniciador)
Tipo:	Secundario
Resumen:	Muestra el manual de usuario
Referencias cruzadas:	Funciones: 3.1

Precondición

- No existe ninguna precondición para este caso de uso.

Flujo Principal

- Este caso de uso empieza cuando el usuario pulsa clic sobre el botón Ayuda de la barra de herramientas (E1).
- El sistema desplegará un archivo que contendrá el manual de usuario

Flujo de excepciones

E1 : El sistema despliega un mensaje indicando que no se encuentra el archivo.

- No se muestra el manual de usuario y el caso de uso termina.

Seleccionar protocolo

Caso de uso:	Seleccionar protocolo
Actores:	Usuario (iniciador)
Tipo:	Primario
Resumen:	El usuario escoge el protocolo con el que desea hacer la simulación.
Referencias cruzadas:	Funciones: 4.1, 4.2, 4.3, 4.4

Precondiciones

- No existe ninguna precondición para iniciar este caso de uso, el usuario lo puede hacer incluso antes de haber construido la red sobre la cual desea simular el protocolo.

Flujo principal

- Este caso de uso empieza cuando el usuario elige en el menú principal del sistema las siguientes opciones: simulación, protocolos y pulsa clic sobre uno de los protocolos que soporta la herramienta.
- Una vez el usuario ha elegido el protocolo que desea simular, el sistema deshabilitara la opción protocolos y mostrará las tramas y los campos de la tabla de enrutamiento que maneja el protocolo seleccionado.

Configurar interfaces del enrutador

Caso de uso:	Configurar interfaces del enrutador
Actores:	Usuario (iniciador)
Tipo:	Primario
Resumen:	El usuario selecciona un enrutador y solicita al sistema configurar las interfaces de dicho componente. El sistema verifica que la operación puede efectuarse y despliega a través de una GUI las interfaces del componente y los campos necesarios para la correcta configuración de cada una de estas. El usuario ingresa la información (dirección IP, mascara, etc) para la configuración. El sistema modifica el estado y los parámetros de la interfaz.
Referencias cruzadas	Funciones: 4.2, 4.3 Casos de uso: Seleccionar protocolo, Conectar figuras.

Precondiciones

- El usuario debe haber ejecutado el caso de uso seleccionar protocolo.
- El usuario debe haber ejecutado el caso de uso conectar figuras.
- La interfaz que desea configurar debe estar conectada a otra interfaz del mismo tipo (Ethernet o serial).

Flujo principal

- Este caso de uso empieza cuando el usuario pulsa doble clic sobre uno de los enrutadores.
- El sistema presenta al usuario el formulario de configuración del enrutador de la figura 15.
- El usuario escoge la opción configurar interfaces.

- El usuario selecciona la interfaz que desea configurar e introduce su dirección IP, la mascara de red, la métrica y opcionalmente el numero de área o sistema autónomo (dependiendo del protocolo que desea simular).

Figura B.21 Formulario configuración enrutador

- Una vez introducidos los datos el usuario escoge la opción aceptar (E1, E2, E3, E4). Los valores de loas campos del anterior formulario pueden ser modificados siempre y cuando el usuario no halla ejecutado el caso de uso empezar simulación, para realizar algún cambio simplemente se modifica el campo y se escoge la opción aceptar nuevamente.
- El sistema muestra un mensaje al usuario anunciándole que ha configurado correctamente la interfaz.

Flujos de Excepción

E1: El sistema despliega un mensaje informando que se deben llenar todos los campos habilitados.

- El sistema regresa al formulario de configuración del enrutador.

E2: El sistema despliega un mensaje informando que la dirección IP o la mascara no tienen un formato adecuado.

- El sistema regresa al formulario de configuración del enrutador.

E3: El sistema despliega un mensaje informando que la dirección IP ya ha sido asignada.

- El sistema regresa al formulario de configuración del enrutador.

E4: El sistema despliega un mensaje informando que no se ha podido configurar la interfaz que revise las conexiones.

- El sistema regresa al formulario de configuración del enrutador.

Empezar simulación

Caso de uso:	Empezar simulación
Actores:	Usuario (iniciador)
Tipo:	Primario

Resumen:	El usuario solicita al sistema iniciar la simulación de uno de los protocolos de enrutamiento que este soporta. El sistema presenta al usuario de manera gráfica el intercambio de paquetes entre los enrutadores y los cambios en la tablas de enrutamiento de estos.
Referencias cruzadas	Funciones: 4.3, 4.4 Casos de uso: Seleccionar protocolo

Precondiciones

- El usuario debe haber ejecutado el caso de uso seleccionar protocolo.

Flujo principal

- Este caso de uso empieza cuando el usuario elige en el menú principal del sistema la opción empezar (E1).
- El sistema ejecuta el algoritmo del protocolo seleccionado sobre la red implementada por el usuario, y muestra a este ultimo el intercambio de mensajes entre los enrutadores y si el usuario lo desea la tabla de enrutamiento del enrutador que el usuario seleccione.

Flujos de Excepción

E1: El sistema despliega un mensaje informando que el usuario debe escoger un protocolo.

- El sistema regresa al formulario que contiene el menú principal.

Activar interfaz del enrutador

Caso de uso:	Activar interfaz del enrutador
Actores:	Usuario (iniciador)
Tipo:	Primario
Resumen:	El usuario selecciona un componente, escoge una interfaz de dicho componente y solicita al sistema activarla.
Referencias cruzadas	Funciones: 4.7 Casos de uso: Empezar simulación, Configurar interfaz

Precondiciones

- El usuario debe haber ejecutado el caso de uso empezar simulación.
- El usuario debe haber insertado un nuevo componente en la red que esta simulando el protocolo.
- El usuario debe ejecutar el caso de uso configurar interfaces sobre el nuevo componente.

Flujo principal

- Este caso de uso empieza cuando el usuario introduce un nuevo componente a una red que ya se encuentra simulando un protocolo y ejecuta el caso de uso configurar interfaces sobre dicho componente.
- El usuario pulsa clic sobre el botón iniciar.
- El sistema inicializa la interfaz y corre sobre ella el protocolo que se esta simulando.

Monitorear figura enrutador

Caso de uso:	Monitorear figura enrutador
Actores:	Usuario (iniciador)
Tipo:	Primario
Resumen:	El usuario le solicita al sistema monitorear el trafico enviado y recibido por un enrutador. El sistema por medio de una GUI le presenta al usuario el tipo de paquetes que desea monitorear. El usuario selecciona el tipo de paquete a monitorear. El sistema presenta al usuario todos los paquetes entrantes y salientes del tipo seleccionado.
Referencias cruzadas	Funciones: 4.9 Casos de uso: Empezar simulación

Precondiciones

- El usuario debe haber ejecutado el caso de uso empezar simulación.

Flujo principal

- Este caso de uso empieza cuando el usuario selecciona un enrutador y pulsa doble clic sobre el.
- El sistema presenta el formulario de configuración del enrutador.
- El usuario escoge la opción monitorear enrutador y dentro de ella el tipo de mensaje que desea monitorear.
- El sistema despliega información sobre los mensajes entrantes y salientes del enrutador.

2.4 Diagramas de secuencia

Dibujar figura

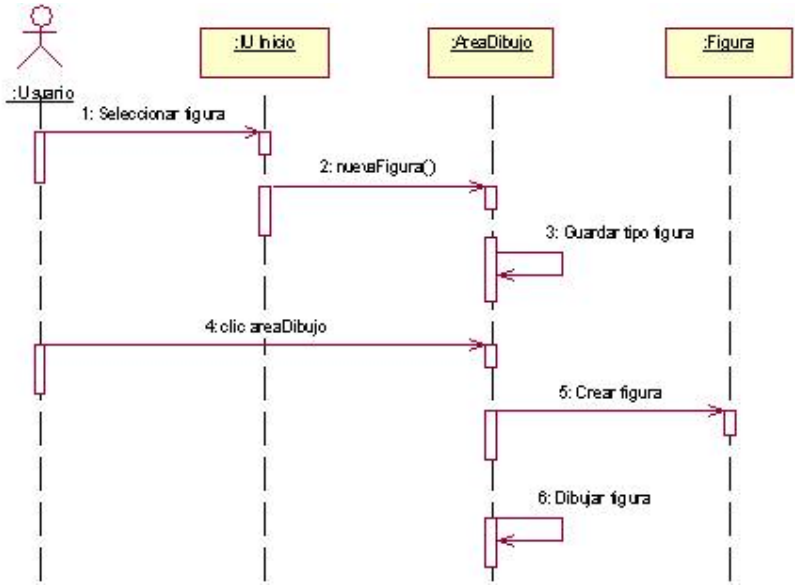

Figura B.22 Diagrama de secuencia dibujar componentes

La secuencia inicia cuando el usuario selecciona una figura del panel Componentes de la clase *IU_Inicio*. Esta invoca la función nuevaFigura(), ofrecida por la clase *AreaDibujo*, la cual guarda el tipo de figura que se va dibujar (enrutador,hub,host).
El usuario pulsa clic sobre el área de dibujo, la cual obtiene la posición donde se realizo el clic, crea una instancia de la clase figura y muestra en pantalla el nuevo componente.

Seleccionar figura

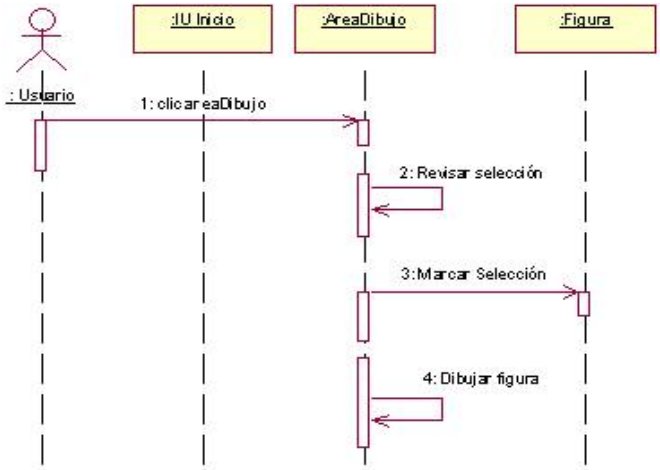

Figura B.23 Diagrama de secuencia seleccionar

El usuario pulsa clic sobre el área de dibujo. Este evento es recibido por la clase *AreaDibujo* a través de mouseClicked() que obtiene la posición donde se realizo el clic y lo compara con la posición de las figuras dentro del área de dibujo; si se encuentran en la misma posición el sistema marcará la figura como seleccionada y la guardará en una variable. Cuando ejecute dibujar figura aparecerán unos pequeños cuadros negros alrededor del dispositivo seleccionado.

Copiar figura (Caso exitoso)

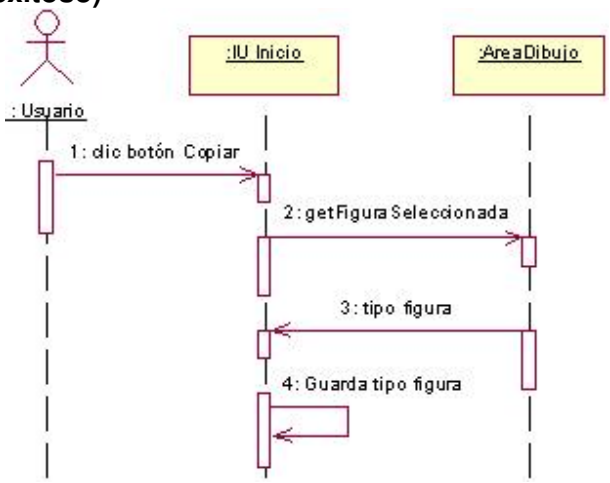

Figura B.24 Diagrama de secuencia Copiar figura

La secuencia se inicia cuando el usuario selecciona una figura (explicado anteriormente). El usuario presiona el botón copiar de la clase Inicio. Este evento invoca a la función getFiguraSeleccionada( ) de la clase *AreaDibujo* para solicitar el tipo de figura que se ha seleccionado para guardarla en una variable.

Conectar figuras

La secuencia se inicia cuando el usuario pulsa clic derecho sobre una figura en el área de dibujo. Este evento es atendido por un objeto de la clase *AreaDibujo* que muestra un formulario con las interfaces (puertos) del dispositivos.
El usuario pulsa sobre cualquiera de los puertos, el formulario desaparece y utiliza la función conectar() de la clase *AreaDibujo* para indicar el número de puerto y el tipo de figura.
Este mismo proceso se realiza en la figura con la cual se quiere conectar. Cuando se completa el proceso, la clase *AreaDibujo* revisa si se puede o no realizar la conexión. Si se pueden conectar, el área de dibujo debe actualizar las puertos de las figuras, dibujar una línea entre las dos dispositivos y crear un objeto de la clase *Conexión*.
Si no se puede realizar la conexión se mostrará un formulario indicando este hecho.

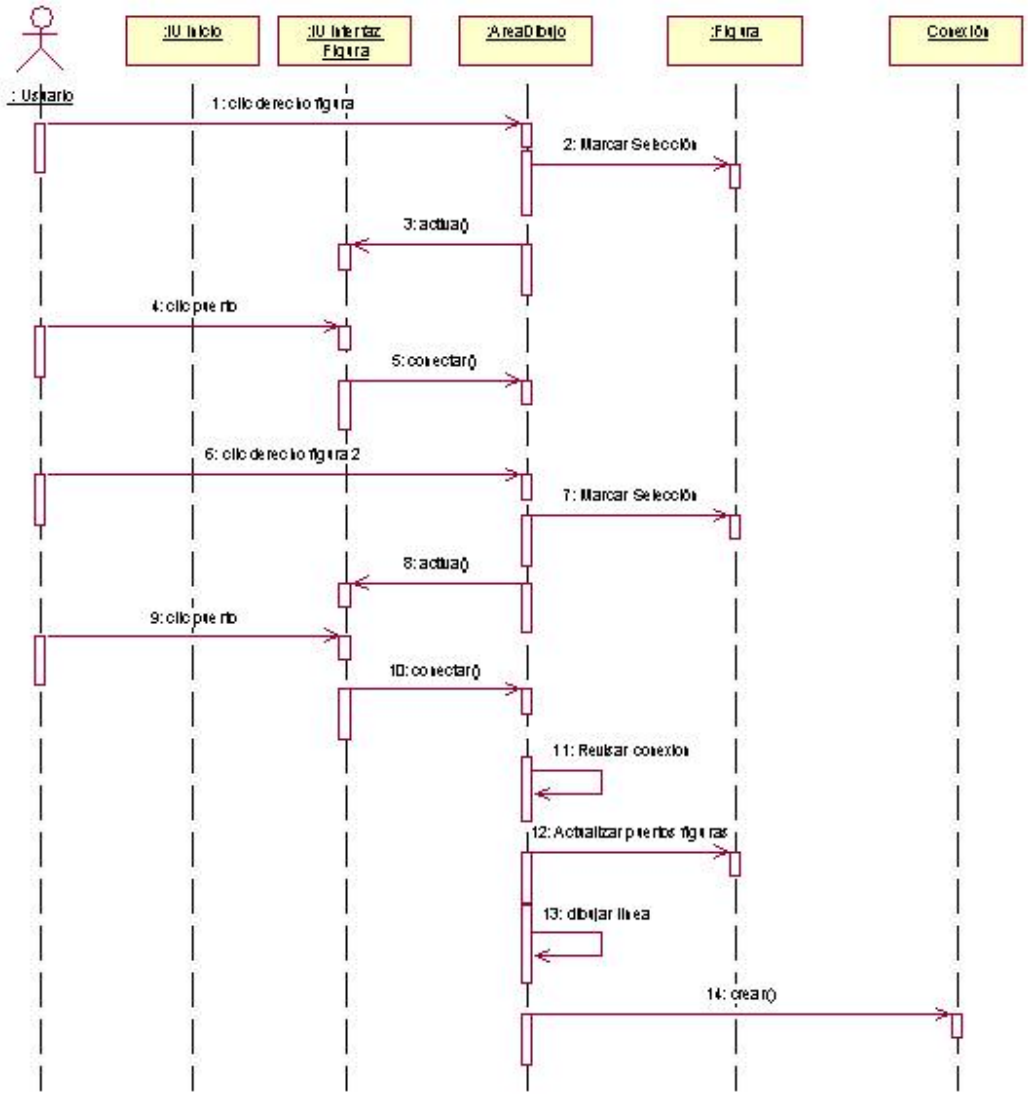

Figura B.25 Diagrama de secuencia Conectar figuras

Grabar archivo

La secuencia inicia cuando el usuario presiona el botón grabar del área de dibujo. Este evento hace que el sistema presente un formulario *IU_Grabar* el cual solicita al usuario la ubicación donde quiere grabar el archivo y el nombre del mismo. El usuario ingresa la información solicitada y presiona el botón guardar. Esta información es enviada a la clase *IU_Inicio* la cual llama a la función guardar de la clase *AreaDibujo*.

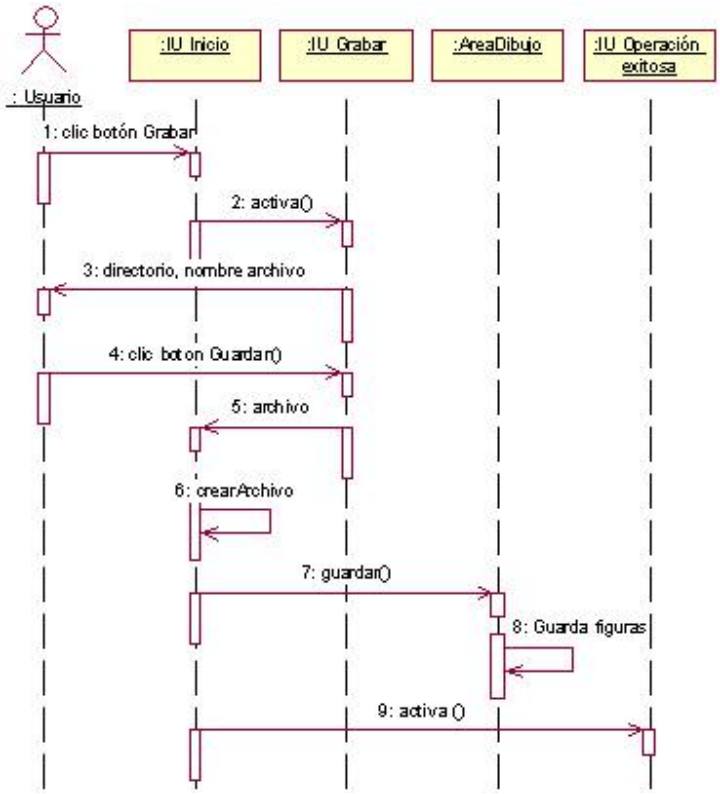

Figura B.26 Diagrama de secuencia Grabar archivo

Consultar Manual de usuario

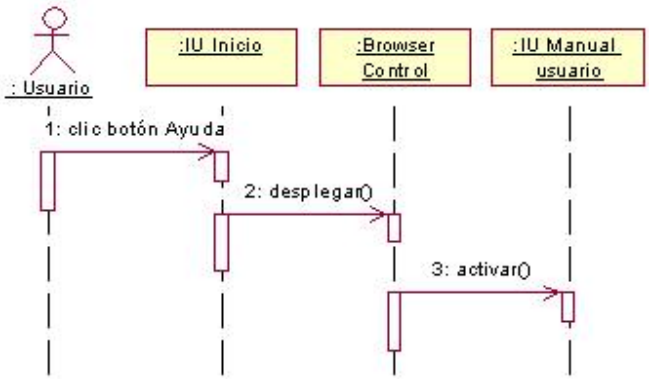

Figura B.27 Diagrama de secuencia Manual de usuario

El usuaria presiona el botón ayuda de *IU_Inicio*. Esta llama a la función displayURL() de *BrowserControl* la cual se encarga de lanzar un formulario con la información correspondiente a la ayuda.

Seleccionar protocolo

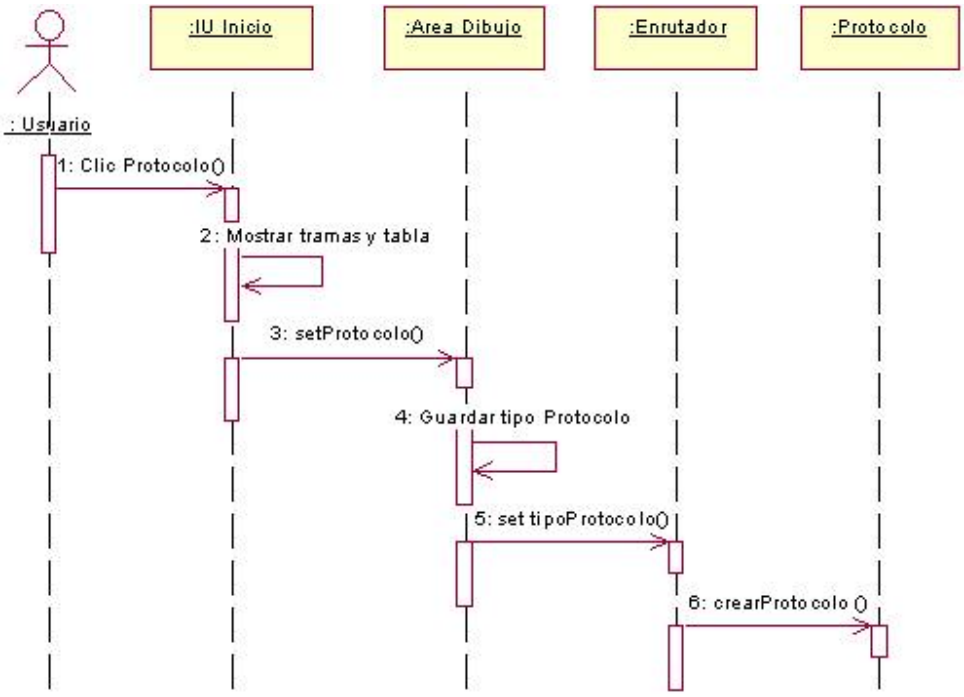

Figura B28. Diagrama de secuencia Seleccionar protocolo

La secuencia se inicia cuando el usuario, al ordenar la ejecución de la aplicación, activa un objeto de la clase *IU_Inicio*, que presenta la interfaz con las opciones que el usuario puede ejecutar. Seguidamente el usuario pulsa clic sobre el botón protocolo y selecciona el que desea simular. La información es recogida por *IU_Inicio* que muestra al usuario los mensajes que usa el protocolo y su tabla de enrutamiento. *IU_Inicio* invoca la operación setProtocolo() ofrecida por *AreaDibujo* que guarda el tipo de protocolo seleccionado y utiliza setTipoProtocolo() en *Enrutador* para crear un objeto de la clase *Protocolo*.

Configurar Interfaces

El usuario pulsa doble clic en el área de dibujo, evento que es recogido por la operación mouseClicked() de *AreaDibujo*, la cual analiza si este se dio sobre un objeto de la clase *Enrutador*, de ser así y de cumplirse con los requerimientos del caso de uso, se activa un objeto de la clase *DialogoConfiguración,* donde el usuario selecciona la interfaz que desea configurar, introduce los datos y presiona clic sobre el botón Aceptar, evento que invoca la operación configurarInterfaz(), que se encarga de analizar los parámetros introducidos y de guardar los datos. El sistema retorna una respuesta exitosa a *DialogoConfiguracion* quien finalmente activa un objeto de la clase *IU_OperacionExitosa.*

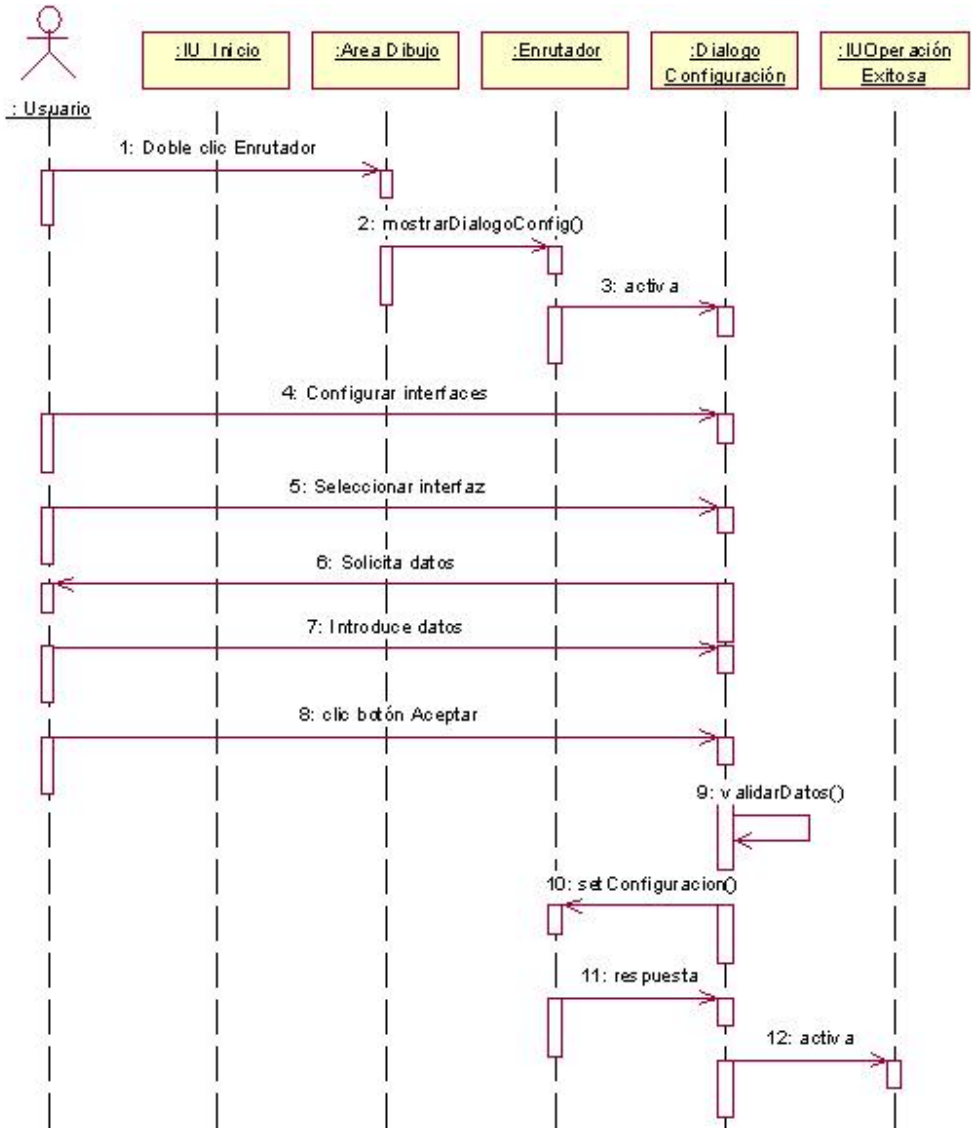

Figura B.29 Diagrama de secuencia Configurar interfaces

Monitorear figura Enrutador

El usuario pulsa doble clic en el área de dibujo, evento que es recogido por la operación mouseClicked() de *AreaDibujo*, la cual analiza si este se dio sobre un objeto de la clase *Enrutador*, de ser así y de cumplirse con los requerimientos del caso de uso, se activa un objeto de la clase *DialogoConfiguración*. El usuario pulsa clic sobre la opción monitorear de *DialogoConfiguracion* y selecciona el tipo de mensaje que desea monitorear.

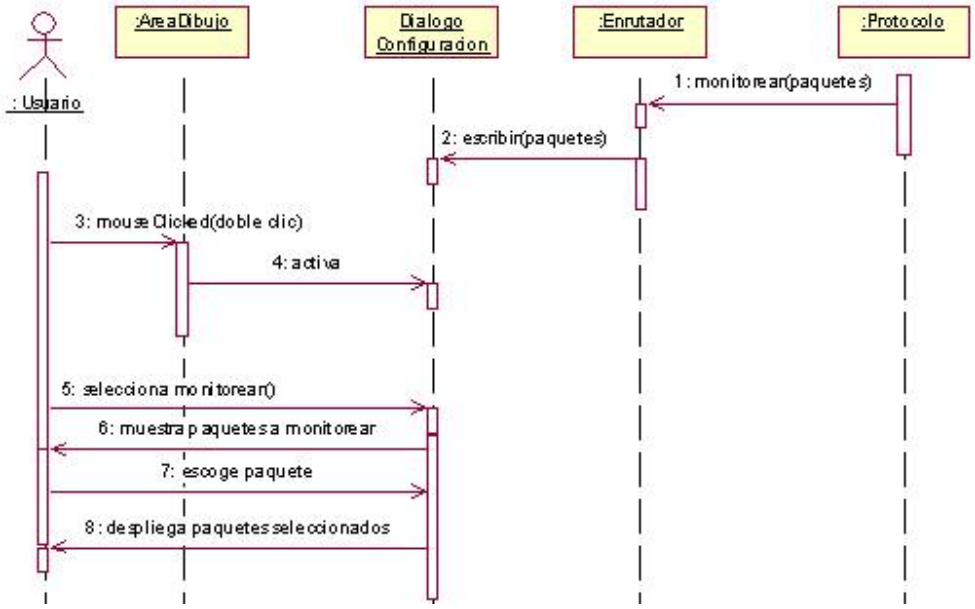

Figura B.30 Diagrama de secuencia monitorear enrutador

Empezar simulación

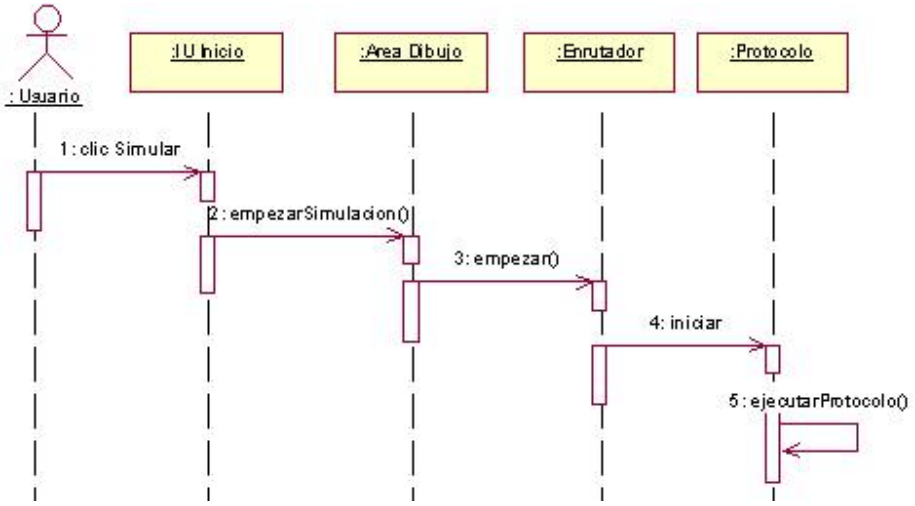

Figura B.31 Diagrama de secuencia Empezar simulación

La secuencia empieza cuando el usuario pulsa clic sobre la opción simular, la información es recogida por un objeto de la clase *IU_Inicio*, que invoca la operación empezar () ofrecida por *AreaDibujo*, la cual invoca el método empezar() de todos los objetos de la clase *Figura*. Por último se invoca la operación iniciar() ofrecida por la clase *Protocolo* quien pone en funcionamiento el protocolo para que efectúe la simulación.

2.5 Diagrama de clases

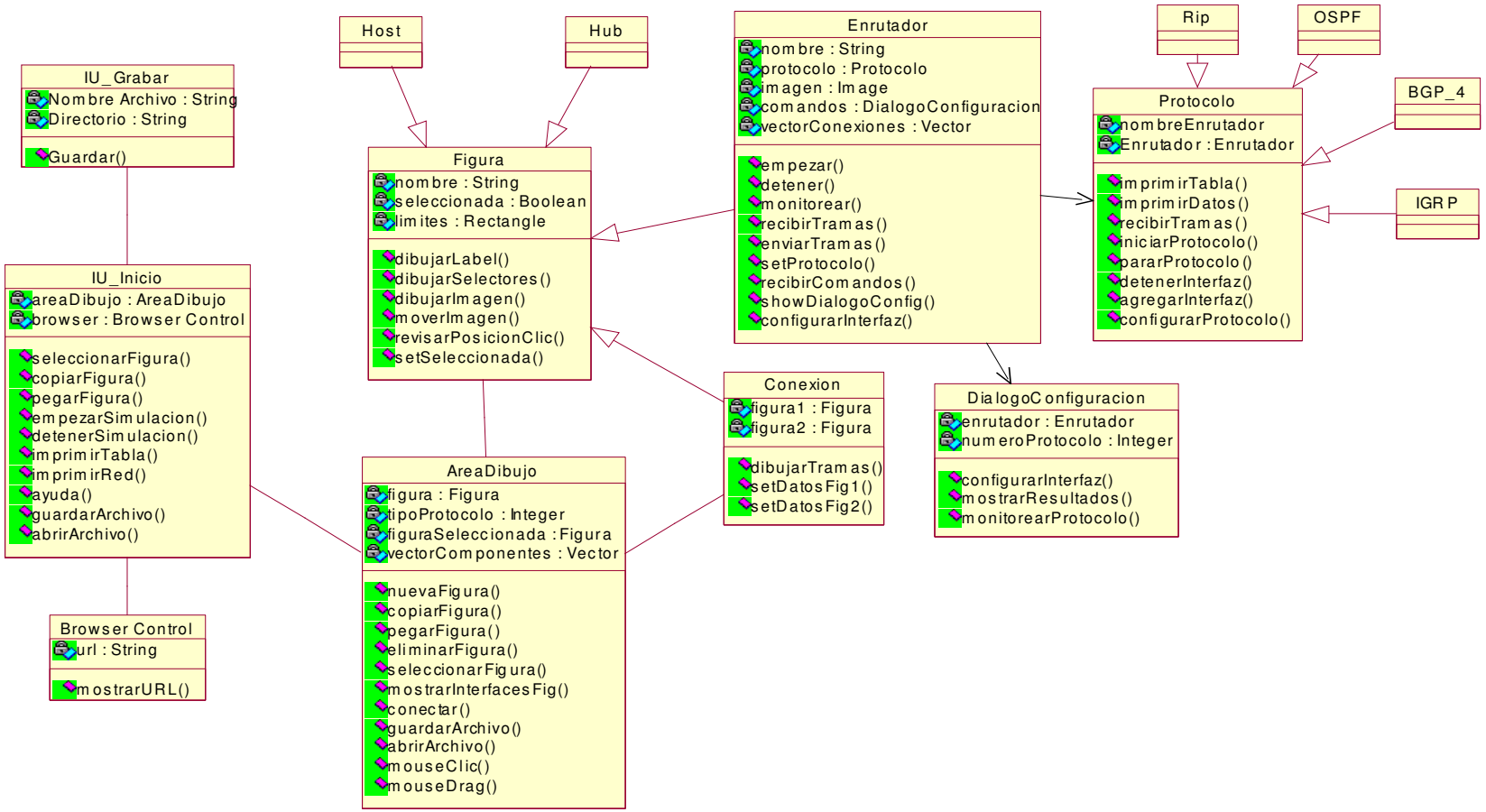

Figura B.32 Diagrama de Clases

ANEXO C. MANUAL DE USUARIO

1. Requisitos de Instalación

1.1 Requisitos Hardware

Para que la herramienta funcione correctamente necesitará como mínimo:

- Computador con procesador Pentium II
- Memoria de 128 MHz
- Máquina Virtual de Java (JDK 1.3.1)

1.2 Instalación en sistemas operativos Windows

Debe ejecutar el archivo de instalación SimRouter_windows_1_0.exe que se encuentra en el CD ROM, el cual lo guiará a través de todo el proceso para realizar una correcta instalación. No se requiere de ningún software adicional.

1.3 Instalación en otros sistemas operativos.

No se provee ningún instalador para otros sistemas operativos, pero podrá correr la herramienta con el archivo interfaz.jar que se encuentra en el CD ROM. Se requerirá además que tenga instalada la maquina virtual de JAVA correspondiente a su sistema operativo, la cual puede ser descargada de www.java.sun.com

2. Entorno del Simulador.

El simulador utiliza la ventana que se ve en la figura C.1 para realizar la mayoría de las funciones: creación de la red, selección de protocolos, inicio de la simulación, guardar archivo, imprimir, etc. Esta ventana contiene varios paneles en los que se efectúan diferentes operaciones que serán explicadas más adelante.

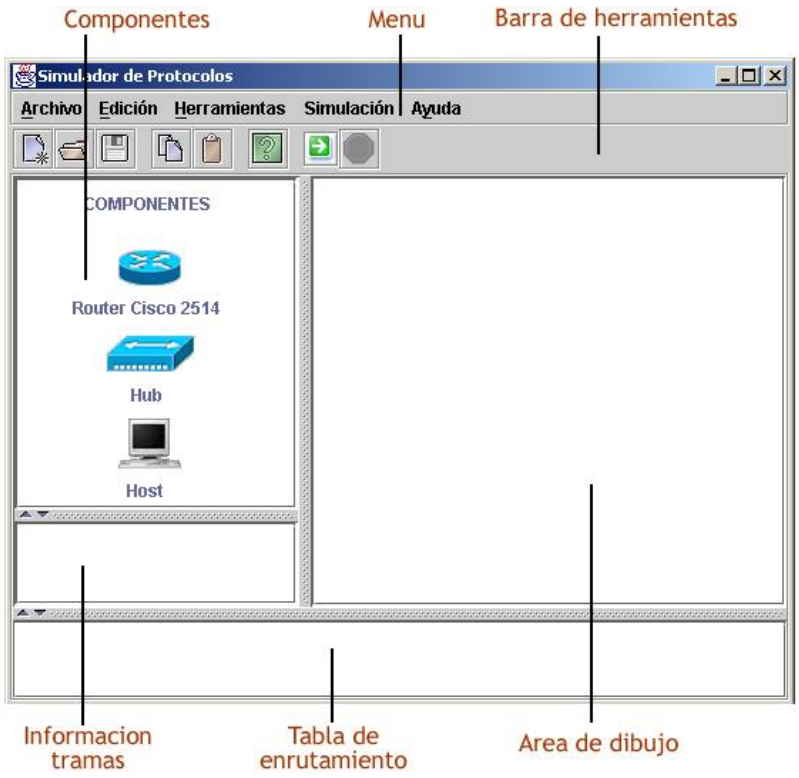

Figura C.1 Entorno del Simulador

2.1 Menú

El menú principal se encuentra en la parte superior de la aplicación. Se puede acceder a el pulsando clic con el botón izquierdo del mouse o a través de las letras aceleradoras (ej: ALT+A). A continuación se detallarán los menús desplegables de la aplicación:

Menú Archivo

Contiene comandos para crear un nuevo proyecto, guardar, abrir, imprimir y salir.

Menú	Tecla aceleradora	Descripción
Nuevo	Ctrl + U	Empieza un nuevo proyecto en blanco
Abrir	Ctrl + A	Abre un proyecto guardado anteriormente. Permite seleccionar un archivo de una lista
Guardar	Ctrl + G	Guarda el proyecto
Guardar Como		Selecciona el lugar y el nombre del archivo y guarda el proyecto

Imprimir	Ctrl + P	Imprime la red que se ha configurado
Salir		Cierra la aplicación

Menú Edición

Contiene comandos para copiar, pegar y borrar

Menú	**Tecla aceleradora**	**Descripción**
Copiar	Ctrl + C	Copia el elemento seleccionado
Pegar	Ctrl + V	Pega en el area de dibujo el elemento copiado
Borrar	Suprimir	Borra el elemento seleccionado

Menú Herramientas

Menú	**Descripción**
Insertar	Coloca en el área de dibujo el elemento seleccionado.
Configurar Protocolos	Permite configurar el modo de funcionamiento para los protocolos RIP y BGP4
Laboratorios	Muestra los laboratorios diseñados para el uso de la herramienta.

Menú Simulación

Contiene comandos para seleccionar el protocolo que se desea simular, para iniciar y detener la simulación.

Menú	Descripción
Protocolos	Selecciona el protocolo que se va simular: RIP, OSPF o BGP-4
Empezar	Empieza la simulación
Detener	Detiene la simulación

Menú Ayuda

Contiene comandos para mostrar el manual de usuario, información acerca de enrutamiento y sobre los desarrolladores del simulador.

Menú	Tecla aceleradora	Descripción
Manual de usuario	F1	Muestra una página html con información relativa al uso de la herramienta.
Enrutamiento	Ctrl + I	Muestra información acerca de los protocolos de enrutamiento
Acerca de..		Muestra información acerca de los desarrolladores del simulador

2.2 Barra de Herramientas

La barra de herramientas principal se encuentra en la parte superior de la aplicación, bajo la barra de menús. Se compone de barras de herramientas menores agrupadas por funciones: Archivo, Guardar, Empezar simulación, Detener simulación y Ayuda.

Las funciones de los botones que se encuentran en la barra de herramientas son idénticas a las funciones del menú por lo tanto no se explicarán nuevamente.

2.3 Componentes

Contiene los elementos con los que se diseñará la red. Cuenta con un enrutador con dos interfaces seriales y dos interfaces Ethertet, un hub de 5 puertos y un PC.

2.4 Area de dibujo

Es el lugar donde se dibujará la red con los elementos que se encuentran en el panel de componentes

2.5 Información Tramas

Este panel muestra los nombres de las tramas utilizadas por el protocolo seleccionado

2.6 Tabla de enrutamiento

Muestra la tabla de enrutamiento solo cuando se esta simulando el protocolo y cuando el usuario selecciona un enrutador dentro del área de dibujo

3. Creación de la red a simular

A continuación se presentarán los pasos necesarios para crear la red a simular.

- **Seleccionar un dispositivo del panel componentes**

El usuario deberá hacer clic sobre cualquiera de los dispositivos que se encuentran el panel componentes para seleccionar el dispositivo que desea colocar en el área de dibujo.

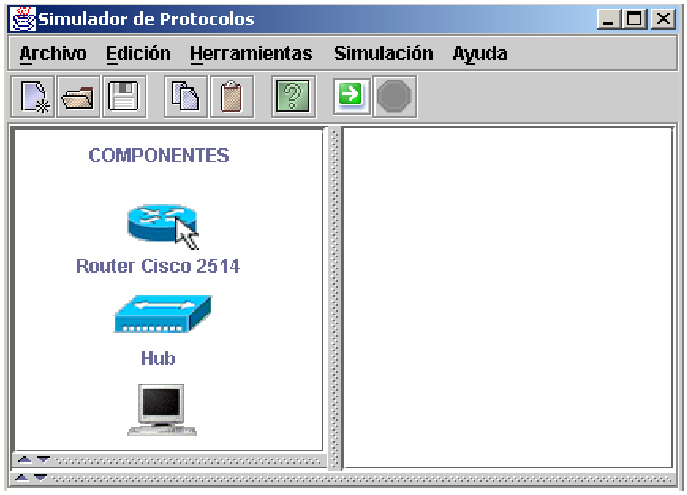

Figura C.2 Seleccionar Dispositivo

- **Clic sobre el área de dibujo**

Una vez seleccionado el dispositivo de clic sobre el área de dibujo, inmediatamente aparecerá el dispositivo con su respectivo nombre, el cual no lo podrá modificar.

Figura C.3 Dibujar Figura

➢ **Repita el proceso para dibujar todos los dispositivos que necesita**

Para este ejemplo la red quedo de la siguiente manera

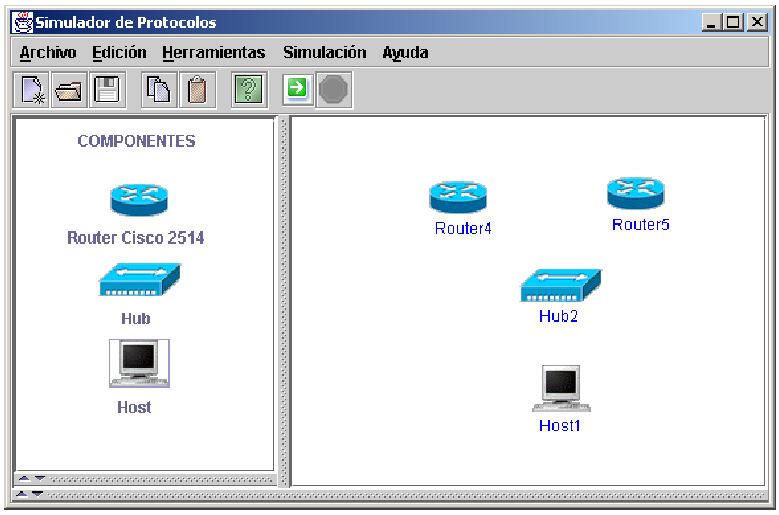

Figura C.4 Topología de ejemplo

➢ **Realizar las conexiones**

Cuando tenga todos los dispositivos con los cuales hará la configuración de la red debe realizar las conexiones. Para ello de clic con el botón derecho del mouse sobre uno de los componente que coloco en el área de dibujo. Se mostrará las interfaces del dispositivo, presione clic sobre cualquier interfaz y diríjase hacia el otro dispositivo con el cual desea realizar la conexión. Repita esta operación y habrá terminado de realizar la conexión.

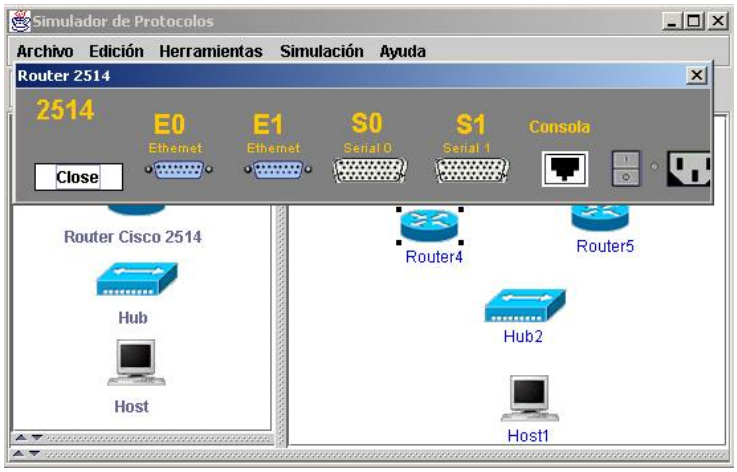

Figura C.5 Conectar Figuras

No todas las conexiones se pueden realizar. Existen algunas restricciones:

1. Solo se pueden conectar interfaces Ethernet con interfaces Ethernet, y seriales con seriales
2. No se pueden conectar dos interfaces de un dispositivo con dos interfaces de otro dispositivo
3. No se puede conectar una interfaz de un dispositivo con otra interfaz del mismo dispositivo
4. No se pueden conectar 2 hubs (Esta restricción se tomo debido a problemas en el tiempo de trasnsporte de los paquetes a través de la red)

Las conexiones para este ejemplo quedaron de la siguiente manera:

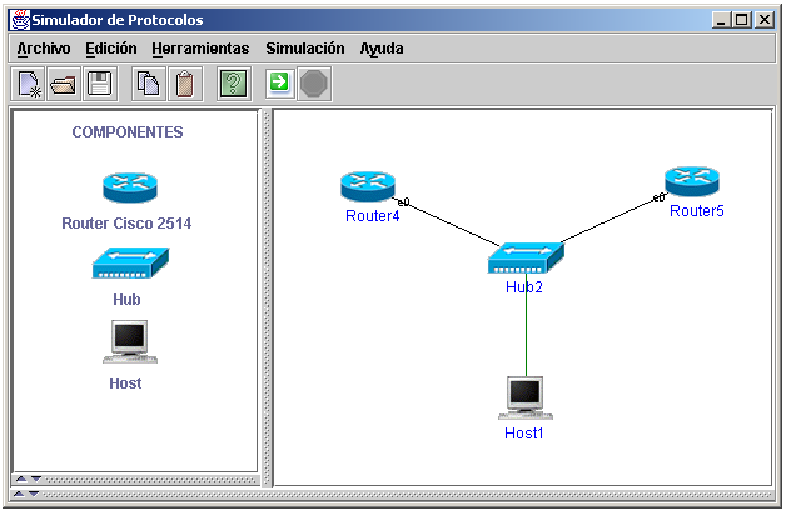

Figura C.6 Topología de ejemplo 2

4. Empezar a simular

Una vez halla dibujado la red puede empezar con el proceso de simulación. Para ello siga los siguientes pasos.

- **Seleccionar el protocolo**

En el menú Simulación, en la opción protocolo elija el protocolo que desea simular RIP, IGRP, OSPF o BGP-4

Cuando seleccione el protocolo se deshabilitara este campo, aparecerá en el panel de Información Tramas las tramas que utiliza el protocolo, y en el panel de Tabla de enrutamiento aparecerá el formato de la trama para dicho protocolo.

➢ **Configurar las interfaces**

Después de seleccionar el protocolo podrá configurar las interfaces de los enrutadores y de los host. Para ello presione doble clic sobre el dispositivo que quiere configurar, aparecerá un dialogo de Configuración que se deberá llenar apropiadamente, con valores coherentes para asegurar el buen funcionamiento de la red.

- Dirección IP: la dirección IP de la interfaz
- Máscara: la mascara de subred asociada a esa interfaz
- Métrica: podrá seleccionar el valor de la métrica para esa red
- Area o SA (Area o Sistema Autónomo): esta opción solo estará habilitada cuando este simulando OSPF o BGP-4

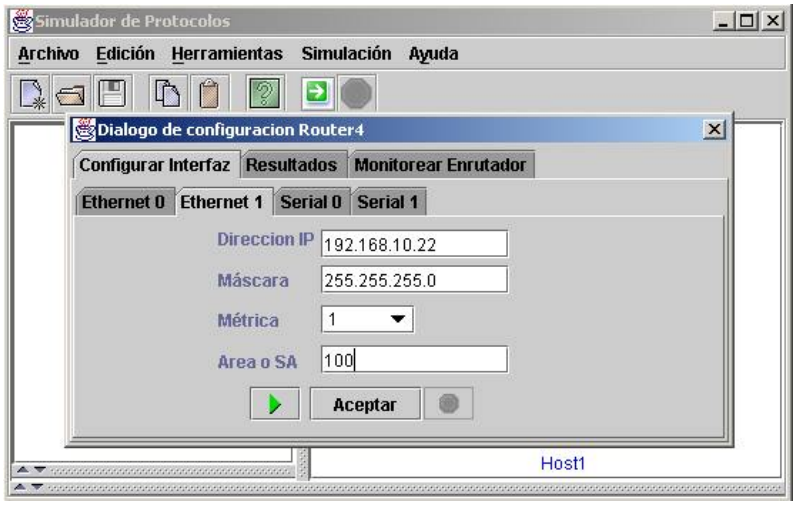

Figura C.8 Configurar Interfaces

Después de llenar los datos de clic en el botón Aceptar y la interfaz quedará configurada.
La interfaz no se podrá configurar cuando:

➢ La dirección IP que se asigna a la interfaz ya había sido asignada anteriormente a otro equipo.
➢ El formato de la dirección IP o de la máscara no sea el apropiado .
➢ No exista una conexión con esa interfaz.
➢ Cuando no ha llenado todos los datos que se solicitan.
➢ Restricción: en OSPF solo puede configurar un enrutador de frontera para un área determinada, cuando intenta configurar más de uno aparecerá un mensaje indicando este suceso.

Después de haber configurado todas las interfaces guarde su proyecto. La opción de guardar se deshabilitará cuando empiece a funcionar el protocolo.

- **Iniciar la simulación**

Puede iniciar la simulación haciendo clic sobre el botón Iniciar de la barra de herramientas El protocolo empezará a funcionar.

- **Detener la simulación**

Para detener la simulación de clic sobre el obre el botón Stop de la barra de herramientas. El protocolo dejara de funcionar.

5. Observar resultados y monitorear enrutador

Una vez comience la simulación podrá observar los resultados que arroja la herramienta a través del panel *resultados* que se encuentra en la interfaz dialogo de configuración del enrutador.

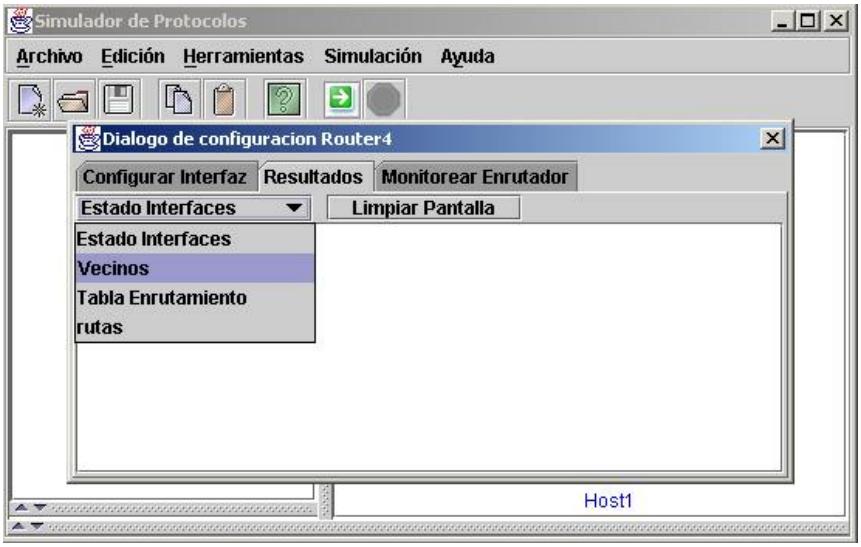

Figura C.9 Resultados de la Simulación

Podrá escoger el tipo de información que desea observar la cual variará de acuerdo al protocolo que este simulando, sin embargo las opciones estado interfaces, vecinos y tabla de enrutamiento están presentes para todos los protocolos.

Para monitorear las tramas que envía y recibe el enrutador, debe seleccionar la opción *monitorear* que se encuentra en la interfaz dialogo de configuración del enrutador.

Deberá escoger el tipo de mensaje que desea monitorear los cuales variarán dependiendo del tipo de protocolo que este simulando. Esta opción será de gran utilidad cuando desee conocer el tratamiento que el protocolo da a los diferentes mensajes.

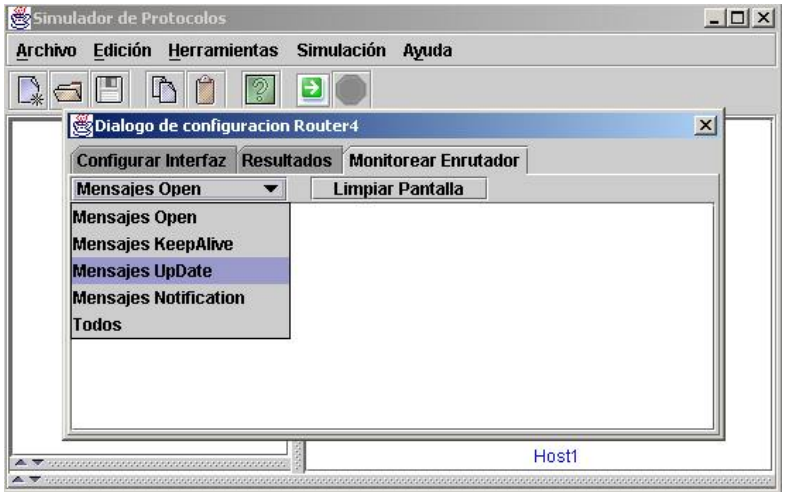

Figura C.10 Monitorear Enrutador

ANEXO D. CODIGO FUENTE

En el presente anexo, se presenta el código fuente de las principales clases que conforman el proyecto, las clases restantes se encuentran en el CD-ROM que se entrega con la monografía.

1. Clase Inicio

```
package interfaz;

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import com.borland.jbcl.layout.*;
import java.io.*;
import javax.swing.border.*;
import java.awt.print.*;
import java.util.Timer;
import java.util.TimerTask;
import java.util.Vector;
import javax.swing.table.*;

public class Inicio extends JFrame {

public Inicio() {
   try {
     jTablaEnrutamiento.disable();//la tabla se debe deshabilitar
     tk = getToolkit();
     d = tk.getScreenSize();
     this.setSize(d);
     filechooser.addChoosableFileFilter(new FiltroArchivo());
     jbInit();
   }
   catch(Exception e) {
     e.printStackTrace();
   }
 }
```

```
  public static void main(String[] args) {
    Inicio inicio = new Inicio();
    inicio.setVisible(true);
  }
  private void jbInit() throws Exception {
  }
void this_windowClosing(WindowEvent e) {
   this.dispose();
   System.exit(0);
  }

/**Informa al área de Dibujo que se desea dibujar un Enrutador*/
  void jButtonRouter_mouseClicked(MouseEvent e) {
    areaDibujo.nuevaFigura(1);
  }
  /**Informa al área de Dibujo que se desea dibujar un Hub*/
  void jButtonHub_mouseClicked(MouseEvent e) {
    areaDibujo.nuevaFigura(2);
  }
  /**Informa al área de Dibujo que se desea dibujar un Host*/
  void jButtonHost_mouseClicked(MouseEvent e) {
    areaDibujo.nuevaFigura(3);
  }

  void jMenuItemAbout_actionPerformed(ActionEvent e) {
    Acercade ad = new Acercade(this,"Información",true);
    ad.setBounds(300,200,400,270);
    ad.show();
  }

  void jMenuItemNuevo_actionPerformed(ActionEvent e) {
    int n = JOptionPane.showConfirmDialog(this,"Desea guardar los cambios?");
    if(n == JOptionPane.YES_OPTION){
      this.jMenuProtocolos.setEnabled(true);
      if(empezar){
        jButtonDetener_actionPerformed(e);
      }
      jMenuItemGuardar_actionPerformed(e);
      areaDibujo = new TestCanvas(this);
```

```
      jScrollPanePaint.getViewport().add(areaDibujo, null);
      jScrollPane1.getViewport().removeAll();
      columnNames.removeAllElements(); // se quita la tabla de enrutamiento (modelo) de la
//interfaz
      model.setColumnIdentifiers(columnNames);
      jTablaEnrutamiento.setModel(model);
      fileGuardar =null;// se hace para asegurar que se le asigne un nuevo nonbre al archivo
//que va a guardar
      jMenuConfigProto.setEnabled(false);
      jRadioButtonBgp.setSelected(true);
      jRadioButtonRip.setSelected(true);
      jCheckBoxMenuRip.setSelected(false);
      jCheckBoxMenuItem2.setSelected(false);
      jCheckBoxMenuItem3.setSelected(false);
      jCheckBoxIGRPItem1.setSelected(false);
    }else if(n == JOptionPane.NO_OPTION){
      this.jMenuProtocolos.setEnabled(true);
      if (empezar){
        jButtonDetener_actionPerformed(e);
      }
      areaDibujo = new TestCanvas(this);
      jScrollPanePaint.getViewport().add(areaDibujo, null);
      jScrollPane1.getViewport().removeAll();
      columnNames.removeAllElements();
      model.setColumnIdentifiers(columnNames);
      jTablaEnrutamiento.setModel(model);
      fileGuardar =null;
      jMenuConfigProto.setEnabled(false);
      jCheckBoxMenuRip.setSelected(false);
      jCheckBoxMenuItem2.setSelected(false);
      jCheckBoxMenuItem3.setSelected(false);
      jCheckBoxIGRPItem1.setSelected(false);
    }
    repaint();
  }

  /**Muestra un dialogo para que el usuario seleccione el archivo
   * que desea abrir, y presenta la topología de red guardada
   * en el área de Dibujo*/
  void jMenuItemAbrir_actionPerformed(ActionEvent e) {
```

```
int returnVal = filechooser.showOpenDialog(this);
if (returnVal == JFileChooser.APPROVE_OPTION) {
  try{
    if (empezar){
      jButtonDetener_actionPerformed(e);
    }
    fileGuardar = filechooser.getSelectedFile();
    FileInputStream fin = new FileInputStream(fileGuardar);
    ObjectInputStream entrada=new ObjectInputStream(fin);
    if(areaDibujo.abrir(entrada)){
      this.jMenuProtocolos.setEnabled(false);
      switch(areaDibujo.getTipoProtocolo()){
        case 1: //protocolo RIP
          columnNames.removeAllElements();
          columnNames.addElement("flag");
          columnNames.addElement("Red destino");
          columnNames.addElement("Máscara");
          columnNames.addElement("Next Hop");
          columnNames.addElement("Interfaz");
          columnNames.addElement("Metrica");
          model.setColumnIdentifiers(columnNames);
          jTablaEnrutamiento.setModel(model);
          TramasRip tramaRip = new TramasRip();
          jScrollPane1.getViewport().add(tramaRip, null);
          jMenuConfigProto.setEnabled(true);
          jMenuConfRip.setEnabled(true);
          jMenuConfBgp.setEnabled(false);
          repaint();
          break;
        case 2://Protocolo OSFF
          columnNames.removeAllElements();
          columnNames.addElement("Tipo");
          columnNames.addElement("Opcion");
          columnNames.addElement("Red");
          columnNames.addElement("Mascara");
          columnNames.addElement("Metrica");
          columnNames.addElement("Area");
          columnNames.addElement("NextHop");
          model.setColumnIdentifiers(columnNames);
          jTablaEnrutamiento.setModel(model);
```

```
            TramasOspf tramaOspf = new TramasOspf();
            jScrollPane1.getViewport().add(tramaOspf, null);
            repaint();
            break;
          case 3: //Protocolo BGP_4
            columnNames.removeAllElements();
            columnNames.addElement("origen");
            columnNames.addElement("red");
            columnNames.addElement("NextHop");
            columnNames.addElement("Path");
            columnNames.addElement("Métrica");
            columnNames.addElement("Interfaz");
            model.setColumnIdentifiers(columnNames);
            jTablaEnrutamiento.setModel(model);
            TramasBgp tramaBgp = new TramasBgp();
            jScrollPane1.getViewport().add(tramaBgp, null);
            jMenuConfigProto.setEnabled(true);
            jMenuConfRip.setEnabled(false);
            jMenuConfBgp.setEnabled(true);
            repaint();
            break;
          case 4: //Protocolo IGRP
            columnNames.removeAllElements();
            columnNames.addElement("Red");
            columnNames.addElement("Mascara");
            columnNames.addElement("NextHop");
            columnNames.addElement("Interfaz");
            columnNames.addElement("Número de saltos");
            columnNames.addElement("Métrica");
            model.setColumnIdentifiers(columnNames);
            jTablaEnrutamiento.setModel(model);
            TramasIGRP tramaIgrp = new TramasIGRP();
            jScrollPane1.getViewport().add(tramaIgrp, null);
            repaint();
            break;
        }
      }
    }catch(Exception ewx){
    }
  }
```

```
}

/**Guarda en un archivo la topología de red creda por el usuario.*/
void jMenuItemGuardar_actionPerformed(ActionEvent e) {
  if (fileGuardar == null){
    jMenuItemGuardarc_actionPerformed(e);
  }else{
    guardar();
  }
}

void jMenuItemGuardarc_actionPerformed(ActionEvent e) {
  int rta = filechooser.showSaveDialog(this);
  if (rta == JFileChooser.APPROVE_OPTION){
    String name = filechooser.getSelectedFile().getName();
    int i = name.lastIndexOf('.');
    if (i > 0 &&  i < name.length() - 1) {
      String extension = name.substring(i+1).toLowerCase();
      if (extension.equals("sim")) {
        fileGuardar = filechooser.getSelectedFile();
        guardar();
      } else {
        fileGuardar = new File(filechooser.getCurrentDirectory().getPath(),name+".sim");
        guardar();
      }
    }else{
      fileGuardar = new File(filechooser.getCurrentDirectory().getPath(),name+".sim");
      guardar();
    }
  }
}

public void guardar (){
  try{  FileOutputStream fos = new FileOutputStream(fileGuardar);
  ObjectOutputStream salida=new ObjectOutputStream(fos);
  if(areaDibujo.guardar(salida)){
    fos.close();
    JOptionPane.showMessageDialog(this,"El archivo se ha guardado satisfactoriamente");
  }else{
    JOptionPane.showMessageDialog(this,"No se pudo grabar el archivo");
```

```
    }
    }catch(Exception ex){}
  }

  /**Imprime la topología de red creada por el usuario*/
  void jMenuItemPrint_actionPerformed(ActionEvent e) {
    PrinterJob printJob = PrinterJob.getPrinterJob();
    printJob.setPrintable(areaDibujo);
    if (printJob.printDialog()) {
      try {
        printJob.print();
      } catch (Exception ex) {
        ex.printStackTrace();
      }
    }
  }

  /**Despliega la página web que contiene el Manual de Usuario*/
  void jMenuItemManual_actionPerformed(ActionEvent e) {
    String separator = System.getProperty("file.separator");
    String url = "file:" + System.getProperty("user.dir") + separator+ "ayuda"+
separator+"Manual de usuario.htm";
    bc.displayURL(url);
  }

    void jMenuItemSalir_actionPerformed(ActionEvent e) {
    System.exit(0);
  }

   /**Informa al área de dibujo que elimine la figura (Enrutador, Hub, o Host) que se enceuntra
   * seleccionada*/
  void jMenuItemBorrar_actionPerformed(ActionEvent e) {
    Figura a = areaDibujo.getFiguraSelec();
    if(a!=null){
      areaDibujo.borrar();
    }
  }

  /**Guarda la figura que se encuentra seleccionada en una variable para
   * utilizarla posteriormente cuando el usuario quiera pegarla*/
```

```
 void jMenuItemCopiar_actionPerformed(ActionEvent e) {
  figuraCopiar = areaDibujo.getFiguraSelec();
  if(figuraCopiar == null){
    JOptionPane.showMessageDialog(this,"Debe seleccionar un componente");
  }
 }

 /**Dibuja en areaDibujo la figura que se había copiado con anterioridad*/
 void jMenuItemPegar_actionPerformed(ActionEvent e) {
  if(figuraCopiar!=null){
    areaDibujo.pegar(figuraCopiar);
  }
 }

 void jMenuItemRouter_actionPerformed(ActionEvent e) {
  areaDibujo.dibujarEnrutador(0,0);
 }

 void jMenuItemHub_actionPerformed(ActionEvent e) {
  areaDibujo.dibujarHub(0,0);
 }

 void jMenuItemHost_actionPerformed(ActionEvent e) {
  areaDibujo.dibujarHost(0,0);
 }

 void jBotonNuevo_actionPerformed(ActionEvent e) {
  jMenuItemNuevo_actionPerformed(e);
 }

 void jBotonAbrir_actionPerformed(ActionEvent e) {
  jMenuItemAbrir_actionPerformed(e);
 }

 void jButtonGuardar_actionPerformed(ActionEvent e) {
  jMenuItemGuardar_actionPerformed(e);
 }

 void jButtonCopiar_actionPerformed(ActionEvent e) {
  jMenuItemCopiar_actionPerformed(e);
```

```
}

void jButtonPegar_actionPerformed(ActionEvent e) {
  jMenuItemPegar_actionPerformed(e);
}

void jButtonAyuda_actionPerformed(ActionEvent e) {
  jMenuItemManual_actionPerformed(e);
}

  /**Empieza la simulación, arranca los timers que son necesarios para
 * el funcionamiento de la herramienta, y dehabilita y habilita algunos
 * elementos de la interfaz de usuario*/

void jButtonEmpezar_actionPerformed(ActionEvent e) {
  if(areaDibujo.empezar()){
    jButtonEmpezar.setEnabled(false);
    jButtonDetener.setEnabled(true);
    jButtonGuardar.setEnabled(false);//no se puede guardar mientra se esta simulando
    jMenuItemGuardar.setEnabled(false);
    jMenuItemGuardarc.setEnabled(false);
    jMenuItemEmpezar.setEnabled(false);
    jMenuItemDetener.setEnabled(true);
    jMenuConfigProto.setEnabled(false);
    timer = new Timer();//timer que presenta las tablas
    timer.schedule(new presentarTabla(),1000,4000);
    empezar = true;
  }else{
    JOptionPane.showMessageDialog(this,"Antes de simular, seleccione un protocolo");
  }
}
void jMenuItemEmpezar_actionPerformed(ActionEvent e) {
  jButtonEmpezar_actionPerformed(e);
}

/**Detiene la simulación, cancela los timers
 * y reestablece la configuración inicial de los botones
 * @param ActionEvent Recibe el evento generado por el usuario
 * */
```

```
 void jButtonDetener_actionPerformed(ActionEvent e) {
  empezar = false;
  jButtonEmpezar.setEnabled(true);
  jButtonDetener.setEnabled(false);
  jButtonGuardar.setEnabled(true);
  jMenuItemGuardar.setEnabled(true);
  jMenuItemGuardarc.setEnabled(true);
  jMenuItemEmpezar.setEnabled(true);
  jMenuItemDetener.setEnabled(false);
  if(areaDibujo.getTipoProtocolo()==1||areaDibujo.getTipoProtocolo()==3){
    jMenuConfigProto.setEnabled(true);
  }else{
    jMenuConfigProto.setEnabled(false);
  }
  areaDibujo.detener();
  int rows = model.getRowCount();
  while (--rows >= 0) {
    model.removeRow(rows);
  }
  timer.cancel();
}

void jMenuItemDetener_actionPerformed(ActionEvent e) {
  jButtonDetener_actionPerformed(e);
}

//Rip
void jRadioButtonRip_actionPerformed(ActionEvent e) {
  if(jRadioButtonRip.isSelected()){
    areaDibujo.configurarProtocolo(0); // esta habilitada la sincronizacion
  }else{
    areaDibujo.configurarProtocolo(1); //no esta habilitada la sincronizacion
  }
}

//BGP
void jRadioButtonBgp_actionPerformed(ActionEvent e) {
  if(jRadioButtonBgp.isSelected()){
    areaDibujo.configurarProtocolo(0); // esta habilitada la sincronizacion
  }else{
```

```
      areaDibujo.configurarProtocolo(1); //no esta habilitada la sincronizacion
    }
  }

/**Selecciona el protocolo RIP, presenta los mensajes que son utilizados
 * por este protocolo y muestra los campos que utiliza en su tabla de enrutamiento*/
  void jCheckBoxMenuRip_actionPerformed(ActionEvent e) {
    columnNames.removeAllElements();
    columnNames.addElement("flag");
    columnNames.addElement("Red destino");
    columnNames.addElement("Máscara");
    columnNames.addElement("Next Hop");
    columnNames.addElement("Interfaz");
    columnNames.addElement("Metrica");
    areaDibujo.setProtocolo(1);
    TramasRip tramaRip = new TramasRip();
    jScrollPane1.getViewport().add(tramaRip, null);
    this.jMenuProtocolos.setEnabled(false);
    jButtonGuardar.setEnabled(true);
    model.setColumnIdentifiers(columnNames);
    jTablaEnrutamiento.setModel(model);
    jMenuConfigProto.setEnabled(true);
    jMenuConfRip.setEnabled(true);
    jMenuConfBgp.setEnabled(false);
    repaint();

  }

  /**Selecciona el protocolo OSPF, presenta los mensajes que son utilizados
   * por este protocolo y muestra los campos que utiliza en su tabla de enrutamiento*/
    void jCheckBoxMenuItem2_actionPerformed(ActionEvent e) {
    columnNames.removeAllElements();
    columnNames.addElement("Tipo");
    columnNames.addElement("Opcion");
    columnNames.addElement("Red");
    columnNames.addElement("Mascara");
    columnNames.addElement("Metrica");
    columnNames.addElement("Area");
    columnNames.addElement("NextHop");
    model.setColumnIdentifiers(columnNames);
```

```
  jTablaEnrutamiento.setModel(model);
  TramasOspf tramaOspf = new TramasOspf();
  jScrollPane1.getViewport().add(tramaOspf, null);
  this.jMenuProtocolos.setEnabled(false);
  jButtonGuardar.setEnabled(true);
  areaDibujo.setProtocolo(2);
}

/**Selecciona el protocolo BGP4, presenta los mensajes que son utilizados
 * por este protocolo y muestra los campos que utiliza en su tabla de enrutamiento*/
void jCheckBoxMenuItem3_actionPerformed(ActionEvent e) {
  columnNames.removeAllElements();
  columnNames.addElement("origen");
  columnNames.addElement("red");
  columnNames.addElement("NextHop");
  columnNames.addElement("Path");
  columnNames.addElement("Métrica");
  columnNames.addElement("Interfaz");
  model.setColumnIdentifiers(columnNames);
  jTablaEnrutamiento.setModel(model);
  TramasBgp tramaBgp = new TramasBgp();
  jScrollPane1.getViewport().add(tramaBgp, null);
  this.jMenuProtocolos.setEnabled(false);
  jButtonGuardar.setEnabled(true);
  areaDibujo.setProtocolo(3);
  jMenuConfigProto.setEnabled(true);
  jMenuConfBgp.setEnabled(true);
  jMenuConfRip.setEnabled(false);
}

/**Selecciona el protocolo IGRP, presenta los mensajes que son utilizados
 * por este protocolo y muestra los campos que utiliza en su tabla de enrutamiento*/
void jCheckBoxIGRPItem1_actionPerformed(ActionEvent e) {
  columnNames.removeAllElements();
  columnNames.addElement("Red");
  columnNames.addElement("Mascara");
  columnNames.addElement("NextHop");
  columnNames.addElement("Interfaz");
  columnNames.addElement("Número de saltos");
  columnNames.addElement("Métrica");
```

```
  model.setColumnIdentifiers(columnNames);
  jTablaEnrutamiento.setModel(model);
  TramasIGRP tramaIgrp = new TramasIGRP();
  jScrollPane1.getViewport().add(tramaIgrp, null);
  this.jMenuProtocolos.setEnabled(false);
  jButtonGuardar.setEnabled(true);
  areaDibujo.setProtocolo(4);
}

public void configProtocol(int tipo){
  areaDibujo.configurarProtocolo(tipo);
}

/**Método que imprime la tabla de enrutamiento en la interfaz de inicio*/
public synchronized void imprimirTabla(Vector tabla){
  //primero se borran los anteriores datos
  try{
    int rows = model.getRowCount();
    while (--rows >= 0) {
      model.removeRow(rows);
    }
    for(int i=0;i<tabla.size();i++){
      Vector imprimir = (Vector)tabla.elementAt(i);
      model.addRow(imprimir);
    }
  }catch(Exception e){
    System.err.println("error al presentar la tabla");
  }
}

class presentarTabla extends TimerTask{
  public void run(){
    Vector tabla = areaDibujo.getTabla();
    if(tabla!=null){
      imprimirTabla(tabla);
    }
  }
}

/**Muestra la página web que tiene los laboratorios diseñados para utilizarlos
```

```
   * con la herramienta*/
  void jMenuLabs_actionPerformed(ActionEvent e) {
    String separator = System.getProperty("file.separator");
    String url = "file:" + System.getProperty("user.dir") + separator+ "ayuda"+
separator+"Laboratorios.htm";
    bc.displayURL(url);
  }

  /**Despliega la página web que contiene información sobre Enrutamiento*/
    void jMenuItemEnrutamiento_actionPerformed(ActionEvent e) {
    String separator = System.getProperty("file.separator");
    String url = "file:" + System.getProperty("user.dir") + separator+ "ayuda"+
separator+"enrutamiento frame.htm";
    bc.displayURL(url);
  }
}
```

2. Clase TestCanvas

```
package interfaz;

import java.awt.Canvas;
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
import java.util.*;
import java.io.*;
import java.awt.print.*;

public class TestCanvas extends JComponent implements MouseMotionListener, MouseListener,Printable{

private int x1,y1,tipoFigura;
private java.util.Timer timer;
private boolean protocoloSelec = false;  // para saber si ya se ha elegido un protocolo a simular cuando se quiera intro otro router
private boolean corriendo = false; //variable que indicara si se encuentra realizado la //simulación

private int tipoProtocolo = 0; // 1_RIP, 2_OSPF, 3_BGP
private int numEnrutadores = 0;
private int numHub = 0;
private int numHost = 0;
private int contConexion=0; // para revisar la existencia de 2 routers
private boolean nuevaConexion=false; // para saber si esta haciendo la conexion
private int xinicio,yinicio,xfinal,yfinal;
private int configuracionProtocolo=0;//variable que permite saber que tipo de configuración //esta corriendo el protocolo para nuevos enrutadores

private Image offscreen = null;
private static final boolean doubleBuffering = false;
private Graphics og = null; // The offscreen graphics context.
private boolean nuevaFigura = false; //variable que indica que se debe dibujar una figura
private boolean dragging = false; //variable que permite arrastrar las  figuras
private boolean mousein = false; //variable que indica si el mouse se encuentra dentro del //area de dibujo
ImageIcon ImageRouter =
```

```
new ImageIcon(interfaz.TestCanvas.class.getResource("router2.gif"));
ImageIcon ImageHub = new ImageIcon(interfaz.TestCanvas.class.getResource("hub2.gif"));
private ImageIcon ImagePC =
 new ImageIcon(interfaz.TestCanvas.class.getResource("computador.gif"));
private Vector vectorEnrutador = new Vector();
private Vector vectorHub = new Vector();
private Vector vectorPC = new Vector();
private Vector vectorConexiones = new Vector();
private Vector vectorComponentes = new Vector();
private Vector vectorTodo = new Vector();
private Vector vectorDirecciones = new Vector(); //Vector que guarda las direcciones IP que
//se han cofigurado
private Vector vectorAreas = new Vector(); //Vector utilizado en OSPF para permitir que solo
//exista un router de frontera en un area
private transient Figura figuraSeleccionada = null;
private Enrutador eSelec= null;  // instancia de enrutador
private Enrutador router1=null;
private Hub sSelec=null;
private Hub Hub1=null;
private Host hSelec=null;
private int numInterfaz2=0;
private Inicio inicio;
private Figura figura2;

/**Crea una instancia de esta clase.
 * @param Inicio Recibe como parámetro inicio relacionar los dialogos de
 * la aplicación con la interfaz principal*/

  public TestCanvas(Inicio inicio){
    this.setBackground(Color.white);
    this.inicio=inicio;
    addMouseListener( this );
    addMouseMotionListener( this );
  }

/** Método utilizado por Inicio para indicar que una nueva figura va a ser
 * desplegada. Cuando se pulsa el mouse sobre los componentes estos llaman a
 * esta función y le envían un número indicando el tipo de elemento que se debe
 * dibujar así:
 * Router = 1
```

```
 * Hub = 2
 * PC = 3
 */
 public void nuevaFigura(int numfigura){
   nuevaFigura = true;
   tipoFigura=numfigura;
 }

 public void paint(Graphics g){
   if(nuevaConexion){
    g.drawLine(xinicio,yinicio,xfinal,yfinal);
   }
   Iterator vC = vectorConexiones.iterator();
   while(vC.hasNext()){
     Conexion cx = (Conexion)vC.next();
     cx.draw(g);
     cx.drawTramaFigura1(g);
     cx.drawTramaFigura2(g);
   }
   Iterator vf = vectorComponentes.iterator();
   while(vf.hasNext()){
     ((Figura) vf.next()).draw(g);
   }
 }

 public int print(Graphics g, PageFormat pf, int pageIndex){
   if (pageIndex != 0) {
     return Printable.NO_SUCH_PAGE;
   }
   paint(g);
   return Printable.PAGE_EXISTS;
 }

 /** Evento que responde cuando el usuario da clic sobre el area de dibujo.
  * Las acciones que puede tomar son las de dibujar un elemento,  la de mostrar
  * el dialogo de Configuración(doble clic), el dialogo de interfaces,
  * o de seleccionar uno de los elementos
  */
 public void mouseClicked( MouseEvent e ) {
   boolean dibujar= false;
```

```
    int x = e.getX();
    int y = e.getY();
    //esta primera parte hace referencia cuando el usuario ha dado clic sobre algun
    // componente (Router,Hub...) y da clicl sobre el area de dibujo, en este caso se debe
    //dibujar el nuevo elemento
    if(nuevaFigura){
      nuevaFigura=false;
      switch(tipoFigura){
        case 1: //JOptionPane.showInputDialog(this,"nombre del elemento","prueba");
          dibujar=true;
          dibujarEnrutador(x,y);
          break;
        case 2: dibujar = true;
          dibujarHub(x,y);
          break;
        case 3: dibujar = true;
          dibujarHost(x,y);
          break;
      }
    }else{
      //Si no es una nueva figura, debe observar si selecciono algun componente al dar clic
      boolean selecciono = false;
      vectorTodo.removeAllElements();
      vectorTodo.addAll(vectorComponentes);
      vectorTodo.addAll(vectorConexiones);
      Iterator ve = vectorTodo.iterator();
      while(ve.hasNext()){ //se busca en todos los vectores si algun componente
        Figura f = (Figura)ve.next();//se encuentra ubicado dentro del sitio donde el usuario dio
//click
        if(f.contains(e.getPoint())) { // entra al if si ha seleccionado
          boolean dobleClick=false;
          selecciono =true;
          if(! f.isSelected()){
            f.setSelected(true); //con esto se busca evitar que la figura se mueva cuando el
//usuario cierre el dialogo
            figuraSeleccionada = f; //esta es la figura que esta seleccionada y se mantiene como
//un parámetro global por si
            showTabla();
            String elemento = f.tipoElemento(); //variable que permite observar que tipo de
//elemento o componente es:Router,hub,pc
```

```
        if (e.getModifiers() == Event.META_MASK){ //mascara que identifica el boton derecho
//del mouse
          if(elemento.equals("Enrutador")){ //si es el enrutador debe mostrar el aspecto físico
//con sus interfaces
            eSelec = (Enrutador)f;
            DialogoRouter d1 = new DialogoRouter(inicio,"Router 2514",true, eSelec, this);
            d1.setBounds(e.getX(),e.getY(),530,110);
            d1.show();
          }else if(elemento.equals("Hub")){//lo mismo ocurre para los demas componente
            sSelec = (Hub)f;
            DialogoHub d2 = new DialogoHub(inicio, "Hub", true,sSelec,this);
            d2.setBounds(e.getX(),e.getY(),510,110);
            d2.show();
          }else if (elemento.equals("Host")){
            hSelec = (Host)f;
            DialogoHost d3 = new DialogoHost(null,"Host", true, hSelec, this);
            d3.setBounds(e.getX(),e.getY(),150,110);
            d3.show();
          }
          dibujar=true;
        }else{
          if(e.getClickCount()==2){ // si ha dado doble click sobre algún enrutador
            if(protocoloSelec){
              if(elemento.equals("Enrutador")){  //se debe mostrar el dialogo que posibilita que
//configure la interfaz, y que se miran
                dobleClick=true;
                Enrutador router = (Enrutador)f; //algunos aspectos de lo que esta pasando con
//los protoclos
                router.showConsola();
              }else if(elemento.equals("Host")){
                dobleClick=true;
                Host host = (Host)f;
                host.showConsola();
              }
            }else{
            JOptionPane.showMessageDialog(this,"Debe seleccionar un protocolo"); //y
//aparece su mensaje indicando este echo
            }
          }
```

```
            /* if(!dobleClick){ //si el usuario da doble click sobre un router cuando salga la figura
//no estará seleccionada
            f.setSelected(true); //con esto se busca evitar que la figura se mueva cuando el
//usuario cierre el dialogo
            figuraSeleccionada = f; //esta es la figura que esta seleccionada y se mantiene como
//un parámetro global por si
            showTabla(); //cuando seleccionan un enrutador hay que mostrar su tabla
            }*/
            dibujar = true;        //necesita realizar alguna accion sobre ella, por ejmplo copiarla o
//borrarla
          }
         }
         break;//quiebra el while evitando que se seleccione + de un componente a la vez
       }else{ // quita las selecciones
         if(f.isSelected()){
           f.setSelected(false);
           dibujar=true;
         }
       }
     }//fin de While revisar Enrutadores
     if(!selecciono&&nuevaConexion){// si se va a realizar una conexion y no se selecciona
       nuevaConexion = false;  //el elemnto con el cual se va ha conectar aparecera el //aviso
       contConexion = 0; //correspondiente
       JOptionPane.showMessageDialog(this,"No selecciono ningún componente");
       dibujar=true;
     }
   }

   if(dibujar) repaint(); //se puede quitar el repaint, debido a que lo estamos haceindo en un
//hilo
  }//end mouse clicKed

  public void mouseDragged( MouseEvent e )  {
    if(dragging&&figuraSeleccionada != null && mousein){
      figuraSeleccionada.move(e.getPoint());
      repaint();
    }
  }

  public void mousePressed( MouseEvent e ){
```

```
  if(figuraSeleccionada != null && figuraSeleccionada.contains(e.getPoint()))
    dragging = true;
 }

 public void mouseReleased( MouseEvent e ){
  dragging = false;
  if(figuraSeleccionada !=null){
    figuraSeleccionada.setSelected(false);
    figuraSeleccionada = null;
  }
  repaint();
 }

 public void mouseEntered( MouseEvent e ) {
  mousein = true;
 }

 public void mouseExited( MouseEvent e ) {
  mousein = false;
 }

 public void mouseMoved( MouseEvent e ) {
  if(nuevaConexion){
    xfinal =e.getX();
    yfinal =e.getY();
    repaint();
  }
 }

 /**Cuando el usuario selecciona un router se debe mostrar su tabla de enrutamiento
  * este metodo es llamado cuando el usuario da click*/
 public void showTabla(){
  try{
    Vector tabla = new Vector();
    if(figuraSeleccionada!=null && figuraSeleccionada.tipoElemento().equals("Enrutador")){
      Enrutador router =(Enrutador)figuraSeleccionada;
      tabla = router.getTablaEnrutamiento();
      if(tabla!=null){
        inicio.imprimirTabla(tabla);//envia la orden de imprimir la tabla
      }
```

```
    }
    }catch(Exception e){}
  }
  /**Obtiene la tabla de enrutamiento del Enrutador que esta seleccionado
   @return Tabla  Vector que contiene la tabla de Enrutamiento*/

  public Vector getTabla(){
   try{
    Vector tabla = new Vector();
    if(figuraSeleccionada!=null && figuraSeleccionada.tipoElemento().equals("Enrutador")){
     Enrutador router =(Enrutador)figuraSeleccionada;
     tabla = router.getTablaEnrutamiento();
    }
   return tabla;
   }catch(Exception e){
    Vector vacio = new Vector();
    return vacio;
   }
  }

  /**Este metodo es llamado por DialogoRouter,DialogoHost y DialogoHub cuando dan click
//sobre una
   * interfaz empezando el proceso de conectar dos componentes.
   * Aqui se revisan algunas políticas de conexion definidas, en caso de ser
   * factible la conexion se realizara, en caso contrario aparecera un mensaje
   * indicando este suceso.
   * @param figura1 Figura que se va a conectar
   * @param numInterfaz Especifíca la interfaz de la figura que se va a conectar
   */
  public void conectar (Figura figura1, int numInterfaz1){
   contConexion++;
   if (contConexion == 1){
    figura2 = figura1;
    numInterfaz2=numInterfaz1;
    xinicio = figura2.center().x;
    yinicio = figura2.center().y;
    xfinal = xinicio;
    yfinal = yinicio;
    nuevaConexion=true;
   }else{
```

```
    boolean rta=true;
    String elemento1 = figura1.tipoElemento();
    String elemento2 = figura2.tipoElemento();
    if(elemento1.equals("Enrutador")&&elemento2.equals("Enrutador")){/*Si se van a
      conectar 2 enrutadores se debe asegurar si se puede hacer la conexion*/
      rta=this.comprobar(numInterfaz1,numInterfaz2);
    }else if
((elemento1.equals("Enrutador")&&elemento2.equals("Host"))||(elemento1.equals("Host")&&e
lemento2.equals("Enrutador"))){//Host y enrutador no se pueden conectar
      rta = false;
    }else if (elemento1.equals("Hub")&&elemento2.equals("Hub")){//2 sitches no se pueden
//conectar
      rta = false;
    }else if ((elemento1.equals("Enrutador")&&elemento2.equals("Hub"))){//enrutadore con
//Hubes solo se pueden
      if(numInterfaz1==1 || numInterfaz1==2) rta=false;  //conectar con las interfaces ethernet
    }else if((elemento1.equals("Hub")&&elemento2.equals("Enrutador"))){
      if(numInterfaz2==1 || numInterfaz2==2) rta=false;
    }
    boolean rta2 = this.comprobar2(figura1.label(), figura2.label());//Compara si no existe una
//relacion entre esos dos dispositivos
    if (rta && !figura1.label().equals(figura2.label())&& rta2){   //No permite que se conecte el
//mismo elemento con el mismo
      contConexion = 0;
      nuevaConexion = false;
      Conexion c = new Conexion(numInterfaz1,numInterfaz2,figura1,figura2);//se realiza la
//nueva conexion que compartiran los dos componentes
      cambiar(figura1,numInterfaz1,c); //se debe cambiar los iconos de las interfaces para
//mostrar que ya existe una conexion
      cambiar(figura2,numInterfaz2,c);
      if(corriendo){
        c.empezar(); //si esta corriendo se empieza el hilo que dibuja las tramas
      }
      vectorConexiones.addElement(c);
    }else{ //si no se pudo realizar la conexion se muestra el respectivo mensaje
      contConexion = 0;
      nuevaConexion = false;
      JOptionPane.showMessageDialog(this,"La conexión no se pudo realizar","Revisar tipo
de conexión",JOptionPane.INFORMATION_MESSAGE);
    }
```

```
  }
 }

  //No se coloca a correr a enrutador porque necesita
 //una configuración previa, los demás componentes si corren de una.
 /**Dibuja un enrutador.
  * @paramas x Posición en el eje de las X donde se debe dibujar el enrutador
  * @paramas y Posición en el eje de las Y donde se debe dibujar el enrutador*/

 public void dibujarEnrutador( int x, int y){
   numEnrutadores++;
   String nombre = "Router"+numEnrutadores;
   Enrutador enrutador =
new Enrutador(nombre,x,y,ImageRouter.getIconWidth(),ImageRouter.getIconHeight());
   enrutador.setCanvas(this,inicio);
   vectorEnrutador.addElement(enrutador);
   vectorComponentes.addElement(enrutador);
   if(protocoloSelec){
     enrutador.setProtocolo(tipoProtocolo);
     enrutador.configProtocolo(configuracionProtocolo);
   }
   if(corriendo){ //si ya esta corriendo el protocolo
     enrutador.empezar();
   }
 }

 public void dibujarHub( int x, int y){
   numHub++;
   String nombre = "Hub"+numHub;
   Hub Hub = new Hub(nombre,x,y,ImageHub.getIconWidth(),ImageHub.getIconHeight());
   vectorHub.addElement(Hub);
   vectorComponentes.addElement(Hub);
   if(corriendo){
     Hub.empezar();
   }
 }

 public void dibujarHost(int x, int y){
   numHost++;
   String nombre="Host"+numHost;
```

```
  Host pc = new Host(nombre,x,y,ImagePC.getIconWidth(),ImagePC.getIconHeight());
  vectorComponentes.addElement(pc);
  vectorPC.addElement(pc);
  pc.setCanvas(this,inicio);
  if(protocoloSelec){
    pc.crearDialogo();
  }
}

 * Método utilizado para cambiar los iconos a las interfaces
 * cuando se crea o se borra una conexion.
 *  */
public void cambiar(Figura figura,int numInterfaz,Conexion c){
  String elemento = figura.tipoElemento();
  if(elemento.equals("Enrutador")){
    cambiarIconosRouter(figura,numInterfaz,c);
  }else if (elemento.equals("Hub")){
    cambiarIconosHub(figura,numInterfaz,c);
  }else if(elemento.equals("Host")){
    cambiarIconosHost(figura,numInterfaz,c);
  }
}

public void cambiarIconosRouter(Figura figura,int numInterfaz,Conexion c){
  Enrutador router = (Enrutador)figura;
  switch (numInterfaz){
    case 0: if (router.getInterfaz("E0")) router.setInterfaz("E0",false,c);
      else router.setInterfaz("E0",true,c);
      break;
    case 1: if (router.getInterfaz("S0")) router.setInterfaz("S0",false,c);
      else router.setInterfaz("S0",true,c);
      break;
    case 2: if (router.getInterfaz("S1")) router.setInterfaz("S1",false,c);
      else router.setInterfaz("S1",true,c);
      break;
    case 3: if (router.getInterfaz("E1")) router.setInterfaz("E1",false,c);
      else router.setInterfaz("E1",true,c);
      break;
  }
}
```

```
  public void cambiarIconosHub(Figura figura,int numInterfaz,Conexion c){
    Hub sw = (Hub)figura;
     switch (numInterfaz){
       case 0: if (sw.getInterfaz0()){
               sw.setInterfaz0(false,c);
               }else{sw.setInterfaz0(true,c);}
               break;
       case 1: if (sw.getInterfaz1()){
               sw.setInterfaz1(false,c);
               }else{sw.setInterfaz1(true,c);}
               break;
       case 2: if (sw.getInterfaz2()){
               sw.setInterfaz2(false,c);
               }else{sw.setInterfaz2(true,c);}
               break;
       case 3: if (sw.getInterfaz3()){
               sw.setInterfaz3(false,c);
               }else{sw.setInterfaz3(true,c);}
               break;
       case 4: if (sw.getInterfaz4()){
               sw.setInterfaz4(false,c);
               }else{sw.setInterfaz4(true,c);}
               break;
      }
  }

  public void cambiarIconosHost(Figura figura,int numInterfaz,Conexion c){
    Host host = (Host)figura;
    if(host.getInterfaz0()){
    host.setInterfaz0(false,c);
    }else{host.setInterfaz0(true,c);
    }
  }
/**
Método utilizado para comprobar si las conexiones entre router son viables.
Solo se pueden conectar interfaces seriales con seriales e interfaces ethernet con ethernet.
S0=1, S1=2, E0=0, E0=3.
@param I1 Especifica el número de la  interfaz que se va a conectar
@param I2 Especifica el número de la otra interfaz que se va a conectar
```

```
@return true Si se puede realizar la conexión
@return false Si no se puede realizar la conexión
*/
  public boolean comprobar (int I1, int I2){
    boolean comprobacion=false;
    int i1=I1;
    int i2=I2;
    if ((i1 == 0 && i2 == 0)||(i1 == 1 && i2 == 1)||(i1 == 2 && i2 == 2)||(i1 == 3 && i2 == 3)
        ||(i1 == 1 && i2 == 2)||(i1 == 2 && i2 == 1)||(i1 == 0 && i2 == 3)||(i1 == 3 && i2 == 0) ){
      comprobacion = true;
    }else{
      comprobacion=false;
    }
    return comprobacion;
  }

/**Metodo que busca si existe una conexion anterior entre las dos figuras
 * @param a Nombre de la primera figura
 * @param b Nombre de la segunda figura
 * @return true - Si no existe ninguna conexión entre las dos figuras
 * @return flase - Si ya existe una conexión entre las dos figuras*/

  public boolean comprobar2(String a, String b){
    boolean res=true;
    Iterator vC = vectorConexiones.iterator();
     while(vC.hasNext()){
      Conexion c = ((Conexion)vC.next());
      String fig1= c.getNombreFigura1();
      String fig2 = c.getNombreFigura2();
      if ((fig1.equals(a)&&fig2.equals(b))||(fig1.equals(b)&&fig2.equals(a))){
      res=false;
      break;
      }
    }
    return res;
  }

/**Devuelve la figura que se encuentra seleccionada
 * @return figuraSeleccionda La figura que el usuario ha seleccionado*/
  public Figura getFiguraSelec(){
```

```
  return figuraSeleccionada;
  }
/**
 Borra la figura que se encuentra seleccionada y sus respectivas conexiones, para
 esto se borran de los vectores los elementos, se repinta el area de trabajo
 y se redibujan los iconos de las interfaces
 */
  public void borrar(){
  String tipo = figuraSeleccionada.tipoElemento();
  if(tipo.equals("Enrutador")){
  Enrutador router = (Enrutador)figuraSeleccionada;
   if(router.getEstado()){//si esta corriendo no se debe dejar borrar
    if(router.getNumInterfazConfiguradas()!=0){
      JOptionPane.showMessageDialog(inicio,"No se puede eliminar el enrutador, \n debido a
      que se encuentra activo ","restriccion", JOptionPane.INFORMATION_MESSAGE);
      return;
    }
   }else{ //antes de borrar el router se deben borrar las direcciones IP configuradas para el
    Vector borrarDir = router.getDirecciones();
    vectorDirecciones.removeAll(borrarDir);//borra las direcciones IP configuradas para el
//router
    if(tipoProtocolo==2){//si es OSPF
      String rta = router.getABR();
      if(!rta.equals("no")){ //si la respuesta es diferente de no
        vectorAreas.removeElement(rta);
      }
    }
   }
  }else if(tipo.equals("Hub")){
   Hub hub = (Hub)figuraSeleccionada;
   if(hub.getEstado()){ //si esta encendida
     if(hub.getNumConexiones()!=0){
       JOptionPane.showMessageDialog(inicio,"No se puede eliminar el Hub, \n esta
conectado a otros dispositivos que pueden estar corriendo", "restriccion",
JOptionPane.INFORMATION_MESSAGE);
       return;
     }
   }
  }
```

```
  Iterator iteratorComponentes = vectorComponentes.iterator();
  Iterator iteratorConexiones = vectorConexiones.iterator();
  Vector borrarConexiones = new Vector();
  Conexion conexion;
  boolean encontro = false;
  String  labelElemento=figuraSeleccionada.label();
  while (iteratorComponentes.hasNext()){
  Figura figuraBusqueda = (Figura)iteratorComponentes.next();
  if(figuraSeleccionada.label().equals(figuraBusqueda.label())){
    vectorComponentes.removeElement(figuraSeleccionada);
    break;
  }
  }
  while (iteratorConexiones.hasNext()){
    conexion = (Conexion)iteratorConexiones.next();

if(conexion.getNombreFigura1().equals(labelElemento)||conexion.getNombreFigura2().equals
(labelElemento)){
      borrarConexiones.addElement(conexion);
      conexion.detener(); //y también se detiene las conexiones relacionadas con la figura
      encontro = true;
    }
  }
  if (encontro){
  vectorConexiones.removeAll(borrarConexiones);
  Iterator borrarInterfaces = borrarConexiones.iterator();
  while(borrarInterfaces.hasNext()){
    conexion = (Conexion)borrarInterfaces.next();
    StringTokenizer token = new StringTokenizer(conexion.label(),"*");
    int interfaz1 = (Integer).valueOf(token.nextToken()).intValue();
    int interfaz2 = (Integer).valueOf(token.nextToken()).intValue();
    if(conexion.getNombreFigura1().equals(labelElemento)){
      Figura figura2 = conexion.getFigura2();
       cambiar(figura2,interfaz2,null);

    }else{//cambiar figura 1
      Figura figura1 = conexion.getFigura1();
       cambiar(figura1,interfaz1,null);
    }
  }
```

```
 }
 repaint();
 }

/**Dibuja la figura que había sido copiada anteriormente*/

public void pegar(Figura figuraCopiar){
 String tipo = figuraCopiar.tipoElemento();
 if(tipo.equals("Enrutador")){
  dibujarEnrutador(0,0);
 }else if(tipo.equals("Hub")){
  dibujarHub(0,0);
 }else if (tipo.equals("Host")){
  dibujarHost(0,0);
 }
 repaint();
 }

 /**Metodo utilizado cuando se va abrir el archivo para cargar en los
  * enrutadores los datos necesarios para su correcto funcionamiento*/

 public void loadDatos(){
  Iterator it = vectorEnrutador.iterator();
  int x =0;
  while(it.hasNext()){
   x++;
  Enrutador enrutador = (Enrutador)it.next();
  enrutador.cargarDatos();

  }
  it = vectorPC.iterator();
  while(it.hasNext()){
   Host pc = (Host)it.next();
   pc.cargarDatos();
  }
 }

 public void loadCanvas(){
  Iterator it = vectorEnrutador.iterator();
  while(it.hasNext()){
```

```
   Enrutador enrutador = (Enrutador)it.next();
   enrutador.setCanvas(this,inicio);
   }
   it = vectorPC.iterator();
   while(it.hasNext()){
   Host pc = (Host)it.next();
   pc.setCanvas(this,inicio);
   }
  }

/**Indica a todos los enrutadores que se han dibujado el tipo
 * de protocolo que deben ejecutar.
 * @param protocolo  Número que indica el tipo de protocolo
 * que se debe ejecutar*/

  public void setProtocolo(int protocolo){
  tipoProtocolo=protocolo;
  Iterator it = vectorEnrutador.iterator();
  while (it.hasNext()){
    Enrutador enrutador = (Enrutador)it.next();
    enrutador.setProtocolo(tipoProtocolo);
  }
  it = vectorPC.iterator();
  while(it.hasNext()){
   Host host =(Host)it.next();
   host.crearDialogo();
  }
  protocoloSelec = true;
  }

  /**Guarda en un archivo la configuracion que el usuario ha realizado
   * de la red, con las direcciones, máscaras y demás datos necearios
   * para que puedan ser utilizados la siguiente oportunidad*/

  public boolean guardar(ObjectOutputStream salida){
  boolean hecho = true;
  try{
    Iterator componentesGuardar;
    componentesGuardar = vectorComponentes.iterator();
    int tamaño;
```

```
    tamaño = vectorComponentes.size();
    salida.writeInt(numEnrutadores);
    salida.writeInt(numHub);
    salida.writeInt(numHost);
    salida.writeInt(tamaño);
    if(protocoloSelec) salida.writeBoolean(true);    // si ha seleccionado un protocolo se guarda
para deshabilitar la seleccion de otros protocolos
    else salida.writeBoolean(false);
    salida.writeInt(tipoProtocolo);
    while(componentesGuardar.hasNext()){//se guardan los router, hubs, etc...
     salida.writeObject(componentesGuardar.next());
    }
    componentesGuardar=vectorConexiones.iterator();//se guardan las conexiones
    tamaño = vectorConexiones.size();
    salida.writeInt(tamaño);
    while(componentesGuardar.hasNext()){
     salida.writeObject(componentesGuardar.next());
    }
    Iterator it = vectorDirecciones.iterator();//se guardan las direcciones IP
    tamaño = vectorDirecciones.size();
    salida.writeInt(tamaño);
    while(it.hasNext()){
    salida.writeObject(it.next());
    }
    it = vectorAreas.iterator(); //se guardan las areas o los SA
    tamaño = vectorAreas.size();
    salida.writeInt(tamaño);
    while(it.hasNext()){
    salida.writeObject(it.next());
    }
    salida.flush();
  }catch(Exception es){
    System.err.println(es);
    hecho = false;
    }
  return hecho;
  }

  /**Recupera el archivo guardado por el usuario y presenta en
   * pantalla la configuración de red con sus respectivos datos
```

```
 * @param ObjectInputStream Canal de entrada del fichero que contiene la configuración
 * de la topología de red.
 * @exception Lanza una excepción si no puede abrir el archivo*/

public boolean abrir(ObjectInputStream entrada){
 boolean selecciono=false; //variable que indica si se ha seleccionado un protocolo para
//deshabilitarla en el menu
 vectorComponentes.removeAllElements();
 vectorConexiones.removeAllElements();
 vectorEnrutador.removeAllElements();
 vectorHub.removeAllElements();
 vectorPC.removeAllElements();
 vectorDirecciones.removeAllElements();
 vectorAreas.removeAllElements();
 try{
 int tamaño;
 numEnrutadores = entrada.readInt();
 numHub = entrada.readInt();
 numHost = entrada.readInt();
 tamaño = entrada.readInt();
 selecciono = entrada.readBoolean();
 protocoloSelec = selecciono;
 tipoProtocolo = entrada.readInt();
 for(int a=0;a<tamaño;a++){  //se recuperan los componentes
 Figura x = (Figura)entrada.readObject();
 vectorComponentes.addElement(x);
  if(x.tipoElemento().equals("Enrutador"))
    vectorEnrutador.addElement(x);
  if(x.tipoElemento().equals("Hub"))
    vectorHub.addElement(x);
  if(x.tipoElemento().equals("Host"))
    vectorPC.addElement(x);
 }
 tamaño = entrada.readInt();//se recuperan las conexiones
 for(int a=0;a<tamaño;a++){
  Figura x = (Figura)entrada.readObject();
  vectorConexiones.addElement(x);
 }
 tamaño = entrada.readInt(); //se recuperan las direcciones IP
 for(int a =0;a<tamaño;a++){
```

```
    String dir = (String)entrada.readObject();
    vectorDirecciones.addElement(dir);
   }
   tamaño = entrada.readInt();//se recuperan las areas
   for(int a =0;a<tamaño;a++){
    String areaAS = (String)entrada.readObject();
    vectorAreas.addElement(areaAS);
   }
   }catch(Exception n){
    System.err.println("error al abrir el archivo");
   }
   loadCanvas();
   setProtocolo(tipoProtocolo);
   loadDatos();
   repaint();
   return selecciono;
}

public int getTipoProtocolo(){
 return tipoProtocolo;
}

/**Este método es invocado cuando el usuario da clic sobre el botón play que
 * se encuentra en la barra de componentes. Se encarga de inicializar todos
 * los componentes y también crea una tarea que refresca la pantalla cada
 * segundo, para de esta forma poder observar las tramas
 * @return true Si la simulación empieza
 * @return false Si no se ha podido iniciar la simulación*/

public boolean empezar(){
   boolean empezo = false;
   if(protocoloSelec){
   corriendo = true;
   timer = new java.util.Timer();
   timer.schedule(new Repintar(this),1000,1000);

   Iterator it = vectorComponentes.iterator();
   while(it.hasNext()){
   Figura f =(Figura)it.next();
   f.empezar();
```

```
  }
  it = vectorConexiones.iterator();
  while(it.hasNext()){
    Conexion c = (Conexion)it.next();
    c.empezar();
  }
  empezo=true;
  }
  return empezo;
}

/**Llama a los metodos detener de todos los componentes para que dejen
 * de funcionar, y también cancela el timer que refresaca la pantalla*/

public void detener(){
  corriendo = false;
  Iterator it = vectorComponentes.iterator();
  while(it.hasNext()){
  Figura f =(Figura)it.next();
  f.detener();
  }
  it = vectorConexiones.iterator();
  while(it.hasNext()){
    Conexion c = (Conexion)it.next();
    c.detener();
  }
  timer.cancel();
  repaint();
}
public class Repintar extends TimerTask{
  private TestCanvas tc;
  public Repintar(TestCanvas tc){
   this.tc = tc;
 }
public void run(){
  tc.repaint();
}
}
```

```
 public void configurarProtocolo(int tipo){
 configuracionProtocolo = tipo;
 Iterator it = vectorEnrutador.iterator();
 while(it.hasNext()){
  Enrutador router = (Enrutador)it.next();
  router.configProtocolo(tipo);
 }
 }

 public boolean buscarDireccion(String dirIP){
  if(vectorDirecciones.contains(dirIP))  return true;
  else  return false;
 }

 public void insertarDireccion(String dirIP){
   vectorDirecciones.addElement(dirIP);
 }

 public void borrarDireccion(String dirIP){
   vectorDirecciones.remove(dirIP);
 }

 public boolean buscarArea (String area){
 if(vectorAreas.contains(area)) return true;
 else return false;
 }
 public void insertarArea(String area){
   vectorAreas.addElement(area);
 }
 public void borrarAreas(String area){
 vectorDirecciones.remove(area);
 }
}
```

3. Clase Enrutador

```
package interfaz;

import java.awt.*;
import javax.swing.ImageIcon;
import java.util.*;
import Protocolo.*;

public class Enrutador extends Figura {
ImageIcon imagen;
private int protocoloSelecc =0, tipoConfiguracion=-1;
private boolean empezar = false;
private boolean corriendo = false;
private Vector conexiones = new Vector();
/**Vector que contiene el nombre de las interfaces con su respectiva Conexion
 [[nomInterfaz,Conexion]]
 */
private Vector vectorInterfaces = new Vector();//[[nomInterfaz,Conexion]]
private transient protocolo proto;
private transient TestCanvas canvas;
private transient Inicio inicio;
private transient DialogoConfiguracion comandos;
private String labelAreaAS="";

/**Crea una instancia de enrutador. Inicializa la imagen que representa a esta
 * figura*/

  public Enrutador(String label, int x, int y, int w, int h) {
      super(label, x, y, w, h);
      imagen = new ImageIcon(interfaz.Enrutador.class.getResource("router2.gif"));
      this.setProtocolo(protocoloSelecc);
  }

/**Dibuja la imagen que representa al Enrutador.*/
  public void drawImage(Graphics g) {
   Graphics2D g2=(Graphics2D)g;
   Point p = super.location();
   g2.drawImage(imagen.getImage(),p.x,p.y,null);
```

```
 }

 public void fillShape(Graphics g) {
   /**@todo: implement this protosim.Figura abstract method*/
 }
 public void drawShape(Graphics g) {
 }

/**Dibuja el nombre del enrutador (Ej: Router 1)*/

 public void drawLabel(Graphics g){
g.drawString(this.label(), this.getBounds().x + 5, this.getBounds().y +  this.getBounds().height
+18);
   if(!labelAreaAS.equals("")){ // Si se esta ejecutando el protocolo OSPF o BGP
     Font fontOld = g.getFont(); //se dibuja el Router ABR, o el Sistema Autónomo al
     Color colorOld = g.getColor(); //cual pertenece el enrutador
     g.setColor(Color.black);
     g.setFont(new Font("Arial",Font.BOLD,12));
     g.drawString(labelAreaAS, this.getBounds().x, this.getBounds().y - 10);
     g.setFont(fontOld);
     g.setColor(colorOld);
   }
 }

 public String tipoElemento(){
 return ("Enrutador");
 }
 public void drawAreaAS(String areaAS){
 labelAreaAS=areaAS;
 }

 /**Este método es llamado desde TestCanvas, y su objetivo es el de guardar
  * en un vector las conexiones que tiene el router por interfaz, además
  * de eso  permite cambiar los iconos de la parte posterior del router
  * @param nomInterfaz Ej: E0, E1, S0, S1
  * @param estado TRUE  Representa si se esta haciendo la conexión
  * @param estado FALSE  Representa que se ha desconectado la interfaz
  * @param Conexion Cada interfaz esta relacionada con una conexión para
  * poder enviar o recibir las tramas.
  * */
```

```
public void setInterfaz(String nomInterfaz,boolean estado,Conexion c){
 if(estado){
 Vector temp = new Vector();//se crea un nuevo vector que contendra la relacion
 temp.addElement(nomInterfaz);
 temp.addElement(c);
 vectorInterfaces.addElement(temp);//y se adiciona al vectorInterfaces
 }else{//si el estado es falso se borraran las entradas de esa interfaz
  for(int i=0;i<vectorInterfaces.size();i++){
    Vector temp = (Vector)vectorInterfaces.elementAt(i);
    String nInterfaz = (String)temp.elementAt(0);
    if(nomInterfaz.equals(nInterfaz)){
      vectorInterfaces.removeElementAt(i);
      break;
    }
  }
  //y también se elimina del vectorConexiones
  for(int i=0;i<conexiones.size();i++){
  Vector temp =(Vector)conexiones.elementAt(i);
  String nInterfaz=(String)temp.elementAt(3);
  if(nomInterfaz.equals(nInterfaz)){
    conexiones.removeElementAt(i);
    break;
  }
  }
 }
}

/**Método utilizado en el proceso de configuracion,para saber si la interfaz
 que se esta configurando tiene un conexión establecida con otro dispositivo o no
 @param nomInterfaz El nombre de la interfaz
 @return false Si la interfaz no tiene una conexión*/

public boolean getInterfaz(String nomInterfaz){
  boolean encontro=false;
 for(int i=0;i<vectorInterfaces.size();i++){
  Vector temp = (Vector)vectorInterfaces.elementAt(i);
  String nInterfaz = (String)temp.elementAt(0);
  if(nomInterfaz.equals(nInterfaz)){
    encontro = true;
    break;
```

```
    }
   }
  return encontro;
 }

 /**Método que se utiliza para saber si el enrutador se encuentra o no
  ejecutando un protocolo de Enrutamiento
  @return false No esta ejecutando ningún protocolo*/

  public boolean getEstado(){
  return corriendo;
  }

  /**Metodo que devuelve el número de Interfaces que se han configurado
   hasta el momento
   @return int El número de interfaces que se ha configurado hasta el momento*/

  public int getNumInterfazConfiguradas(){
  return conexiones.size();
  }

 /**Metodo que guarda la configuracion de las interfaces
  * @param nomInterfaz El nombre de la interfaz
  * @param dirIP: la dirección IP asociada a esa interfaz
  * @param Máscara: la máscara de la direccion IP
  * @param Red: la red a la cual esta conectada la interfaz
  * @param Métrica: el costo de usar ese enlace
  * @param AreaoAS: dependiendo del tipo de protocolo se requiere o no este parámetro
  * @return true Si se ha podido configurar la interfaz
  * */

  public boolean setConfiguracion(String nomInterfaz,String dirIP,String mascara,String
   red,String metrica, String areaoAS){
    //se debe borrar primero la configuracion anterior para esa interfaz
    boolean realizado = false;
   for(int i=0;i<conexiones.size();i++){
    Vector temp = (Vector)conexiones.elementAt(i);
    String inter = (String)temp.elementAt(3);
    if(inter.equals(nomInterfaz)){ //si encuentra la interfaz tiene que borrarla
    conexiones.removeElementAt(i);
```

```
    canvas.borrarDireccion((String)temp.elementAt(1));//se borra la direccion IP que estaba
configurada antes
    break;
    }
   }
   //después de haber borrado(es posible que no haya encontrado nada)la configuración
//anterior se introduce la nueva
   if(getInterfaz(nomInterfaz)){ //si se ha realizado una conexion con dicha interfaz
     //es posible seguir con el proceso de configuracion y guardar los datos
     Conexion c=null;
     for(int i=0;i<vectorInterfaces.size();i++){
     Vector temp = (Vector)vectorInterfaces.elementAt(i);
     String nInterfaz = (String)temp.elementAt(0);
     if(nomInterfaz.equals(nInterfaz)){
       c = (Conexion)temp.elementAt(1);
       break;
     }
    }
    //se agrega al vector conexiones la conexion con su respectiva configuracion
     if(protocoloSelecc==4)c.setDirecciones(nomInterfaz.toLowerCase()+"
"+dirIP,this.label());//dibuja las direcciones IP de las interfaces sobre la conexion
     else  c.setDirecciones(nomInterfaz.toLowerCase()+"  "+dirIP+"("+metrica+")",this.label()  );
//dibuja  las direcciones IP de las interfaces y la métrica sobre la conexion
     Vector relacion = new Vector();
     relacion.addElement(c);
     relacion.addElement(dirIP);
     relacion.addElement(mascara);
     relacion.addElement(nomInterfaz);//nombre de la interefaz Ej, E0,S1....
     relacion.addElement("Inactiva");
     relacion.addElement(areaoAS);
     relacion.addElement(metrica);
     relacion.addElement(red);
     conexiones.addElement(relacion);
     canvas.insertarDireccion(dirIP);//se adiciona al vectorDirecciones de Canvas la nueva
      //direccion IP
     realizado=true;
   }
   return realizado;
  }
 /**Metodo que busca en el Area de dibujo si ya existe la direccion IP que se
```

```
 esta configurando, o si ya existe un router de frontera
 @param 1 Busca la dirección IP
 @param 2 Busca el área
 @param parámetro El parámetro a buscar.
 @return true Si la búsqueda arrojo resultados positivos
 */

 public boolean buscarenCanvas(int opcion,String parametro){
 boolean resultado = true;
 try{
 switch(opcion){
 case 1://buscar direccion IP en canvas
       resultado = canvas.buscarDireccion(parametro);
       break;
 case 2: //buscar area
       resultado = canvas.buscarArea(parametro);
       break;
 }
 return resultado; //devuelve false si no contiene el parametro
 }catch(Exception e){
  // System.err.println("enrutador error \n"+e);
 return true;
 }
 }

/**Cuando se configura una interfaz, es necesario que la dirección IP
 sea guardada en el área de Dibujo para prevenir que se configure otra
 interfaz con la misma dirección IP*/

 public void insertarenCanvas(int opcion,String parametro){
   switch(opcion){
   case 1://insertar direccion IP en canvas
         canvas.insertarDireccion(parametro);
         break;
   case 2: //insertar area
         canvas.insertarArea(parametro);
         break;
 }
 }
```

```
  /**Metodo utilizado para configurar las interfaces cuando se abre un
   * archivo*/
  public void cargarDatos(){
    /**Carga las interfaces como inactivas para empezar a correr el protocolo nuevamente*/
    for(int i=0;i<conexiones.size();i++){
      Vector temp = (Vector)conexiones.elementAt(i);
      temp.setElementAt("Inactiva",4);
      String red =(String)temp.elementAt(7);
      String mascara = (String)temp.elementAt(2);
      String inter = (String)temp.elementAt(3);
      String metrica = (String)temp.elementAt(6);
      String areaoAS = (String)temp.elementAt(5);
      String dirIP = (String)temp.elementAt(1);
      comandos.setInterfaz(inter,dirIP,mascara,red,metrica,areaoAS);
    }
  }

    /**Método llamado cuando se inicia la simulación, el cual coloca en
    * funcionamiento el protocolo que ha sido configurado por el usuario*/
  public void empezar(){
    empezar=true;
    for(int i=0;i<conexiones.size();i++){ //antes de empezar se deben agregar las interfaces
      corriendo=true;
      Vector temp = (Vector)conexiones.elementAt(i);
      String red =(String)temp.elementAt(7);
      String mascara = (String)temp.elementAt(2);
      String inter = (String)temp.elementAt(3);
      String metrica = (String)temp.elementAt(6);
      String areaoAS = (String)temp.elementAt(5);
      String dirIP = (String)temp.elementAt(1);
      Conexion c = (Conexion)temp.elementAt(0);
      c.setDesconectado(false,this.label()); //se quita el dibujo de la X
      proto.agregarInterfaz(red,mascara,inter,metrica,areaoAS,dirIP);//agrega la configuracion al
protocolo
      comandos.deshabilitarBotones(inter);//hay que deshabilitar los botones de las interfaces
    }
    proto.initProtocolo("todas");
  }

/**Detiene el protocolo que se esta ejecutando en ese momento*/
```

```
public void detener(){
 empezar=false;
 proto.stopProtocolo();
 comandos.habilitarBotones();
 switch (protocoloSelecc){
    case 1: proto = new Rip(this);
            if(tipoConfiguracion!=-1){
            proto.configurarProtocolo(tipoConfiguracion);
            }
            break;
    case 2: proto = new Ospf(this);
            break;
    case 3: proto = new Bgp(this);
            if(tipoConfiguracion!=-1){
            proto.configurarProtocolo(tipoConfiguracion);
            }
            break;
    case 4: proto = new Igrp(this);
            break;
}
}

 /**Devuelve las direcciones configuradas para las interfaces de este router
  las cuales serán utilizadas en el proceso de borrar un router de la pantalla*/

 public Vector getDirecciones(){
 Vector direcciones = new Vector();
 for(int i=0;i<conexiones.size();i++){ //antes de empezar se deben agregar las interfaces
   Vector temp = (Vector)conexiones.elementAt(i);
   String dirIP = (String)temp.elementAt(1);
   direcciones.addElement(dirIP);
 }
 return direcciones;
 }

 /**Método utilizado solo por OSPF, el cual devuelve el area que se debe borrar
  * en caso de que sea un router ABR*/
 public String getABR(){
 return comandos.getAreas();
 }
```

```
/**Cuando el usuario selecciona un protocolo, se llama a este método para
 * indicarle al enrutador que procolo debe colocar en funcionamiento cuando
 * se arranque la simulación.*/

  public void setProtocolo(int protocolo){
    protocoloSelecc=protocolo;
    switch (protocolo){
      case 1: proto = new Rip(this);
              comandos = new DialogoConfiguracion(inicio,"Dialogo de configuracion
"+this.label(),true,this,1);
              break;
       case 2: proto = new Ospf(this);
              comandos = new DialogoConfiguracion(inicio,"Dialogo de configuracion
"+this.label(),true,this,2);
              break;
       case 3: proto = new Bgp(this);
              comandos = new DialogoConfiguracion(inicio,"Dialogo de configuracion
"+this.label(),true,this,3);
              break;
       case 4: proto = new Igrp(this);
              comandos = new DialogoConfiguracion(inicio,"Dialogo de configuracion
"+this.label(),true,this,4);
              break;
    }
  }

  /**Se pasa TestCanvas porque se necesita para realizar comparaciones
   con las direcciones configuradas en los otros routers*/

  public void setCanvas(TestCanvas tc,Inicio inicio){
  canvas = tc;
  this.inicio=inicio;
  }


/**Recibe las opciones para configurar el protocolo*/

  public void configProtocolo(int tipoConfig){
    tipoConfiguracion=tipoConfig;
    proto.configurarProtocolo(tipoConfig);
```

```
 }

/**Recibe una serie de comandos o de indicaciones que le indican al enrutador
 los procesos que debe ejecutar*/

  public Vector recibirComandos(int operacion,String parametro){
   Vector resultado = new Vector();
   switch(operacion){
    case 1://activar interfaz
        if(empezar){ //si han dado clic en Inicio en el botón de empezar global
        boolean continuar=false;
        for(int i=0;i<conexiones.size();i++){ //antes de empezar se deben agregar las interfaces
          Vector temp = (Vector)conexiones.elementAt(i);
          String inter = (String)temp.elementAt(3);
          String estado = (String)temp.elementAt(4);
          if(inter.equals(parametro)&&estado.equals("Inactiva")){
            String red =(String)temp.elementAt(7);
            String mascara = (String)temp.elementAt(2);
            String metrica = (String)temp.elementAt(6);
            String areaoAS = (String)temp.elementAt(5);
            String dirIP = (String)temp.elementAt(1);
            Conexion c = (Conexion)temp.elementAt(0);
            c.setDesconectado(false,this.label()); //se quita el dibujo de la X
            proto.agregarInterfaz(red,mascara,inter,metrica,areaoAS,dirIP);
            continuar=true;
            corriendo=true;
            break;
          }
        }
        if(continuar){
          resultado.addElement("true");//se devuelve un Vector que tiene un string
          proto.initProtocolo(parametro); //con un valor = True para indicar que se ejecuto la
operacion
        }else{
          resultado.addElement("false");//no se completo la operacion
        }
        }else{
         resultado.addElement("false");//no se completo la operacion
        }
        break;
```

```
   case 2://detener interfaz
         if(proto.detenerInterfaz(parametro)){
           Conexion c=null;
           for(int i=0;i<vectorInterfaces.size();i++){
             Vector temp = (Vector)vectorInterfaces.elementAt(i);
             String nInterfaz = (String)temp.elementAt(0);
             if(parametro.equals(nInterfaz)){
               c = (Conexion)temp.elementAt(1);
               c.setDesconectado(true,this.label());
               break;
             }
           }
          resultado.addElement("true");// se completo la operacion
         }else{
          resultado.addElement("false");//no se completo la operacion
         }
         break;

    case 3://mostrar tabla de enrutamiento
          resultado = proto.imprimirTabla();
          break;

    case 4: // mostrar otros datos importantes del protocolo
          resultado=proto.imprimirDatos(parametro);
   }
   return resultado;
  }

/**Recupera la tabla de enrutamiento de el protocolo que se esta ejecutando en ese
   * instante
   * @return Tabla Vector que contiene la tabla de enrutamiento*/


   public Vector getTablaEnrutamiento(){
   try{
   Vector tabla = proto.imprimirTabla();//    proto.imprimirTabla();
   return tabla;
   }catch(Exception e){
   return null;
   }
```

```
 }

/**Método que recibe las tramas provenientes de la conexión
 * @param trama Trama que recibe el enrutador
 * @param interfaz Interfaz por la cual recibio la trama*/

public synchronized void recibirDatos(Object trama, int interfaz){
 Trama tramaProtocolo=(Trama)trama;
 if(empezar){
 String inter=null;
 String dirIpProtocolo=null;
 switch(interfaz){
    case 0: inter = "E0";
         dirIpProtocolo = getDireccion("E0");
         break;
    case 1: inter = "S0";
         dirIpProtocolo= getDireccion("S0");
         break;
    case 2: inter = "S1";
         dirIpProtocolo = getDireccion("S1");
         break;
    case 3:inter = "E1";
          dirIpProtocolo = getDireccion("E1");
          break;
 }
proto.tramaRecibida(tramaProtocolo,dirIpProtocolo,inter);
}
}

public String getDireccion(String interfaz){
String dirIP = "";
  for(int i=0;i<conexiones.size();i++){ //antes de empezar se deben agregar las interfaces
    Vector temp = (Vector)conexiones.elementAt(i);
    String inter = (String)temp.elementAt(3);
    if(inter.equals(interfaz)){
      dirIP = (String)temp.elementAt(1);
      break;
    }
  }
 return dirIP;
```

```
}

/**Método que envía una trama por la interfaz que tiene la dirección IP
 *que se recibe como parámetro
 * @param trama Trama que se va a enviar
 * @param dirIP Dirección IP de la interfaz por la que se va a enviar la trama*/

  public void enviarTrama(Trama trama, String dirIP){
     Iterator it = conexiones.iterator();
      while(it.hasNext()){
        Vector relacion = (Vector)it.next();
        Conexion con = (Conexion)relacion.elementAt(0);
         if(((String)relacion.elementAt(1)).equals(dirIP)){
           if(con.getNombreFigura1().equals(this.label())){
           con.setDatosFigura1(trama);
           }else{
           con.setDatosFigura2(trama);
           }
         break;
         }
     }
  }

  /**Método que envía una trama posion por la interfaz que recibe como parámetro
   * @param trama Trama que se va a enviar
   * @param nomInterfaz Interfaz por la que se va a enviar la trama*/

  public void enviarTramaPoison(Trama trama,String nomInterfaz){
    Iterator it = conexiones.iterator();
    while(it.hasNext()){
    Vector relacion = (Vector)it.next();
    Conexion con = (Conexion)relacion.elementAt(0);
    String dirInterfaz = (String)relacion.elementAt(1);
    String interfaz = (String)relacion.elementAt(3);
    if(interfaz.equals(nomInterfaz)){
      if(con.getNombreFigura1().equals(this.label())){
           trama.setDirecciones(dirInterfaz,"multicast");
           this.monitorear("se envia trama con horizonte dividido por la interfaz: "+nomInterfaz+
" Con las siguientes entradas");
           for(int ii=0;ii<trama.getNumEntradas();ii++){
```

```
            this.monitorear(trama.getEntradas(ii).toString());
            }
            con.setDatosFigura1(trama);
        }else{
             trama.setDirecciones(dirInterfaz,"multicast");
             this.monitorear("se envia trama con horizonte dividido por la interfaz: "+nomInterfaz+
" Con las siguientes entradas");
             for(int ii=0;ii<trama.getNumEntradas();ii++){
             this.monitorear(trama.getEntradas(ii).toString());
             }
            con.setDatosFigura2(trama);
           }
       break;
    } } }

public Vector getConexiones(){
  return conexiones;
}

/**Muestra el dialogo de Configuración del enrutador*/
public void showConsola(){
comandos.setBounds(40,40,470,236);
comandos.show();
}

/**Imprime en el dialogo de Configuración del enrutador los eventos
 que ocurren con el protocolo que se esta simulando*/

public void monitorear(String datos){
comandos.escribir(datos);
} }
```

4. Clase Figura

```
package interfaz;

import java.io.Serializable;
import java.awt.*;

public abstract class Figura implements Serializable {

        private Rectangle bounds;
        private String label;
        private transient boolean selected = false;

        public Figura(String label, int x, int y, int w, int h)        {
              bounds = new Rectangle(x, y, w, h);
               this.label = label;
        }

    /**Método invocado por paint() que maneja las normas para dibujar
     la figura. Llama a los métodos drawLabel() para mostar el nombre,
     drawImage() que pinta la imagen que representa la figura (un Host es
     representado por un PC) y si la figura ha sido seleccionada llama al método
drawSelectors()
     */
        public void draw(Graphics g)   {
              drawShape(g);
               Color c = g.getColor();
               g.setColor(Color.white);
               fillShape(g);
               g.setColor(Color.blue);
               drawLabel(g);
               g.setColor(c);
               drawImage(g);
               if(selected) drawSelectors(g);
        }

        public void drawOutline(Graphics g)
        {  drawShape(g);
```

```
        }

        public abstract void drawShape(Graphics g);

        public abstract void fillShape(Graphics g);

        /**Dibuja la imagen que representa a la figura, EJ: Para un Host, la imagen que
         representa la figura es un PC*/

        public abstract void drawImage(Graphics g);

        /**Método que retorna el tipo de Elemento, es decir si es un Router,
         un host o un hub
         @return tipoElemento*/

        public abstract String tipoElemento();
        public abstract void recibirDatos(Object trama,int interfaz);//  metodo utilzado para recibir
//los datos de la conexión

        /**Método que inicializa los procesos de la figura*/
        public abstract void empezar();

        /**Método que detiene los procesos de la figura*/
        public abstract void detener();

       /**Dibuja el nombre de la figura*/
        public void drawLabel(Graphics g)
        {  g.drawString(label, bounds.x + 5, bounds.y + bounds.height+15);
        }

       /**Verifica si la figura ha sido seleccionada
        * @return true La figura esta seleccionada
        * @return false La figura no esta seleccionada*/
        public boolean isSelected(){
        return selected;
        }

        /**Cuando el usuario da clic sobre el área de dibujo y este se encuentra
         en los límites de la figura se llama a este método para indicar que la
         figura ha sido seleccionada
```

```
 @param true La figura ha sido seleccionada*/

public void setSelected(boolean t){selected = t;}

public void toggleSelected(){selected = !selected;}

/**Cuando la figura ha sido seleccionada se debe dibujar unos pequeños
 cuadros alrededor de la misma. Este método es el encargado de esta operación*/
public void drawSelectors(Graphics g)
{  g.fillRect(bounds.x - 2 , bounds.y - 2, 4, 4);
       g.fillRect(bounds.x + bounds.width - 2 , bounds.y-2, 4, 4);
       g.fillRect(bounds.x-2 , bounds.y + bounds.height - 2, 4, 4);
       g.fillRect(bounds.x + bounds.width - 2 , bounds.y + bounds.height - 2, 4, 4);
}

public Rectangle rectangle(){ return bounds; }

public Point location(){ return new Point(bounds.x, bounds.y); }

public Point center(){
return new Point(bounds.x+bounds.width/2, bounds.y+bounds.height/2); }
public Dimension dimension(){ return new Dimension(bounds.width, bounds.height);}

public void move(int x, int y)
{  bounds.x = x; bounds.y = y;
}

public void move(Point p)
{  bounds.x = p.x; bounds.y = p.y;
}

/**Verifica si el clic dado por un usuario se encuentra dentro de los límites
 de la figura
 @param Point Especifica la posición donde el usuario pulso clic
 @return true Si el clic fue pulsado en los límites de la figura */

public boolean contains(Point p)
{  return bounds.contains(p);
}
```

```
        public boolean contains(int x, int y)
        {  return bounds.contains(x, y);
        }

        public Rectangle getBounds(){
          return bounds;
        }

        /**Método con el cual se obtiene el nombre de la figura
         * @return label El nombre de la figura*/
        public String label()
        {  return label;
        }

        public void cambiarLabel(String label){
          this.label = label;
        }
}
```

5. Clase Protocolo

```
package Protocolo;

import interfaz.Enrutador;
import java.util.*;
import java.awt.Color;

public abstract class protocolo{
private String nombreEnrutador;

  public protocolo(Enrutador router) {
  }
  /**Con este método se obtiene la tabla de enrutamiento generada por el
   protocolo que se está ejecutando
   @return Vector Contiene la tabla de enrutamiento*/

  public abstract Vector imprimirTabla();

  /**Se obtienen otras datos importantes del protocolo que se está ejecutando
   @param parametro Indica el tipo de información que se solicita al protocolo
   @return Vector Contiene la información solicitada
   */
  public abstract Vector imprimirDatos(String parametro);

  /**Cuando llega alguna trama al enrutador, este pasa la trama al protocolo para
   ser procesada
   @param trama La trama que ha sido recibida
   @param dirInterfaz La dirección IP de la interfaz por la cual se recibio la trma
   @param interfaz La interfaz por la cual se recibió la trama*/

  public abstract void tramaRecibida(Trama trama, String dirInterfaz, String interfaz);

  /**Método para iniciar el protocolo de enrutamniento*/
  public abstract void initProtocolo(String interfaz);

  /**Detiene el protocolo que se está ejecutando */
  public abstract void stopProtocolo();
```

```
  /**Detiene una interfaz específica del enrutador
   @param interfaz Nombre de la interfaz que se va a detener*/
  public abstract boolean detenerInterfaz(String interfaz);

  /**Cuando se configuran las interfaces, se deben pasar los datos al
   protocolo para que este las guarde en su tabla de enrutamiento*/
  public abstract void agregarInterfaz(String red,String mascara,String interfaz,String
   metrica,String areaoAS,String dirInterfaz);

  /**Método que configura algunas opciones especiales que tienen los protocolos
   que se están simulando*/
  public abstract void configurarProtocolo(int tipoConfiguracion);
}
```