Proyecto de fin de carrera [1]

EasySim Revolution

Documentación del programa

Juan Pablo Valentín jpvalentin@gmail.com
Alejandro Selios aselios@gmail.com
Leonardo Alves lalves@gmail.com

año 2006

Tutores :
Vítor González Barbone
Pablo Belzarena

[1]Instituto de Ingeniería Eléctrica. Facultad de Ingeniería - Uruguay

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Capítulo 2

Descripción del simulador

(análisis y diseño general del simulador)

Capítulo 3

Core

Los elementos que puede contener una red a ser simulada en EasySim Revolution (ESR) son: hosts, routers y links.

3.1. Host

3.1.1. Análisis de requerimientos

Los Hosts están compuestos por:

- agentes de tráfico.
- generadores de tráfico.
- un clasificador de paquetes.
- una interfaz.

Agentes de tráfico Son los encargados de controlar tanto el tráfico proveniente de los generadores, como el entrante al Host. En ésta versión de ESR el único agente existente es UDP.

Generadores de tráfico Cada agente de tráfico puede recibir los paquetes generados por uno o varios generadores de tráfico. En cada generador se puede elegir:

- que los tiempos inter paquetes sigan una distribución o que sea determinístico.

- que los tamaños de los paquetes sigan una distribución o que sea determinístico.
- destino del paquete.
- clase de tráfico del paquete.
- prioridad al descarte del paquete.
- tiempo de prendido y de apagado del generador.

Clasificador Es el elemento del host que conecta a los agentes de tráfico con la interfaz del host. Su función principal es la de enviar al agente debido los paquetes provenientes de la interfaz.

Interfaz Cumple dos funciones:

- recibe los paquetes provenientes del exterior del Host. Estos son recibidos y enviados directamente al clasificador del host.
- recibe los paquetes provenientes del clasificador del host. Según la política de descarte elegida para la cola de la interfaz, los paquetes son encolados y luego despachados siguiendo la disciplina de despache elegida para el despachador de la misma.

Cada interfaz tiene una cola física, la cual está dividida en n colas lógicas. A cada clase de tráfico, le corresponde una cola lógica.

Las políticas de descarte existentes en ésta versión del simulador son:

- WRED
- Drop Tail

Las disciplinas de despache disponibles son:

- Fifo
- Lifo
- Prioridad Estricta Fifo
- Prioridad Estricta Lifo
- DWRR

Para que la interfaz tenga un comportamiento más realista, se le puede configurar una probabilidad de caída. De ésta manera, durante la simulación, la interfaz puede caerse o levantarse con una probabilidad establecida por el usuario.

Casos de uso

El generador crea un paquete de datos.

Caso de uso: El generador crea un paquete de datos.

Actor: Generador.

Tabla 3.1: Caso de uso El generador crea un paquete de datos.

Acción del actor	**Acción del sistema**
1. El generador crea un paquete de tamaño, destino, clase, prioridad al descarte específicos. 2. El paquete es enviado al agente del host.	
	3. El agente recibe el paquete, lo trata según su política y lo envía al clasificador.
	4. El clasificador del host recibe el paquete y lo envía a la interfaz.
	5. Cuando el paquete llega a la interfaz, es encolado según la política de descarte de la cola.
	6. En caso que la cola haya estado vacía, la interfaz le avisa al despachador que hay paquetes para despachar.
	7. El despachador despacha el paquete según la disciplina de despache elegida.

La interfaz recibe un paquete del exterior del host.

Caso de uso: La interfaz recibe un paquete del exterior del host.

Actor: Interfaz del host.

Tabla 3.2: Caso de uso La interfaz recibe un paquete del exterior del host.

Acción del actor	**Acción del sistema**

1. La interfaz recibe un paquete del exterior del host y lo envía al clasificador.	2. El clasificador del host recibe el paquete y lo envía al agente que corresponda. 5. El agente recibe el paquete y realiza las acciones debidas.

Modelo de dominio

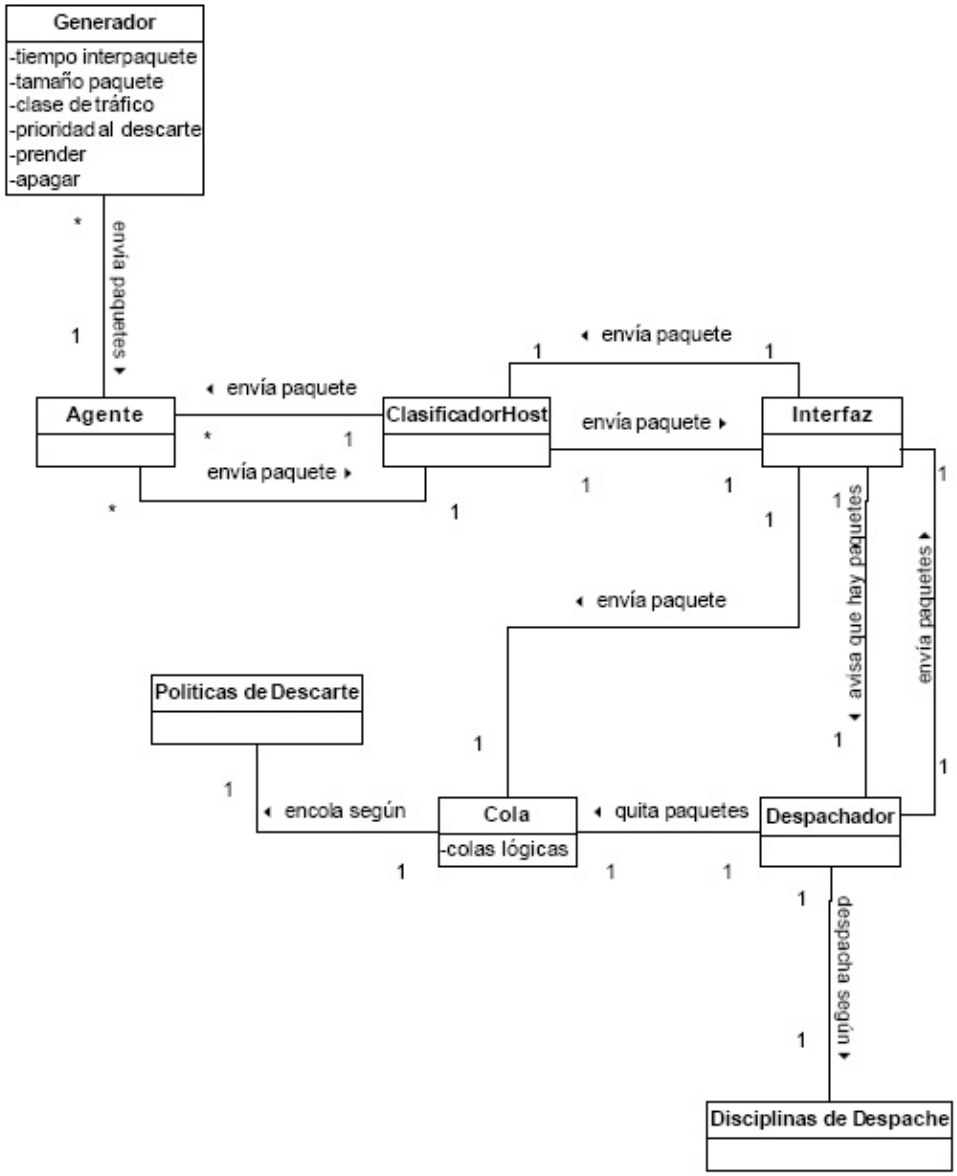


Figura 3.1: Modelo de dominio del Host

3.1.2. Diseño

Diagrama de Clases

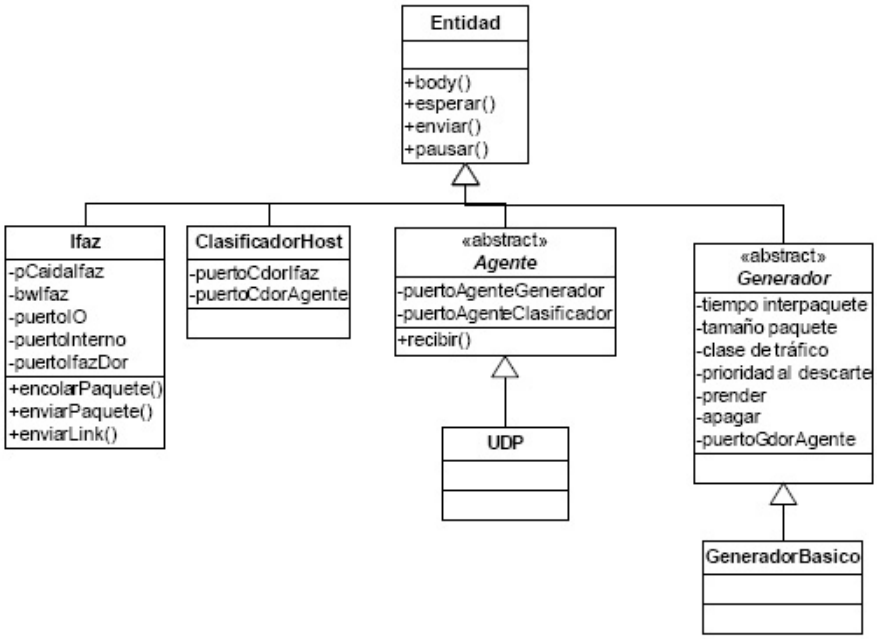


Figura 3.2: Diagrama de Clases de Diseño de las entidades del Host

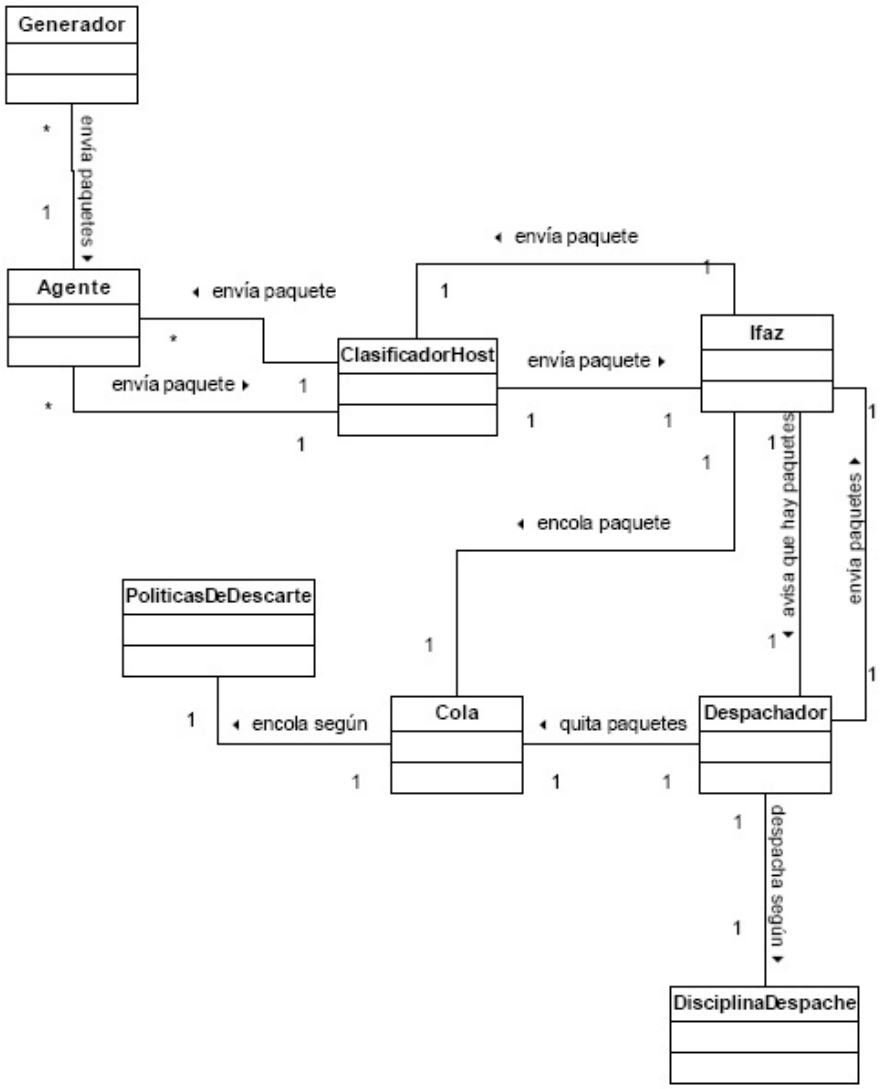


Figura 3.3: Diagrama de Clases de Diseño del Host

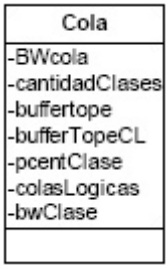


Figura 3.4: Diagrama de Clases de Diseño de las colas de las interfaces

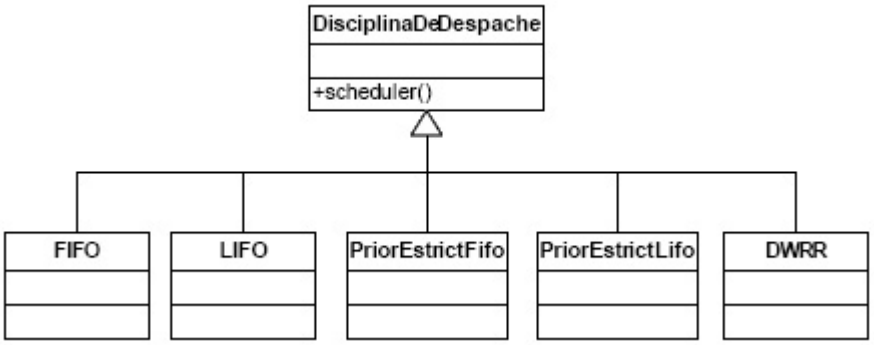


Figura 3.5: Diagrama de Clases de Diseño de las disciplinas de despache

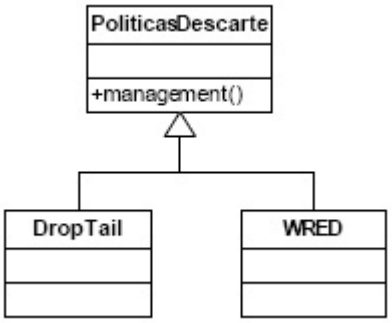


Figura 3.6: Diagrama de Clases de Diseño de las políticas de descarte

La comunicación entre las distintas Entidades que conforman un host se realiza a través de los distintos puertos de las mismas. La imagen de la figura 3.7 muestra la conexión de los distintos puertos.

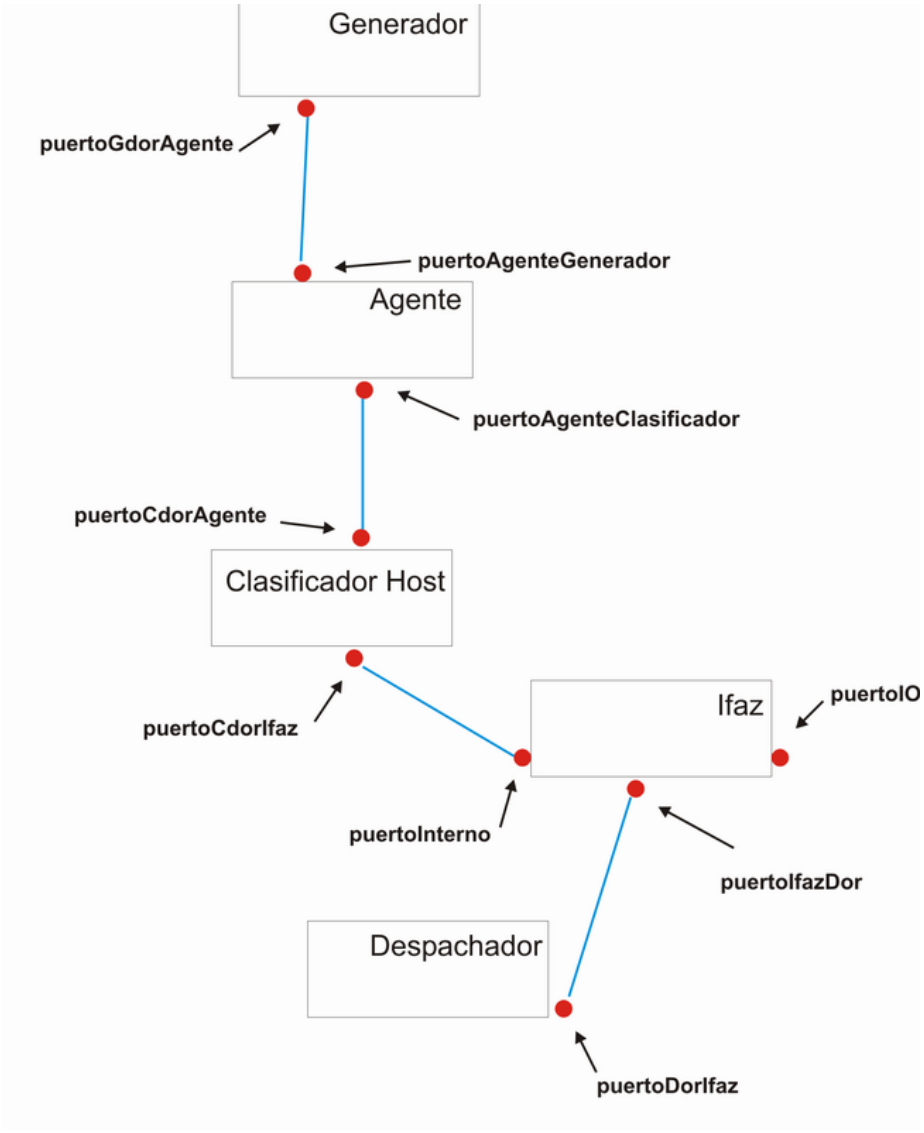


Figura 3.7: Conexión de los puertos de las entidades de los host

Diagramas de Colaboración

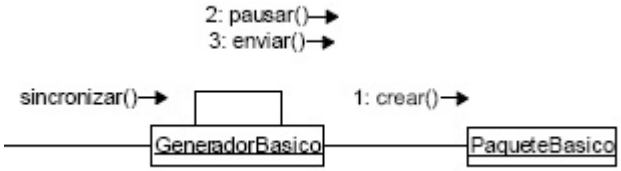


Figura 3.8: Diagrama de colaboración para la generación de paquetes

Si el agente recibe un paquete del puerto `puertoAgenteGenerador`, lo envía al clasificador del host por medio del puerto `puertoAgenteClasificador`.

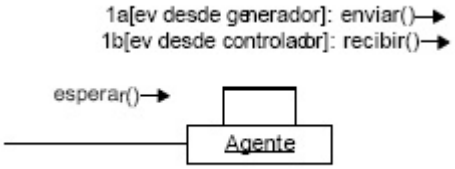


Figura 3.9: Diagrama de colaboración para los agentes de tráfico

El clasificador del host, espera paquetes de los dos puertos que tiene. En caso que el paquete ingrese por el puerto `puertoCdorIfaz`, se chequea a que agente pertenece el paquete y se envía al puerto del agente correspondiente. En caso que el paquete provenga de un agente, se envía a la interfaz por el puerto `puertoCdroIfaz`.

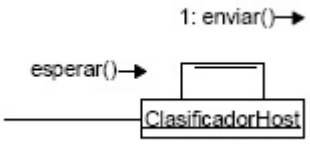


Figura 3.10: Diagrama de colaboración para el clasificador del Host

La interfaz espera los paquetes provenientes de los distintos puertos. En caso que el paquete ingrese por el puerto `puertoIO`, la interfaz lo envía al puerto `puertoInterno`. Si viene del `puertoInterno` el paquete es encolado y si viene del puerto `puertoIfazDor`, se envía al link.

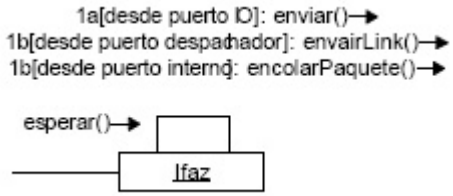


Figura 3.11: Diagrama de colaboración para la interfaz del Host

3.1.3. Implementación

Como puede verse en los diagramas de clases, los hosts están compuestos básicamente por entidades que interactúan entre sí. Cada entidad tiene definido su comportamiento en el método `body()` de la clase que extiende de `Entidad`[1]

Lo que hacen es esperar la llegada de paquetes a los distintos puertos para enviarlos a los puertos que correspondan una vez se les haya hecho el tratamiento específico.

Existe una excepción en la clase que representa las interfaces `Ifaz`, ya que cuando el método `body()` se corre por primera vez, se recorre un ArrayList `acciones`. Cada elemento del mismo es un `ArrayList` que contiene un `int` que representa una acción y un `double` que representa el tiempo en que se debe cumplir dicha acción. A medida que se recorre `acciones` se va agendando la acción y el tiempo en que debe ejecutarse.

Una vez agendados todos los eventos, se comienzan a esperar los distintos eventos. Si llega un evento que contiene un paquete, se envía por el puerto debido. En caso que llegue un evento que represente una de las acciones agendadas al inicio del `body()`, se ejecuta el método que representa dicha acción.

A continuación se muestra parte del método `body()`. Se puede identificar en que lugar se agendan las acciones y luego se ejecutan.

```
public static final int TIRAR_IFAZ = 60;
public static final int LEVANTAR_IFAZ = 61;
public static final int TIRAR_LINK = 70;
public static final int LEVANTAR_LINK = 71;

public void body() {

//se agendan todos los eventos a ser realizados por la interfaz y el  link
  if (acciones!=null){
    Iterator iter=acciones.iterator();
    while (iter.hasNext()) {
        ArrayList tiempoAccion = (ArrayList) iter.next();
```

[1] ver capitulo

```
        double tiempo=((Double)tiempoAccion.get(0)).doubleValue();
        int tag=((Integer)tiempoAccion.get(1)).intValue();
        enviar(this.get_id(),tiempo,tag);
     }
  }
  while (Sistema.isRunning()) {
  //se esperan eventos.
  //pueden venir paquetes del puerto que se comunica con el exterior,
  // y se envia al puerto interno
  //                      del puerto interno y se encola
  //                      del puerto de comunicacion con el despachador, se envia
  //al seudoLink
  //tambien pueden venir eventos programados por el usuario.
  //                      tirar o levantar ifaz
  //                      tirar o levantar seudoLink
  Evento ev = new Evento();
  this.sim_get_next(ev);
  if (ev.getDato()!=null) {
  //se chequea que el dato sea un paquete
  //se toman acciones en caso que la interfaz no
  //este caida
    if (ev.get_data() instanceof Paquetes) {
         Paquetes paquete = (Paquetes)ev.getDato();
   ..............................................
    }else{// en caso que no sea un paquete puede ser un evento
    switch (ev.get_tag()){
        case 60: this.tirarIfaz(); break;
        case 61: this.levantarIfaz(); break;
        case 70: this.tirarLink(); break;
        case 71: this.levantarLink(); break;
    }
   }
  }
}
```

Las acciones que se pueden ejecutar en ésta versión de ESR son:

- tirar un link.
- levantar un link.
- tirar una interfaz.
- levantar una interfaz.

En caso que se deseen agregar nuevas acciones, solo basta con determinar un número entero que identifique la acción, agregar el método que la ejecute y agregar lo necesario en el bloque *switch* del *body*().

Como se puede deducir de las secciones anteriores, cada interfaz está formada por una cola física la cual contiene a su vez n colas lógicas. Según sea la política de descarte y la disciplina de despache elegida para la interfaz, será como se encolen y se despachen los paquetes.

Las políticas de descarte existentes en ESR son Drop Tail y WRED. En caso que se desee agregar una nueva política, basta con crear una clase que herede de `PoliticaDeDescarte` y que implemente el método `management(PaquetesDatos paquete,Cola cola)`. El parámetro de entrada `paquete` es el paquete que se desea encolar en la cola `cola`.

Al igual que en las políticas de descarte, nuevas disciplinas de despache pueden ser agregadas con solo crear una clase que herede de `DiscipinaDeDespache` y que implemente el método `scheduler(double time, Cola cola)`, donde `time` es el tiempo en que se despacha un paquete de la cola `cola` siguiendo una disciplina definida en el método `scheduler()`.

3.2. Router

3.2.1. Análisis de requerimientos

Los routers están compuestos por:

- interfaces.
- módulos de control.
- módulo Ruteo de Capa de Red.
- módulo Unidad de Control.
- módulo Recursos.
- un clasificador de paquetes.
- un clasificador de paquetes de control.

Interfaces Los routers pueden tener más de una interfaz. Al igual que la de los hosts, están compuestas por una cola física con n colas lógicas en la cual los paquetes son encolados según una política de descarte determinada y un despachador que quita los paquetes de la cola siguiendo una disciplina de despache.

Tiene dos funciones principales:

- enviar los paquetes que ingresan al router al clasificador de paquetes.

- recibir paquetes de los distintos módulos del router. Los paquetes pueden ser de control o de datos. En caso que se trate de un paquete de datos, se intenta encolar. Si el paquete es de control, se envía directamente al link sin pasar por la cola de la interfaz.

Clasificador de paquetes La función del clasificador de paquetes es la de recibir los paquetes enviados desde la interfaz, identificar si se tratan de paquetes de datos o de control y enviarlos al módulo que realice el ruteo de los paquetes de datos ó al clasificador de control respectivamente. En caso de que este activo el módulo MPLS se pasa el paquete a él, si no se pasa el paquete al módulo de Ruteo de Capa de Red.

Clasificador de paquetes de control Los paquetes de control que recibe éste clasificador son enviados al módulo de control adecuado.

Módulos de control Cada router puede tener distintos módulos que se encarguen del enrutamiento de los paquetes, descubrimiento y actualización de la información de la red, etc.[2].

Módulo Ruteo de Capa de Red Éste módulo siempre está presente en los routers. Es el encargado de dirigir los paquetes de datos provenientes del clasificador de paquetes cuando no se ha establecido el envío MPLS. La forma de enrutar los paquetes de éste módulo está basada en el ruteo IP.

Se busca el destino del paquete en una tabla de ruteo para saber por que interfaz se debe enviar el paquete.

Módulo Unidad de Control Es el módulo encargado de avisar a los distintos módulos de control, que deben ejecutar las acciones debidamente agendadas.

Módulo Recursos Es el módulo del router que contiene toda la información que el router conoce de la red a la que pertenece.

Casos de uso

Una interfaz recibe un paquete del exterior.

Caso de uso: Una interfaz recibe un paquete del exterior.

Actor: Una de las interfaces del router.

[2]Por más información consultar la sección 4

Tabla 3.3: Caso de uso Una interfaz recibe un paquete del exterior.

Acción del actor	**Acción del sistema**
1. Una de las interfaces del router recibe un paquete del exterior. Lo envía al clasificador de paquetes del router.	
	2. El clasificador de paquetes recibe el paquete y chequea si es un paquete de control o de datos. Si es un paquete de control, se envía al clasificador de control. Si el paquete es de datos y si se configuró el envío MPLS se envía al módulo MPLS para que sea correctamente tratado. Si es de datos pero no existe envío MPLS, el paquete es enviado al módulo Ruteo Capa de Red.

El clasificador de control recibe un paquete.

Caso de uso: El clasificador de control recibe un paquete.

Actor: Clasificador de datos.

Tabla 3.4: Caso de uso El clasificador de control recibe un paquete.

Acción del actor	**Acción del sistema**
1. Envía un paquete de control al clasificador de control.	
	2. El clasificador de control recibe el paquete y chequea a que módulo debe dirigir el paquete para luego enviarlo.

El módulo Ruteo de Capa de Red recibe un paquete.

Caso de uso: El módulo Ruteo de Capa de Red recibe un paquete.

Actor: Clasificador de datos.

Tabla 3.5: Caso de uso El módulo Ruteo de Capa de Red recibe un paquete.

Acción del actor	Acción del sistema
1. Envía un paquete de datos al módulo de Ruteo de Capa de Red.	2. El módulo de Ruteo de capa de red, chequea el destino del paquete recibido. 3. Según sea el destino, se obtiene de la tabla de ruteo la interfaz por la que se debe enviar el paquete. 4. Se envía el paquete a la interfaz obtenida.

La interfaz recibe un paquete de control del interior del router.

Caso de uso: La interfaz recibe un paquete de control del interior del router.

Actor: Módulo del router.

Tabla 3.6: Caso de uso La interfaz recibe un paquete de control del interior del router.

Acción del actor	Acción del sistema
1. Envía un paquete de control a la interfaz debida.	2. Cuando la interfaz recibe el paquete y verifica que es un paquete de control, se envía directamente al link sin ser encolado.

La interfaz recibe un paquete de datos del interior del router.

Caso de uso: La interfaz recibe un paquete de datos del interior del router.

Actor: Módulo Ruteo de Capa de Red.

Tabla 3.7: Caso de uso La interfaz recibe un paquete de datos del interior del router.

Acción del actor	Acción del sistema

1. Envía un paquete de datos a la interfaz debida.	2. Cuando la interfaz recibe el paquete y verifica que es un paquete de datos, se trata de encolar en la cola lógica que le corresponda a la clase de tráfico del paquete. El que el paquete sea encolado o no depende de la prioridad al descarte del paquete, el estado de la cola lógica y la política de descarte de la cola.

Despache de paquetes.

Caso de uso: Despache de paquetes.

Actor: Despachador.

Tabla 3.8: Caso de uso Despache de paquetes.

Acción del actor	**Acción del sistema**
1. Según el estado de la cola y la disciplina de despache del despachador de la interfaz, se van sacando los paquetes de la cola para ser enviados a la interfaz.	2. Cuando la interfaz recibe un paquete proveniente del despachador, lo envía al link.

Modelo de dominio

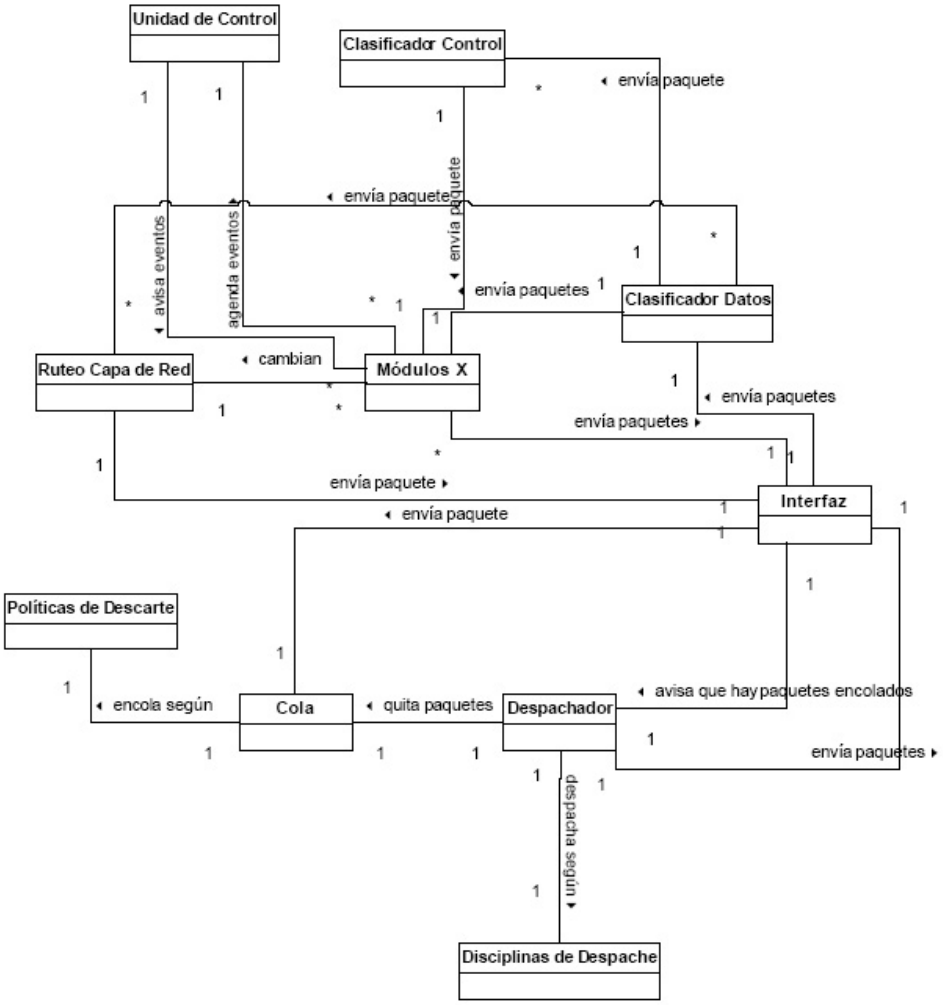


Figura 3.12: Modelo de dominio del Router

3.2.2. Diseño

Diagrama de Clases

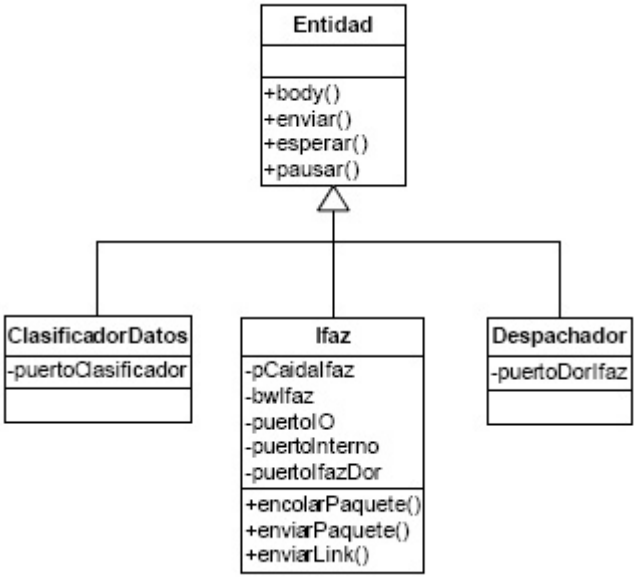


Figura 3.13: Diagrama de Clases de Diseño de las entidades del Router

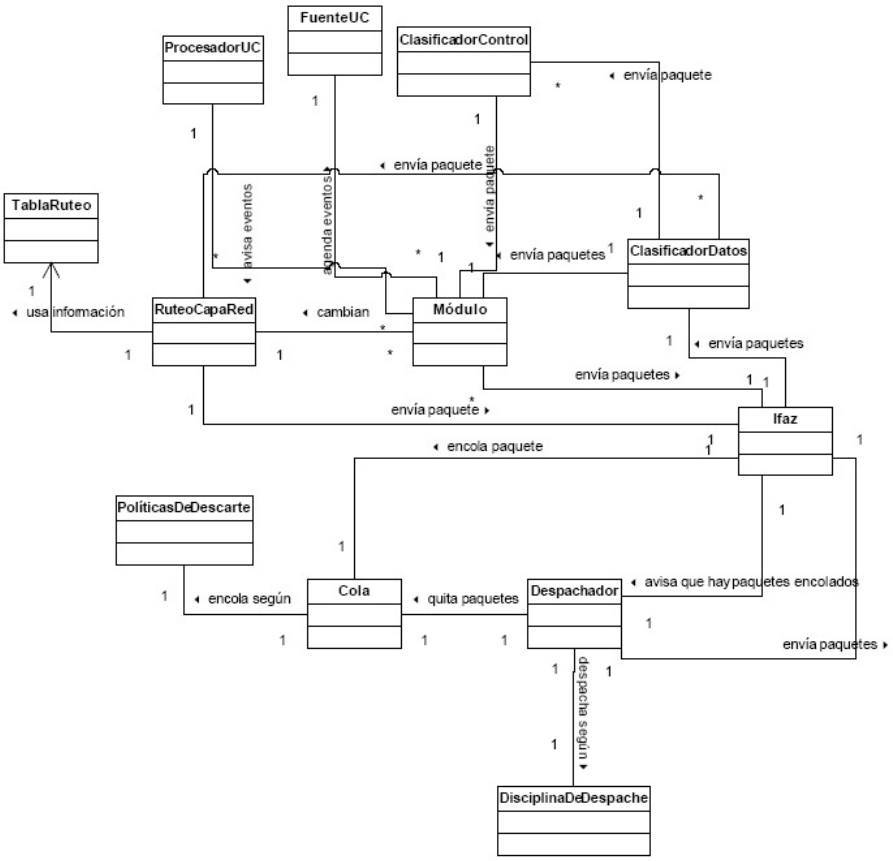


Figura 3.14: Diagrama de Clases de Diseño del Router

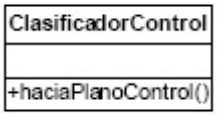


Figura 3.15: Diagrama de Clases de Diseño del Clasificador de Control

Los diagramas de clases de las disciplinas de despache, políticas de descarte , cola e interfaz son iguales que en los Host.

Diagramas de Colaboración

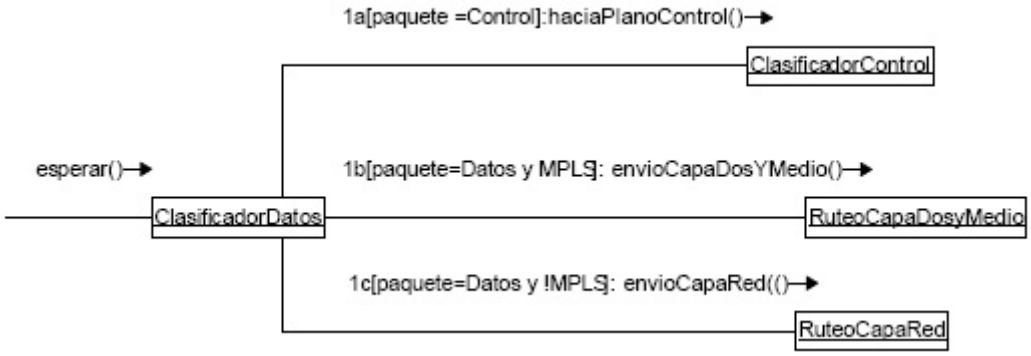


Figura 3.16: Diagrama de colaboración para la clasificación de paquetes.

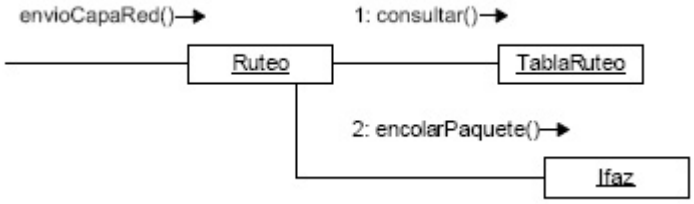


Figura 3.17: Diagrama de colaboración para Ruteo de Capa de Red.

La interfaz espera los paquetes provenientes de los distintos puertos. En caso que el paquete ingrese por el puerto `puertoIO`, la interfaz lo envía al puerto `puertoInterno`. Si viene del puerto interno el paquete es encolado y si viene del puerto `puertoIfazDor`, se envía al link.

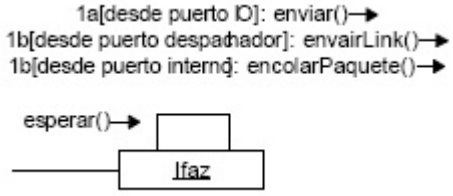


Figura 3.18: Diagrama de colaboración para la interfaz del Router

3.2.3. Implementación

3.3. Link

3.3.1. Análisis de requerimientos

Los links son los elementos de la red encargados de unir los nodos de la red. Para modelar los enlaces reales, los enlaces a simular tienen:

- ancho de banda finito.
- probabilidad de caída.
- probabilidad de pérdidad de un paquete.
- retardo.

Casos de uso

Un extremo del link recibe un paquete..

Caso de uso: Un extremo del link recibe un paquete.

Actor: Host o router .

Tabla 3.9: Caso de uso Un extremo del link recibe un paquete.

Acción del actor	**Acción del sistema**
1. Una interfaz de un router o host envía al link un paquete..	2. En caso que el link no esté caído, el paquete es recibido en un extremo y demora en llegar al otro extremo un tiempo determinado por el tamaño del paquete, el ancho de banda del enlace y el retardo del enlace.

Modelo de dominio

3.3.2. Diseño

Por la simplicidad de los links, se decidió agregar en la clase Ifaz un método que simula el comportamiento de los mismos.

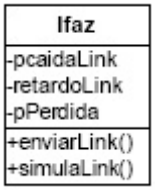


Figura 3.19: Diagrama de Clases de Diseño del Link

Diagramas de Colaboración

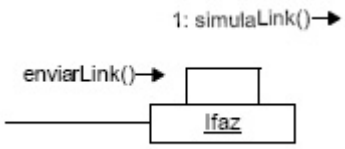


Figura 3.20: Diagrama de colaboración del Link

Capítulo 4

Módulos

En este capítulo se describen los diferentes módulos que están implementados en el simulador. Según sean los módulos que se le configure a cada nodo, será la funcionalidad que tenga. La idea de los módulos es que cada uno implemente alguna función totalmente diferenciada de los demás y que interactuando entre ellos se logre dotar al nodo de funcionalidades que por si solo cada módulo no lo lograría. Un ejemplo claro es la interacción entre los módulos de Rutas y Descubrimiento, en el caso de exista solo el módulo de Rutas, este podrá realizar el cálculo de rutas pero solo de forma estática, en cambio con la colaboración del módulo de Descubrimiento, el módulo de Rutas puede se puede usar para implementar ruteo dinámico.

4.1. MPLS

4.1.1. Análisis de requerimientos

Multi Protocol Label Switching (MPLS), es un protocolo que realiza la función forwarding, es decir de envío de los paquetes a través de la red. Lo hace empleando etiquetas. Para realizar dicha función utiliza varías tablas, mediante las cuales dado un paquete, MPLS determina las operaciones a realizar sobre el mismo para que sea enviado a su destino.

Para el envío de los paquetes emplea túneles llamados Label Switch Path (LSP) que unen el punto de ingreso con el punto de salida del dominio MPLS. El punto de ingreso y de egreso se denominan Label Edge Router (LER) de ingreso y egreso respectivamente. Los nodos internos se denominan Label Switch Router (LSR). Las operaciones que ejecutan los nodos sobre los paquetes son, label Push, label Pop y label Swap.

Los párrafos anteriores realizan una breve introducción a los términos manejados a lo largo de este documento, pero por más detalle consular **??**. El módulo

MPLS implementa las funciones básicas de MPLS, que se describen a lo largo de esta sección.

Para que MPLS pueda encaminar paquetes a través de LSP, se deben mapear los paquetes sobre los LSP. Dicha función la realiza el módulo de FEC (ver 4.2).

Cada LSP tiene un parámetro Forwarding Equivalent Class (FEC) que es utilizado para realizar la correspondiente asociación entre ambos. Cuando se crea el LSP el sistema realiza la asociación a la FEC, es decir a cada FEC le asigna la Next Hop Label Forwarding Entry (NHLFE). El conjunto de éstas asociaciones se agrupan en la tabla FEC to NHLF (FTN) (ver figura 4.1).

FEC	NHLFE
8	(3)
10	(1)
11	(5) (6)
20	(4) (8) (7)

Figura 4.1: Asociación de la FEC con el LSP. Para cada FEC puede existir más de una etiqueta (más de un LSP), permitiendo realizar balanceo de carga. A la tabla anterior se le denomina FTN

Hasta ahora solo se mencionó como se asocian las FEC. Ahora es momento de presentar las tablas que permitirán realizar el ruteo MPLS, explicando su funcionamiento. En la figura 4.1 , se presenta la primera tabla, la cual es usada para establecer la asociación entre la FEC y la entrada de la tabla Next Hop Label Forwarding (NHLF) (ver 4.2).

indice	NH	Operación	Etiqueta	Interfaz
(1)	LSR1	Label Push	10	i1
(2)	LSR2	Label Push	[13, 25, 44]	i3
(3)	LER2	Label Swap	15	i1
(4)	LER4	Label Pop	22	--
(5)	LSR6	Label Push	30	i2

Figura 4.2: Tabla NHLF. Indica que operaciones se deben realizar en el envío MPLS.

La tabla NHLF contiene toda la información referente a que operación debe realizar el módulo de MPLS para encaminar los paquetes a su próximo salto. El primer campo de esta tabla es un índice, llamado NHLFE y es usado para identificar cada entrada de la misma, del cual se hace referencia desde otras tablas (como ser la tabla FTN). Los restantes campos se describen en el siguiente esquema:

NH(Next Hop): Indica el nombre del nodo del próximo salto. Hasta ahora no se piensa que sea útil para realizar alguna operación solo, está a modo informativo.

Operación: Establece la operación que se debe realizar sobre la etiqueta del paquete. Pueden ser:

- **Label Push:** coloca una o varias etiquetas al paquete, dependiendo del campo etiqueta.
- **Label Swap:** realiza el intercambio de etiquetas. Es decir retira la del paquete y le coloca una nueva.
- **Label Pop:** retira la etiqueta del nivel superior del paquete.

Etiqueta: Nueva etiqueta que se le colocará al paquete, según la operación. Este campo puede contener una o un array de etiquetas. Un array de etiquetas solo puede estar asociado a una operación de Label Push, donde se van apilando las etiquetas de derecha a izquierda en un Stack de etiquetas. La función Label Pop siempre retira la última en ser colocada, es decir la del nivel superior.

Interfaz: Indica la interfaz por la cual se debe de enviar el paquete. Puede que no se especifique la interfaz, cuando la operación asociada es un Label Pop y esta sea la última etiqueta que exista en el paquete, es decir la del nivel 1. En este caso el paquete es encaminado por el protocolo de ruteo de Capa de Red.

La última de las tablas MPLS que resta mencionar es la Incoming Label Map (ILM). La tabla ILM (ver 4.3) asocia al paquete etiquetado que llega a un nodo de la red la entrada correspondiente en la NHLF para determinar su próxima operación.

Etiqueta	indice
12	(1)
1	(2)
5	(3)
2	(4)
8	(5)

Figura 4.3: Tabla ILM. Asocia al paquete etiquetado un entrada a la NHLF.

Cuando un paquete arriba a un LER, se clasifica. La primera distinción que se realiza es diferenciar entre un paquete de datos o de control. Si se trata de un paquete de control se pasa al plano de control enviándolo al Clasificador del plano de control quién lo deriva al módulo de control correspondiente.

Si es un paquete de datos, pueden ocurrir dos cosas. Una es que se encuentre activo el módulo MPLS(o dicho de otra forma que el router posea un módulo de MPLS), y la otra que no lo esté. Cuando se encuentra activo el módulo MPLS, el Clasificador de datos lo pasa al módulo de MPLS quien consulta al módulo de FEC para determinar si el paquete tiene una asociada. En caso afirmativo, este devuelve la FEC a la cual pertenece el paquete. Luego se consulta la tabla FTN, con la FEC obtenida para obtener la NHLFE. Con la NHLFE, él módulo MPLS consulta la tabla NHLF y obtiene la información necesaria para encaminar el paquete hacia su próximo salto. En caso de que el paquete no pertenezca a ninguna de las FECs, este se debe de enrutar según el protocolo de capa de red.

Por lo general cuando un paquete arriba a un LER y el mismo pertenece a alguna de las FEC activas (ver 4.2), la primer operación sobre el paquete implica la colocación de una etiqueta, es decir un label push. Por lo tanto, cuando el paquete arriba al próximo nodo, el módulo MPLS consulta la tabla ILM usando la etiqueta del paquete para determinar la NHLFE.

En el caso de que él módulo MPLS no se encuentre activo, el Clasificador del plano de datos deriva el paquete al módulo de Ruteo de Capa de Red.

Casos de uso

Ruteo de los paquetes de datos

Caso de uso: Ruteo de los paquetes de datos.

Actor: Módulo MPLS, Módulo ruteo de capa de red.

Tabla 4.1: Caso de uso, ruteo de los paquetes de datos.

Acción del actor	**Acción del sistema**
	1. Arriba un paquete que no pertenece a ninguna FEC definida a un LER de ingreso.
	2. La interfaz pasa el paquete al Clasificador del plano de datos, el cual se da cuenta de que se trata de un paquete de datos, por lo cual pasa el paquete al Módulo MPLS.
3. El módulo MPLS, consulta al Módulo de FEC, para determinar si el paquete pertenece a alguna FEC definida.	
	4. El módulo de FEC determina que el paquete no pertenece a ninguna FEC consultando la tabla de definición de FECs.
5. El módulo MPLS pasa el paquete al módulo de Ruteo de Capa de Red para que el mismo se ruteado por el protocolo de dicha capa.	
6. El módulo de Ruteo de Capa de Red, consulta la tabla de ruteo con el destino del paquete y así obtiene la interfaz de salida.	
7. El módulo de Ruteo de Capa de Red envía el paquete a la correspondiente interfaz.	
	8. El paquete es enviado hacia el próximo salto por la interfaz.
	9. El paquete arriba a un LSR.
	Se repiten los pasos del 2 al 8 para cada nodo de la red, incluyendo el LER de egreso.
	10. Arriba un paquete al LER de ingreso que pertenece a una FEC.

	11. La interfaz pasa el paquete al Clasificador del plano de datos, el cual se da cuenta de que se trata de un paquete de datos, por lo cual pasa el paquete al Módulo MPLS.
12. El módulo MPLS, consulta al Módulo de FEC, para determinar si el paquete pertenece a alguna FEC definida.	
	13. El módulo de FEC determina la FEC al cual pertenece el paquete consultando la tabla de definición de FECs y retorna la FEC encontrada.
14. El módulo MPLS consulta la tabla FTN, para determinar la NHLFE.	
15. Consulta la tabla NHLF a partir de la NHLFE obtenida. De esta manera obtiene la operación a aplicar sobre el paquete y también obtiene la interfaz de salida.	
16. Ejecuta la operación y luego envía el paquete por la interfaz correspondiente.	
	17. El paquete etiquetado arriba a un nodo de la red, suponiendo que en el paso anterior la operación fue un label push.
	18. El Clasificador de datos pasa el paquete módulo MPLS.
19. El módulo MPLS obtiene la etiqueta del paquete, con la cual consulta la tabla ILM, para obtener la NHLFE. Aquí se repiten los pasos 15 y 16.	
	2a.11a. En caso de que el Módulo MPLS no esté activo, el Clasificador de datos directamente pasa el paquete al módulo de ruteo de capa de red.

Diagramas de dominio

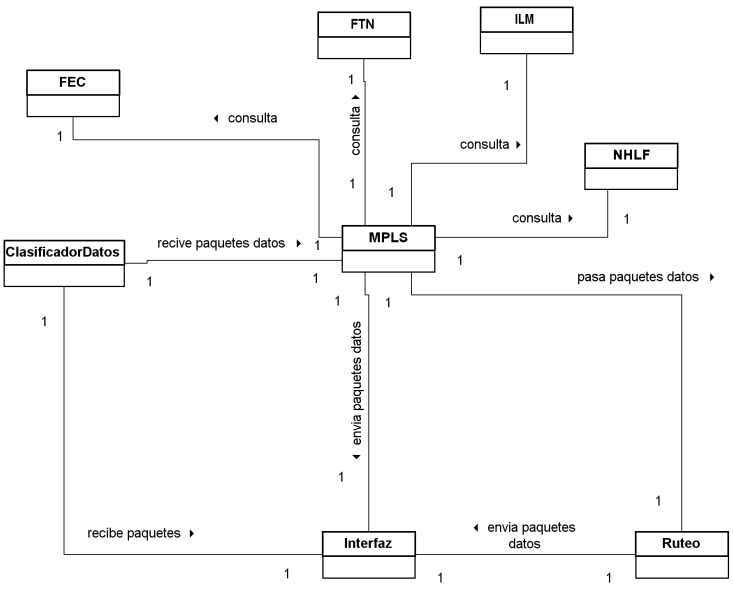


Figura 4.4: Diagrama de dominio del módulo MPLS

4.1.2. Diseño

Diagrama de Clases

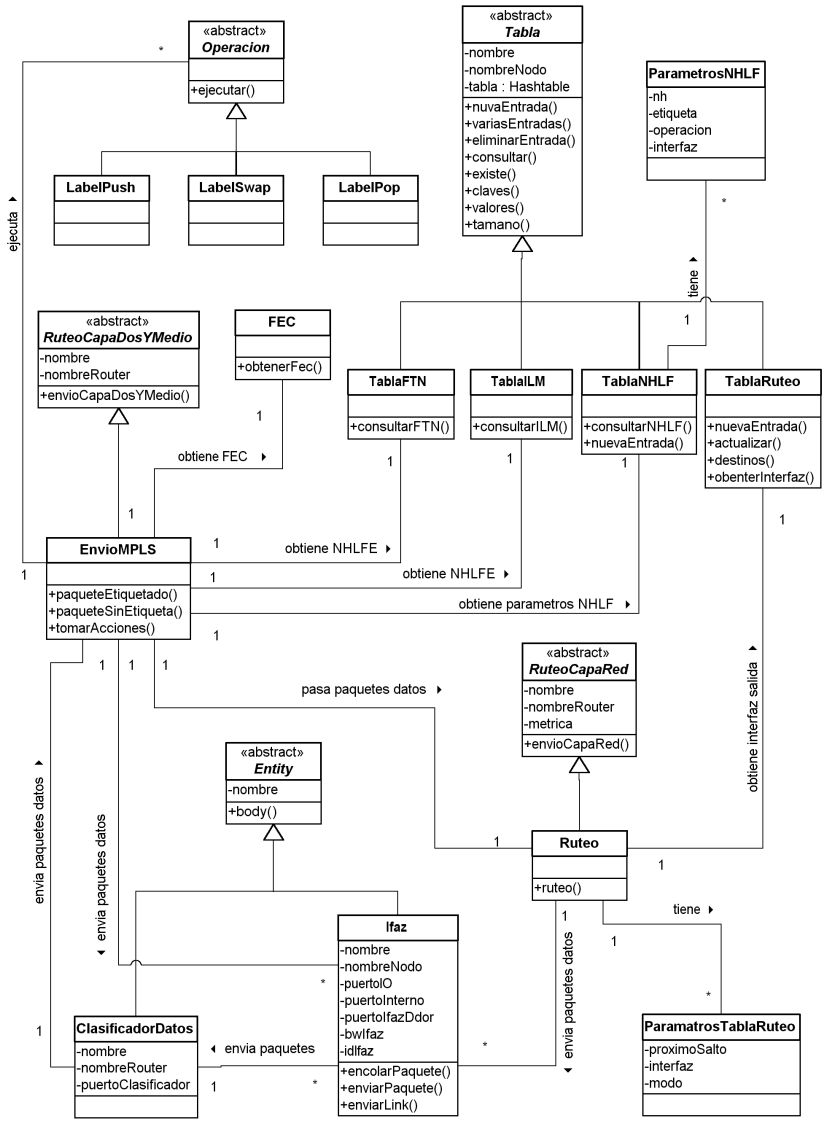


Figura 4.5: Diagrama de clase del módulo MPLS

Diagramas de Colaboración

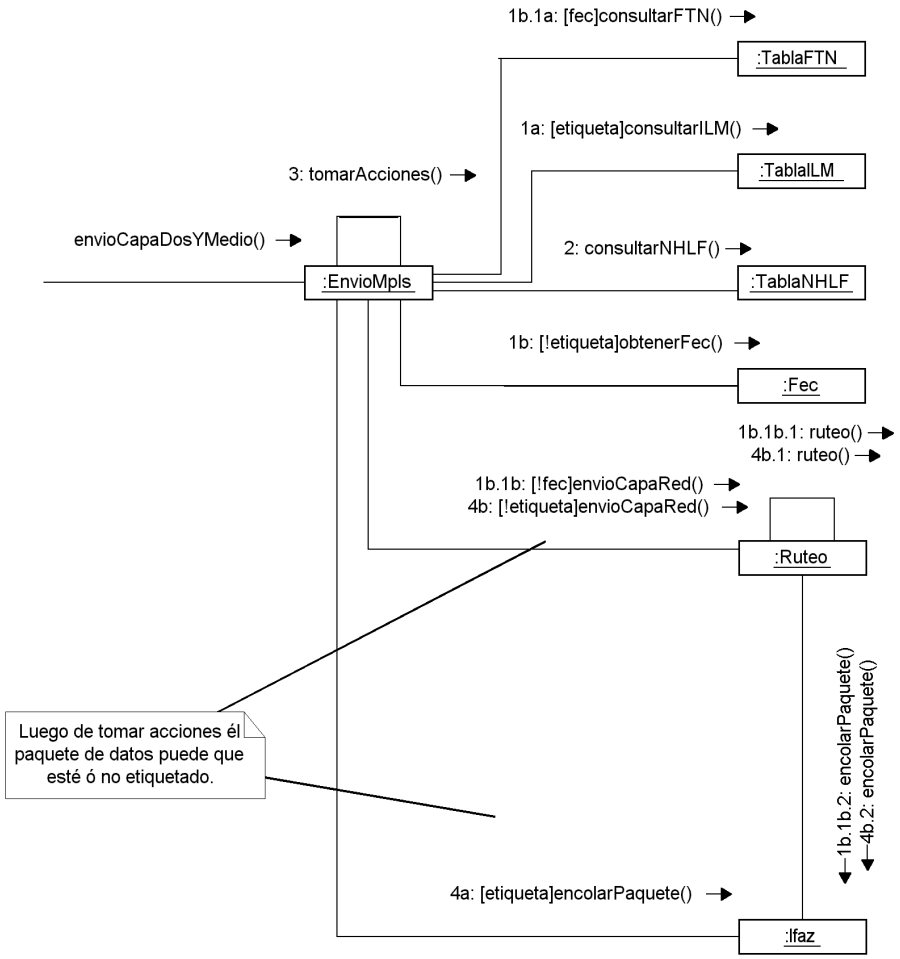


Figura 4.6: Diagrama de colaboración del módulo MPLS

4.2. FEC

4.2.1. Análisis de requerimientos

El procedimiento de mapeo del tráfico en la red se realiza mediante dos funciones. La primera es asignarle una FEC de acuerdo a criterios preestablecidos,

mientras que la segunda función se encarga de seleccionar la etiqueta con la cual se enrutaran los paquetes de la FEC. Es posible que para una FEC existan más de un LSP, permitiendo realizar balanceo de carga.

La asignación de una FEC para cada tráfico debe realizarse antes de que comience la simulación siendo ésta responsabilidad del usuario. No puede crearse más de una FEC para cada tráfico, es decir, al aplicar alguno de los criterios que se listan abajo, no puede existir la posibilidad de que un paquete pueda pertenecer a más de un grupo. Para que esto no suceda se debe definir una política, de manera que dado un paquete el cual puede pertenecer a más de un grupo de definiciones (grupo de FECs) definido en el LER de ingreso, se corresponda solo uno.

- Origen, Destino.
- Clase de tráfico, Origen, Destino.
- Clase de tráfico, Destino.
- Clase de tráfico, Origen.

La clase de tráfico, es un parámetro de "Diffserv"que identifica una de las clases preestablecidas. Un ejemplo de ello sería identificar las clases como oro, plata o bronce .

La segunda función la cual asocia una FEC a una entrada en la tabla NHLF, puede ser realizada manualmente por el usuario, antes de que comience la simulación o por el propio sistema.

Para que el usuario pueda realizar la asignación de la FEC a la NHLFE, debe conocer de ante mano las entradas asociadas a cada LSP a la tabla NHLF. En ésta versión solo sé podrá realizar esta asignación cuando se crean manualmente las tablas de MPLS, o dicho de otra manera, no podrá utilizar enrutamiento explícito.

Por otra parte, cuando se crean rutas explícitas, el sistema es el responsable de asignar a cada FEC uno ó un conjunto de LSP. Por lo tanto, una vez creado un LSP el sistema debe realizar ésta asociación.

Entonces, el usuario construye la FEC completando una tabla similar a la que se muestra en la figura 4.7, en donde se especifica el criterio y la FEC.

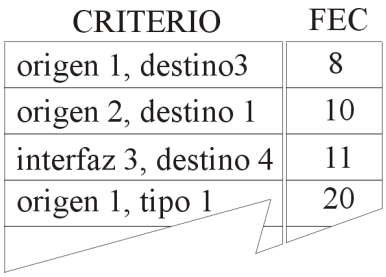


Figura 4.7: Definición de FEC

La función del módulo de FEC, queda resumida en la definición y asignación de las diferentes FEC a los paquetes de la red. Una vez que el usuario establece los diferentes criterios que definen las FEC e identifica a cada una de ellas de manera única, las restantes tareas del módulo son responder a solicitudes de determinación de la FEC de los paquetes que ingresen al nodo. El módulo de FEC debe responder retornando la FEC a la cual el paquete pertenece en el caso exista una definición en el que el paquete verifique. En caso que no exista una FEC asignada que verifique con el paquete el módulo, también debe responder indicando esta situación.

Cuando se ingresa una entrada en la tabla de definición de FEC, la misma debe contener un parámetro que indique el estado de la FEC. Es decir debe indicar si existe ó no un LSP creado, lo que se traduce en decir si la entrada esta activa ó inactiva. De esta manera cuando le realizan alguna consulta al módulo de FEC, este solo debe tener en cuenta las entradas activas. De lo contrario, el sistema intentaría enviar los paquetes que pertenezcan a una FEC inactiva por un LSP que no está establecido aún.

Casos de uso

Asignación de FEC

Caso de uso: Asignación de FEC

Actor: Usuario

Tabla 4.2: Caso de uso, asignación de FEC

Acción del actor	**Acción del sistema**
1. El Usuario realiza la definición de las FECs, estableciendo los criterios y representa cada FEC con una cadena de texto cualequiera pero única. Da la orden de ingresar estos datos al sistema[1].	

[1]La orden puede ser mediante la configuración de un archivo de texto, ó de otro medio de

2. El Usuario realiza el ingreso de todos los LSP que desea crear tanto en línea como fuera de línea. 3. El Usuario da la orden de que comience la simulación.	
	4. El sistema realiza una comprobación de errores, se fija que no haya FECs repetidas o mal definidas.
	5. El sistema realiza la asociación de las FEC y los LSP según el parámetro FEC del LSP, de los LSP creados de manera estática.
	6. Cuando se establece un nuevo LSP, es decir cuando el lsp setup arriba al LER de ingreso el sistema realiza la asociación de la FEC con el LSP creado, completando la tabla FTN e indicando como activa la entrada correspondiente en la tabla de definición de FEC. *Este procedimiento (paso 6) se repite a lo largo de toda la simulación cada vez que se crea un nuevo LSP.*
	7. Cuando el paquete lsp withdraw (por liberación de LSP) arriba a un LER de ingreso este actualiza las asociaciones de la FEC con los LSP actualizando la tabla FTN e indicando como inactiva a la entrada correspondiente en la tabla de definición de FEC.

configuración de la simulación

Diagramas de dominio

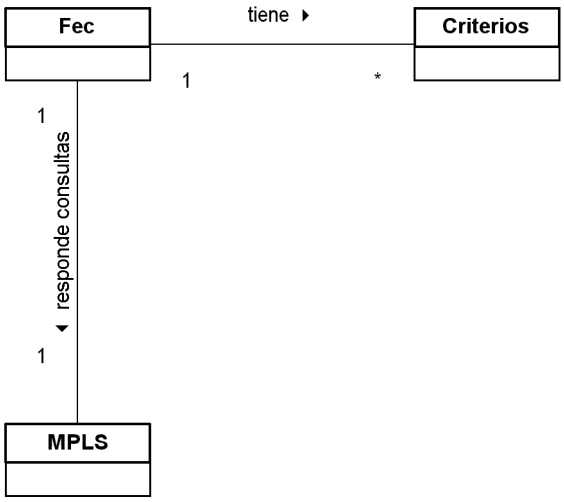


Figura 4.8: Diagrama de dominio del módulo FEC

4.2.2. Diseño

Diagrama de Clases

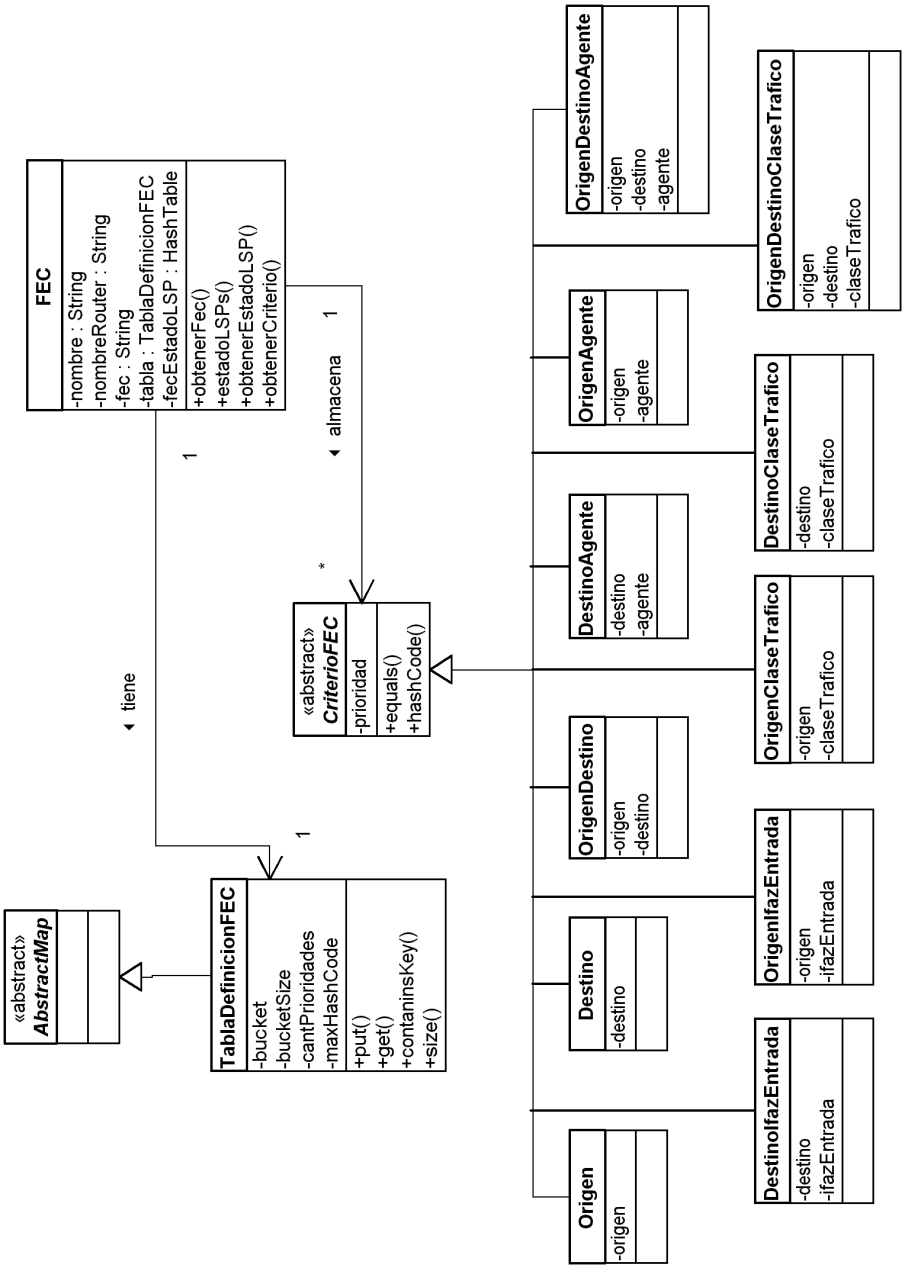


Figura 4.9: Diagrama de clase del módulo FEC

4.2.3. Implementación

En esta sección se presenta la solución implementada para satisfacer los requerimientos, impuestos para el módulo de FEC. El requisito más fuerte de satisfacer está vinculado con el proceso de determinar la FEC a la cual pertenece un paquete de datos que ingresa al nodo. Ver 4.2.

La solución tentativa para crear una tabla como se muestra en la figura 4.7, fue intentar usar un `Hashtable`. Ésto no es posible dado que al consultar el `Hashtable` para buscar un dato en la tabla, éste devuelve el primer valor para el cual la clave asociada coincide con el objeto que se usa como clave para buscar en la tabla, y en el caso de las FEC, más de un criterio podría coincidir con un paquete, pero solo se debe hacer corresponder a la FEC cuyo criterio es el más especifico.

Entonces para implementar la tabla de definición de FEC, se utilizó una versión modificada de un `Hashtable`. A continuación se explicará muy sucintamente el funcionamiento de un `Hashtable`, lo cual será muy útil para explicar como se implementó la tabla de definición de FEC.

Un `Hashtable`, es una tabla en la que se pueden almacenar valores indexados por algún objeto llamados clave. Para almacenar datos en la tabla se usa el método `put(Object clave, Object valor)`, y para obtener un valor se usa `get(Object clave)`. Las búsquedas en un `Hashtable` no son lineales, los elementos de la tablas poseen una ordenación tal que al momento de realizar las búsquedas no es necesario leer toda la tabla.

Los elementos en el `Hashtable` están indexados por un valor denominado "hashCode". En la clase `Object` existe un método `hashCode()` que devuelve este valor. Para fijar ideas, se muestra en la figura 4.10una representación gráfica de los `Hashtable`. Los `hashCode` se almacenan en un contenedor denominado "bucket", en cada posición del bucket puede existir ó no otro contenedor el cual almacena los datos (clave, valor).

Para almacenar un dato en la tabla, se extrae el `hashCode` de la clave. En base a dicho `hashCode` se ubica el dato en la tabla. Es posible que más de un objeto diferente pueda tener el mismo `hashCode`, en este caso cuando se almacena un dato en la tabla se guarda en la misma ubicación que todos los objetos que tengan el mismo `hashCode`. Éstos objetos son guardados en un contenedor como puede ser un `Array`. Las búsquedas entre objetos con del mismo `hashCode` son lineales, pero se supone que en una tabla que contenga unos cuantos elementos, el `Hashtable` es rápido para realizar las búsquedas.

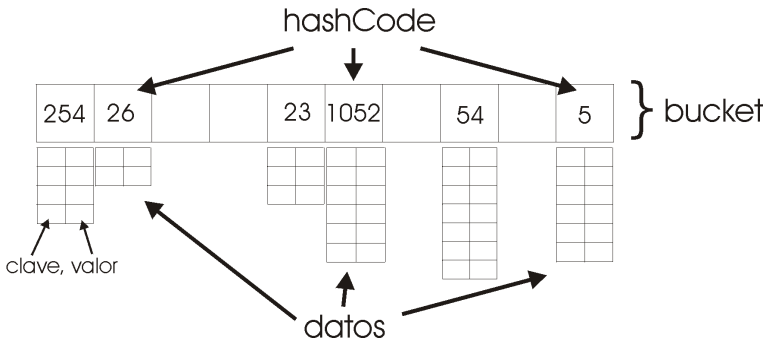


Figura 4.10: Representación gráfica de un Hashtable

Para finalizar con la presentación del `Hashtable`, cabe señalar que cuando se agrega un dato, se extrae el `hashCode` de la clave y se compara la clave con todos los elementos del `Hashtable` que poseen el mismo `hashCode`. Para ello se utiliza el método `equals()`. En caso de que se encuentre una clave que coincida con la que se quiere guardar se sobrescribe el valor y se devuelve el que estaba almacenado. Por otra parte, cuando se realiza una consulta, se extrae el `hashCode` del objeto que se utiliza para buscar en la tabla y por medio del método `equals()` se compara con las claves que tienen el mismo `hashCode` que el objeto con el cual se busca en la tabla y se devuelve la primer coincidencia.

La tabla de definición de FEC se implementó creando un contenedor de tipo Hash, donde los datos son indexados por el `hashCode` y por un índice al cual se denomina "prioridad". De esta manera el bucket queda definido por una matriz, es decir el contenedor que almacena los datos se ubica con dos parámetros. La clase que implementa este contenedor se denomina `TablaDefinicionFEC`.

La clase `TablaDefinicionFEC` define los métodos, `put(Object clave, Object valor)`, `get(Object clave)`, `containsKey(Object clave)` y `size()` para ingresar un nuevo dato, obtener un dato, consultar la existencia de alguna clave y obtener el tamaño de la tabla respectivamente.

Cuando se almacena un dato (clave, valor) en la `TablaDefinicionFEC`, con el `hashCode` y la `prioridad` de la clave, se encuentra la posición en el bucket donde se encuentra el array que contiene los datos. Si en esa posición ya se encuentra una clave que coincida con la del dato que se quiere ingresar, se reemplaza el valor por el nuevo y se devuelve el valor que estaba almacenado en la tabla .

Cuando se realiza una consulta a la tabla, se comienza buscando entre los datos de la tabla de mayor `hashCode` y mayor `prioridad` sin importar el `hashCode` de la clave del objeto que usa para realizar la búsqueda. Se devuelve el primer valor que coincida con la clave del objeto que usa para la búsqueda, para lo cual se usa el método `equals()`. El avance de la búsqueda por la tabla es, dado un `hashCode` se recorren todos los datos comenzando con los de mayor `prioridad` y finalizando en la menor `prioridad`. La siguiente iteración comienza en el siguiente `hashCode` de menor valor que el anterior hasta llegar al menor `hashCode` presente en la tabla.

Antes de explicar como funciona la `TablaDefinicionFEC`, se hace un repaso de como esta compuesta la misma. En la tabla de definición de FEC, se almacenan los criterios y la FEC. Los criterios son los que según ciertos parámetros definen una FEC. Estos parámetros podrían ser el nodo de origen y destino de un paquete. Para averiguar si un paquete pertenece a alguna FEC, se comparan los campos de origen y destino del paquete con los parámetros del criterio. Por otra lado, la FEC puede ser cualquier cadena de texto, pero debe de ser única en la red.

Los criterios deben heredar de la clase `CriterioFEC`, la cual es abstracta. La clase `CriterioFEC` tiene definido un método abstracto `equals()`, el cual debe ser implementado por los criterios (clases que heredan de `CriterioFEC`) y se utiliza para realizar comparaciones entre criterios, entre un criterio y `PaquetesDatos`. También define un el método `hashCode()`, que retorna el valor del `hashCode` correspondiente al criterio. Este valor es la cantidad de parámetros que lo definen, es decir es una medida cuan específico es el criterio. No es necesario que las clases que hereden de `CriterioFEC` implementen este método pero para que funcione es imprescindible que se definan únicamente como atributos públicos los parámetros del criterio. Por otra parte en la clase `CriterioFEC` se define el atributo `prioridad`. Este atributo se usa para otorgar una jerarquía entre criterios de igual especificidad, es decir entre criterios de igual cantidad de parámetros.

Cuando se ingresa un nuevo dato en la `TablaDefinicionFEC`, dentro del método `put(Object clave, Object valor)` se llama al método `hashCode()` y al método `getPrioridad()` de la clase `CriterioFEC` para obtener la posición en el bucket. Luego se comparan todos los datos del array usando el método `equals()` de la clave del nuevo dato.

Cuando se realiza una consulta, el objeto que se pasa como clave es de la clase `PaquetesDatos`, el cual puede pertenecer a cualquier criterio que este definido en la tabla, y además puede que coincida con más de uno. Se distinguen dos casos, cuando el paquete puede coincidir con más de un criterio pero de diferente cantidad de parámetros y cuando el paquete puede coincidir con más de un criterio de igual cantidad de parámetros.

Caso 1. Por ejemplo si existen definidos los criterios (Origen, Destino) y (Origen, Destino, Clase de Tráfico) el paquete podría satisfacer ambos criterios. Para el primer criterio, el `hashCode` vale 2 y para el segundo vale 3, de manera que la búsqueda comienza por el más especifico.

Caso 2. Por otro lado si existen definidos los criterios (Interfaz Entrada, Destino) y (Origen, Destino), el `hashCode` para ambos es 2, pero se debió haber establecido el valor de la prioridad cuando se programaron las clases que definen ambos criterios. A modo de ejemplo, sí se supone tienen el valor de 2 y 4 respectivamente, cuando se realiza la búsqueda se hace primero entre los datos que pertenecen al criterio (Interfaz Entrada, Destino).

En este punto debe quedar claro que se pueden definir cualquier criterio que el programador desee con solo heredar de la clase `CriterioFEC`, definir los parámetros e implementar el método `equals()`.

4.3. Señalización

4.3.1. Análisis de requerimientos

El objetivo del módulo de Señalización, es el establecimiento de LSP. Los LSP pueden ser creados de manera estática ó dinámica. A pesar de que para el establecimiento estático no es necesario utilizar un protocolo de señalización, para el establecimiento dinámico si lo es.

En el establecimiento estático, el usuario es el encargado de configurar manualmente las tablas de los routers, tanto internos como de borde. Es inmediato pensar que no se necesita de un protocolo de señalización para este tipo de establecimiento.

En el establecimiento dinámico, el encargado de configurar las tablas es el protocolo de señalización, pero el LSP, puede ser determinado por el usuario o por algún protocolo de ruteo.

De ahora en más no hablaremos de establecimiento dinámico, se hablará de rutas explícitas establecidas fuera de línea ó en línea. A continuación se describen ambas modalidades.

LSP explícito

Una ruta explícita (Explicit Route (ER)) es una secuencia de nodos lógicos entre un nodo de ingreso y otro de egreso, la cual se define y establece desde un nodo de la frontera. A pesar que existen varias formas de especificar una ruta explícita, en ESR sólo existe una. Para el simulador los nodos lógicos son los routers de la red, en donde se distinguen dos tipos, LSR, y LER.

Los caminos son creados por los LER. El LER de aguas arriba es el que inicia la petición del establecimiento del camino. Para que esto sea posible, debe estar creada la tabla de ruteo de capa de red en cada router. El encargado de realizar esta tabla es el protocolo de ruteo de dicha capa.

El usuario, especifica en que tiempo se debe iniciar el establecimiento del LSP. En ese momento, el LER de ingreso envía un paquete[2] de petición de establecimiento (lsp request) por la ruta explícita. Esto implica que antes de enviar el lsp request, el LER debe conocer cual es la ER a establecerse. Esta información llega al LER mediante un procedimiento realizado fuera de línea por el usuario, o en línea por un algoritmo de búsqueda del mejor camino implementado por él módulo de Rutas (ver 4.6).

Cuando el lsp request llega al LER de egreso, este envía un paquete de establecimiento (lsp setup) por la ER. Mediante el lsp setup, el router de aguas abajo (downstream) comunica al router de aguas arriba (upstream) que eti-

[2]Por más información consultar la sección Paquetes de señalización.

quetas debe usar para los paquetes que sean enviados por dicho LSP. Este procedimiento se repite a lo largo de toda la ER hasta llegar al LER de ingreso, momento en el cual queda establecido el LSP. El proceso descrito para la creación del LSP es el procedimiento básico. Existen otros tópicos que deben tenerse en cuenta, los cuales se irán abordando poco a poco para no ahondar en complejidades. Antes de continuar con los mismos, se definen las propiedades de los LSP.

El LSP tiene diferentes parámetros que pueden ser establecidos por el usuario, los cuales son usados a la hora de establecer el LSP, liberarlo (eliminarlo), asociar las clases tráfico que circulan por el mismo, etc. Los parámetros se clasifican en esenciales y opcionales, ver 4.11.

ER especificada fuera de línea Como se mencionó antes, para el establecimiento de un LSP, se necesita información de la ER, la cual puede ser especificada fuera de línea. Dicha especificación debe contener los parámetros esenciales y podrá contener los opcionales.

La ruta explícita será una secuencia de nodos:

$$[\text{LER1, LSR1, LSR2, } \cdots \text{, LER5}]$$

La secuencia comienza con el LER de ingreso y finaliza con el LER de egreso, entre ambos se especifican los saltos por los LSR.

ER especificada en línea En este caso la ER es calculada por él modulo Rutas (ver 4.6), según las restricciones impuestas por la métrica. Las métricas disponibles serán:

Minimum Hop Routing: mínima cantidad de saltos.

Minimum Administrative Weight Routing: enlaces de menor peso.

1/BWreservado: métrica cuyos pesos se basan en el inverso del BW reservado.

1/BWdisponible: es una métrica opuesta a la anterior.

El usuario deberá elegir un criterio de desempate en caso de que el algoritmo de búsqueda del mejor camino encuentre más de una opción válida. En la sección del módulo de Rutas (ver 4.6) se encontrará una explicación más detallada sobre este tema.

Para poder calcular la ER se necesita especificar entre que nodos se desea hacerlo. En primera instancia la ER se calcula entre LERs, es decir se debe especificar el origen, (LER de ingreso) y el destino (LER de egreso).

(a) Parámetros esenciales

Parámetros esenciales	
ER	ruta explícita
FEC	clase de equivalencia para el envío de los paquetes
LSPID	identificador numérico del LSP
time	tiempo de simulación en que se crea el LSP

(b) Parámetros opcionales

Parámetros opcionales	
BW	ancho de banda ha reservar
plano	plano de reserva de BW, puede ser Control ó Datos
ultima operación	indica que operación debe realizar el LER de egreso con la etiqueta de los paquetes que viajan por el LSP
setup	prioridad de establecimiento del LSP (max. 1, min. 7)
hold	prioridad para liberar el LSP (max. 1, min. 7)
tráfico	tipo de tráfico que circulará por el LSP
time-off	tiempo en que se liberará el LSP
persistencia	intervalo de tiempo en que se reintentara crear el LSP en caso de falla del establecimiento
reroute	indica si el ER-LSP permite que sea vuelto a enrutar (válido solo para ER en línea)
backup	ER alternativa para ser usada en caso de falla

Figura 4.11: En las tablas se muestran los parámetros de los LSP los cuales son usados para el establecimiento, mantenimiento y liberación

LSP con y sin restricciones

En las secciones anteriores se mencionó que cuando se establece un LSP, se pueden reservar recursos de la red. En esta sección se describen los procesos vinculados a la reservación de recursos cuando se establece el LSP. No se describe como se reservan los recursos, porque este tema se aborda en la sección que trata sobre el módulo de Recursos (ver 4.7).

Cuando se habló sobre el establecimiento de LSP, se explicó que por medio de los paquetes lsp request y lsp setup, se transmiten la información entre los nodos de la red. Sólo se mencionó que se realizaba la distribución de etiquetas por medio del lsp setup. Ahora se verá el resto de la información que transportan dichos paquetes.

LSP sin restricciones de BW Cuando la especificación del LSP no cuenta con los parámetros que especifican los recursos, se procede a crearse un LSP sin reservar recursos. Este LSP puede ser creado tanto fuera de línea o en línea. Si es creado fuera de línea, no se especifica la reserva de Ancho de Banda (BW).

En el proceso de establecimiento del LSP (ver 4.3.1), el paquete lsp setup, lleva la información de la etiqueta que debe usar el upstream para enviar paquetes por dicha interfaz al downstream y los parámetros de la ER. Con esta información, se actualiza la ILM y la NHLF, y es suficiente para que los nodos puedan identificar a los LSP y puedan realizar algún control sobre los mismos.

LSP con restricciones Se pueden crear LSP imponiendo alguna restricción y reservando recursos, tanto fuera de línea como en línea. La idea es que se pueda imponer una restricción para realizar la búsqueda de la mejor ruta, reservando recursos o no por la misma. En esta versión solo se reserva ancho de banda de los enlaces.

Fuera de línea En este apartado se verá una descripción del establecimiento de un LSP donde la ER es especificada fuera de línea, conjuntamente con el parámetro BW.

Cuando se inicia el proceso de establecimiento del LSP, el LER de ingreso envía el paquete lsp request que este transporta los parámetros del LSP. Cuando el lsp request arriba a un nodo (LSR o LER), en primera instancia se verifica en la tabla de recursos la existencia de suficientes recursos en el enlace que une el nodo actual con el próximo salto. Si así sucede, se continúa con el procedimiento hasta llegar al LER de egreso. Cuando el lsp request arriba al LER de egreso, envía de regreso por la ER el paquete lsp setup que al arribar a cada nodo primero se vuelve a chequear que existan recursos suficientes y luego se actualiza la la tabla de recursos, la ILM y la NHLF. Este procedimiento se repite en cada nodo de la red hasta llegar al LER ingreso. En el LER de ingreso además se actualiza la tabla FTN y a partir de ese momento queda establecido el LSP.

Sí en el proceso de establecimiento descrito en el párrafo anterior se encuentra que no hay suficiente recursos disponibles, antes de darse por vencido, se verifica si existen LSP establecidos y cual tiene más prioridad, sí el existente o el que se quiere establecer. Este procedimiento se describe en la sección preemption 4.3.1. Los casos en que no se establece se estudian en la sección falla de establecimiento.

En línea El proceso de establecimiento del LSP en línea es algo diferente al proceso fuera de línea, pero cuando sé realiza el mapeo del LSP sobre la red, el procedimiento es el mismo.

Cuando se inicia el proceso de establecimiento, el LER de ingreso realiza un llamado al módulo de Rutas (ver 4.6), para que calcule la ER, basándose en los parámetros que este le indica. Estos parámetros son: métrica, origen, destino, criterio, BW, setup y hold. El detalle de cómo funciona se encuentra en la sección del módulo de Rutas. Si bien se pasan los parámetros de setup y hold, el módulo de Rutas no realiza funciones de preemption. Solo usa estos parámetros en caso de que no encuentre una camino posible entre el origen y destino porque los recursos están siendo usados por otros LSPs. En ese caso consulta si es posible que se pueda establecer el LSP apoderándose de los recursos de otro LSP de menor prioridad. En caso de que así sea, el módulo de Rutas además de devolver la ER calculada, también devuelve el LSPID perteneciente al LSP que hay que eliminar.

Si el módulo de Rutas consigue hallar una ER, el LER de ingreso procede enviar un paquete lsp request por dicha ER, transportando con él los parámetros del LSP. Se procede de la misma manera para realizar el establecimiento del LSP que el caso fuera línea (ver 4.3.1).

Liberación de LSP

La liberación de un LSP, se refiere a la eliminación del mismo. Este hecho puede originarse por varias causas, las cuales se pasan a enumerar:

- el LSP alcance su tiempo de vida especificado por el parámetro time-off.
- un fallo en régimen de la red, es decir que se caiga un router o un enlace, después de que se estableció el LSP.
- fallo en el establecimiento.
- preemption.
- se decida encaminar el LSP por otro camino.

Tiempo de vida alcanzado Cuando el LER de ingreso detecta que el tiempo de vida de uno de sus LSPs alcanzó su fin, este envía por la ER un paquete lsp ralease. Cuando el lsp ralease llega al LER de egreso este inmediatamente envía

por la ER un paquete lsp withdraw, que es re-enviado por los LSR hasta que llegua al LER de ingreso momento en el cual el LSP queda completamente eliminado.

Cuando el paquete lsp withdraw, arriba a cada nodo se elimina la entrada en la ILM y NHLF, se liberan los recursos, eliminando la entrada en la tabla de recursos y realizando las acciones necesarias en el plano de datos.

La caída de un enlace o de un nodo de la red es detectada, por el envío de paquetes de actualización. Cuando se envía un paquete hello por una interfaz y no se recibe respuesta, se da cuenta que el enlace o el nodo esta caído.

Fallo de la red Cuando el upstream (respecto al enlace o nodo caído) detecta la caída, este envía hacia el LER de ingreso un paquete de notificación, lsp notify, indicando la falla, este a su vez podrá o no tomar la decisión de liberar el LSP según la política de re-enrutamiento que haya escogido el operador de red. En caso de que la opción sea liberar el LSP, el LER de ingreso envía un paquete lsp ralease por la ER hasta el nodo que envío la notificación del fallo.

Por otro lado cuando el downstream (respecto al enlace o nodo caído) detecta la falla, envía un lsp notify hacia el LER de egreso, este basándose también en la política de re-enrutamiento, envía o no un paquete lsp ralease para eliminar el LSP.

Fallo en el establecimiento Si en el intento de establecer un LSP, uno de los paquetes lsp request o lsp setup se pierde en alguna parte de la red no se podrá completar el establecimiento. El LER de ingreso tiene un parámetro llamado persistencia, el cual indica el tiempo en que se debe reintentar establecer el LSP.

Una vez alcanzado el tiempo de persistencia se vuelve a intentar establecer el camino y el contador de tiempo comienza a correr nuevamente de cero. Cuando un paquete lsp request arriba a un nodo se verifica entre otras cosas que no exista una entrada en la tabla NHLF con el mismo LSP Identifier (LSPID), si esto ocurre, se detiene el proceso de establecimiento e inmediatamente se envía un paquete lsp notify hacia el LER de ingreso para que tome las correspondientes acciones.

Cuando llega un paquete lsp notify al LER de ingreso indicando que no se puede establecer el LSP porque ya esta establecido, este envía un lsp ralease para eliminar el LSP (de igual manera que en 4.3.1), luego de que se completa la tarea, se comienza nuevamente con el establecimiento del LSP.

Preemption En caso de que un LSP sea removido por otro de mayor prioridad en algún enlace, el nodo implicado (upstream respecto al enlace) debe enviar un paquete notify hacia el LER de ingreso del LSP correspondiente para indicar que se debe liberarlo.

El LSP eliminado puede ser enrutado nuevamente según las políticas de re-enrutamiento. Si por el contrario el LSP no se puede re-enrutar el tráfico que circulaba por el pasa a ser encaminado según el protocolo de capa de red

Re-enrutamiento del LSP Cuando se decide re-enrutar un LSP que esta operativo, luego de que se establece el nuevo LSP, el LER de ingreso realiza la liberación del anterior de igual forma que en el caso que se haya alcanzado el tiempo de vida (ver 4.3.1).

Apoderamiento o Preservación de LSPs (Preemption)

Los LSP tienen asociados los parámetros setup y hold, los cuales si no se especifican son establecidos como un LSP de baja prioridad. Estos se utilizan para establecer una jerarquía de LSP, y sirve para dar lugar al establecimiento de LSP de más prioridad y asegurar la permanencia de los mismos.

El parámetro setup, indica la prioridad de establecerse cuyo valor varía de 1 a 7, el valor 1 es el de mayor prioridad. El parámetro hold, indica la prioridad de permanencia, con la misma escala que el setup.

Cuando se va establecer un LSP y no tiene recursos suficientes en el enlace, pero existen LSP establecidos, se compara el parámetro setup del LSP a establecerse contra el parámetro hold del LSP establecido, el que tenga mayor prioridad de los dos es quién gana, en caso de empate gana el que LSP que está establecido.

Políticas de re-enrutamiento

Existen dos parámetros del LSP reroute y backup que son utilizados para establecer políticas de re-enrutamiento.

Cuando una ER es especificada fuera de línea, es posible sólo especificar el parámetro de backup, el cual es una ruta explícita alternativa. La idea de esta ruta de respaldo es que no contenga ningún nodo de la ruta principal (no es una restricción) para que en caso de falla de la ER principal, se pueda establecer el LSP por la ruta alternativa. Este parámetro es opcional.

Si la ER es especificada en línea, se puede establecer el parámetro reroute para indicar al LER si esta ruta permite o no volver a ser calculada y por lo tanto tomar otra ruta alternativa.

Ambas políticas de re-enrutamiento, asumen una escena de recuperación global y no local, es decir en caso de detectar la falla en un nodo interno, se libera el LSP y se enruta desde el LER de ingreso hasta el LER de egreso. No se busca un camino alternativo entre los nodos internos que son bordes de la falla, es decir si la falla es un enlace caído no se busca otro camino que une los LSR que están aguas arriba y abajo del enlace caído.

Paquetes de señalización

Son los paquetes originados dentro de los límites del dominio MPLS, y su utilidad es para el mantenimiento y configuración de la red. Los paquetes de control son encaminados por la red directamente por el módulo al cual pertenezcan. En este caso, el propio módulo de Señalización es quién determina y envía el paquete por la correspondiente interfaz. A continuación se enumeran los diferentes paquetes de control con una breve reseña indicando su función.

lsp request: Es creado por el LER de ingreso y es utilizado para realizar el establecimiento de los LSP. Este paquete recorre la ER desde el LER de ingreso hasta el LER de egreso. No realiza modificaciones sobre la red, solo verifica que se cumplan las condiciones para establecer la ruta.

lsp setup: Es creado por el LER de egreso y este se utiliza para que los nodos de la red realicen los cambios necesarios para establecer el LSP y transporta las etiquetas que se usarán para enviar los paquetes por el LSP a crearse.

lsp ralease: Es creado por el LER de ingreso y es utilizado para que iniciar el procedimiento de eliminación de un LSP. Este paquete viaja desde el LER de ingreso hasta el de egreso.

lsp withdraw: Se utiliza para llevar a cabo la eliminación de un LSP, es creado por el LER de egreso. Cuando este paquete arriba a un nodo este procede a eliminar el LSP que el paquete informa.

lsp notify: Puede ser creado por cualquier nodo de la red, y su función es de notificar hechos anómalos, como la caída de un enlace.

lsp hello: Es creado y utilizado por todos los nodos de la red para determinar si existe comunicación con los vecinos. Es usado por los nodos para anunciarse cuando aparecen en la red. (por ejemplo cuando un nodo se levanta luego de estar caído)

Casos de uso

Establecimiento de LSP

Caso de uso: Establecimiento de LSP

Actor: Usuario

Tabla 4.3: Caso de uso del establecimiento de LSP

Acción del actor	**Acción del sistema**
1. El Usuario configura los correspondientes parámetros del LSP y da la orden de que comience la simulación.	

	2. El sistema avisa al LER de ingreso que se debe establecer el LSP. 3. Si la ER esta especificada fuera de línea, se continua con el siguiente paso, de lo contrario, el LER de ingreso calcula la ER según información de nodo origen, destino, BW, otras restricciones y criterios. 4. El LER de ingreso crea un paquete lsp request y lo envía al siguiente nodo de la ER. 5. El lsp request arriba a un LSR. El LSR verifica si son suficientes los recursos para establecer el nuevo LSP. Siendo así se envía el lsp request hacia el próximo nodo de la ER. *El paso 5 se repite en todos los nodos de la red hasta que el lsp request arriba al LER de egreso.* 6. El lsp request arriba al LER de egreso. Este crea un paquete lsp setup, adjuntándole además de la información de la ER, la etiqueta que deberá emplear el upstream para enviar los paquetes hacia el downstream. 7. El LER de egreso envía el lsp setup al nodo anterior a él (upstream). 8. El lsp setup arriba a un LSR. Este verifica nuevamente si se satisfacen los recursos, si es así se genera la etiqueta que el upstream debe usar para enviarle paquetes y se actualizan la ILM y la NHLF. También si es necesario se actualiza la tabla de recursos. 9. El LSR envía el lsp setup al siguiente nodo de la ER (upstream). *El paso 8 y 9 se repite en todos los nodos de la red hasta que el lsp request arriba al LER de ingreso.*

	10. Cuando el lsp setup arriba al LER de ingreso finaliza el establecimiento, realizando la asociación de la FEC con el LSP. Para ello se actualiza la FTN y en la tabla de definición de FEC se indica el estado de activa a la correspondiente FEC. También aquí se actualiza la tabla de recursos si es necesario.

Generación de un lsp request

Caso de uso: Generación de un lsp request.

Actor: Módulo LSP, Usuario.

Tabla 4.4: Caso de uso, generación de un lsp request

Acción del actor	**Acción del sistema**
1. El Usuario configura los correspondientes parámetros del LSP y da la orden de que comience la simulación.	
	2. La unidad de control avisa, cuando se produce el evento que da la orden de comenzar el establecimiento del LSP.
3. El Módulo de Señalización obtiene la ER, la cual si no esta dada fuera de línea, se debe crear en línea. Es decir el Modulo de Señalización le solicita al Módulo de Rutas que encuentre una ER según los parámetros del LSP.	
	4. El Módulo de Rutas busca el mejor camino según los parámetros y métricas del LSP.
5. El Módulo Señalización crea un paquete lsp request, el cual contiene toda la información pertinente al LSP, incluida la ER.	
6. El Módulo Señalización toma el siguiente salto de la ER y determina la interfaz por la cual debe enviar el paquete.	

	7. El lsp request es enviado por la correspondiente interfaz hacia el próximo salto.

Arribo de un lsp request a un LSR

Caso de uso: Arribo de un lsp request a un LSR.

Actor: Módulo Señalización.

Tabla 4.5: Caso de uso, arribo de un lsp request a un LSR

Acción del actor	**Acción del sistema**
	1. Arriba un paquete a un LSR.
	2. El Clasificador del plano de datos identifica al paquete. Al ser un paquete de control lo deriva al plano de control, donde es tomado por el Módulo Señalización.
3. El Módulo Señalización, identifica de cual paquete de control se trata.	
4. El Módulo Señalización, al ser un lsp request consulta si existen suficientes recursos para establecer el nuevo LSP, por lo cual pasa los parámetros BW, setup y hold.	
	5. El Módulo de Recursos verifica si existen suficientes recursos disponibles.
6. Con recursos suficientes, el Módulo Señalización, determina la interfaz por la cual debe enviar el paquete.	
	7. Se envía el paquete por la interfaz correspondiente.

Arribo de un lsp request a un LER de egreso

Caso de uso: Arribo de un lsp request a un LER de egreso

Actor: Módulo Señalización.

Tabla 4.6: Caso de uso, arribo de un lsp request a un LER de egreso

Acción del actor	**Acción del sistema**
	1. Arriba un paquete a un LER de egreso.
	2. El Clasificador del plano de datos identifica al paquete. Al ser un paquete de control lo deriva al plano de control, donde es tomado por el Módulo Señalización.
3. El Módulo Señalización, identifica de cual paquete de control se trata.	
4. Al ser un lsp request, el LER de manera automática crea un paquete lsp setup el cual contiene la información necesaria del LSP.	
5. Se genera una nueva etiqueta la cual se adjunta al lsp setup.	
6.En caso de que la última operación sobre la etiqueta sea un POP o un Swap se debe actualizar las tablas ILM y NHLF para el LER en cuestión.	
7. El Módulo Señalización, determina la interfaz por la cual debe enviar el paquete.	
	8. Se envía el paquete por la interfaz correspondiente.

Arribo de un lsp setup a un LSR

Caso de uso: Arribo de un lsp setup a un LSR.

Actor: Módulo Señalización.

Tabla 4.7: Caso de uso, arribo de un lsp setup a un LSR

Acción del actor	**Acción del sistema**
	1. Arriba un paquete a un LSR.

	2. El Clasificador del plano de datos identifica al paquete. Al ser un paquete de control lo deriva al plano de control, donde es tomado por el Módulo Señalización.
3. El Módulo Señalización, identifica de cual paquete de control se trata.	
4. Al ser un lsp setup, el Módulo Señalización, extrae información del paquete. Obtiene la ER, LSPID, BW, setup, hold. Además de estos parámetros, extrae la etiqueta que envía el downstream con la cual se le deben enviar paquetes por el LSP, y la interfaz de salida (interfaz por la cual llegó el paquete).	
5. El Módulo Señalización, solicita que se reserven recursos para establecer el nuevo LSP, por lo cual le pasa los parámetros BW, setup y hold.	
	6. El Módulo de Recursos verifica si existen suficientes recursos disponibles.
	7. Como existen suficientes recursos, el Módulo de Recursos, reserva los mismos y actualiza la tabla de recursos, con los datos del LSP.
	8. Contesta diciendo que se reservaron los recursos.
9. El Módulo Señalización, toma la primer etiqueta disponible de su colección y la sustituye por la etiqueta recibida del downstream.	
10. El Módulo Señalización, actualiza la ILM y la NHLF.	
11. El Módulo Señalización, determina la interfaz por la cual debe enviar el paquete.	
	12. Se envía el paquete por la interfaz correspondiente.

Arribo de un lsp setup a un LER de ingreso

Caso de uso: Arribo de un lsp setup a un LER de ingreso

Actor: Módulo Señalización.

Tabla 4.8: Caso de uso, arribo de un lsp setup a un LER de ingreso

Acción del actor	**Acción del sistema**
	1. Arriba un paquete a un LER de ingreso.
	2. El Clasificador del plano de datos identifica al paquete. Al ser un paquete de control lo deriva al plano de control, donde es tomado por el Módulo Señalización.
3. El Módulo Señalización, identifica de cual paquete de control se trata.	
4. Al ser un lsp setup, el Módulo Señalización, extrae información del paquete. Obtiene la ER, LSPID, BW, setup, hold. Además de estos parámetros, extrae la etiqueta que envía el downstream con la cual se le deben enviar paquetes por el LSP, y la interfaz de salida (interfaz por la cual llegó el paquete).	
5. El Módulo Señalización, solicita que se reserven recursos para establecer el nuevo LSP, por lo cual le pasa los parámetros BW, setup y hold.	
	6. El Módulo de Recursos verifica si existen suficientes recursos disponibles.
	7. Como existen suficientes recursos, el Módulo de Recursos, reserva los mismos y actualiza la tabla de recursos, con los datos del LSP.
	8. Contesta diciendo que se reservaron los recursos.
9. El Módulo Señalización, actualiza la FTN y la NHLF, con lo cual queda establecido el nuevo LSP.	

Diagramas de dominio

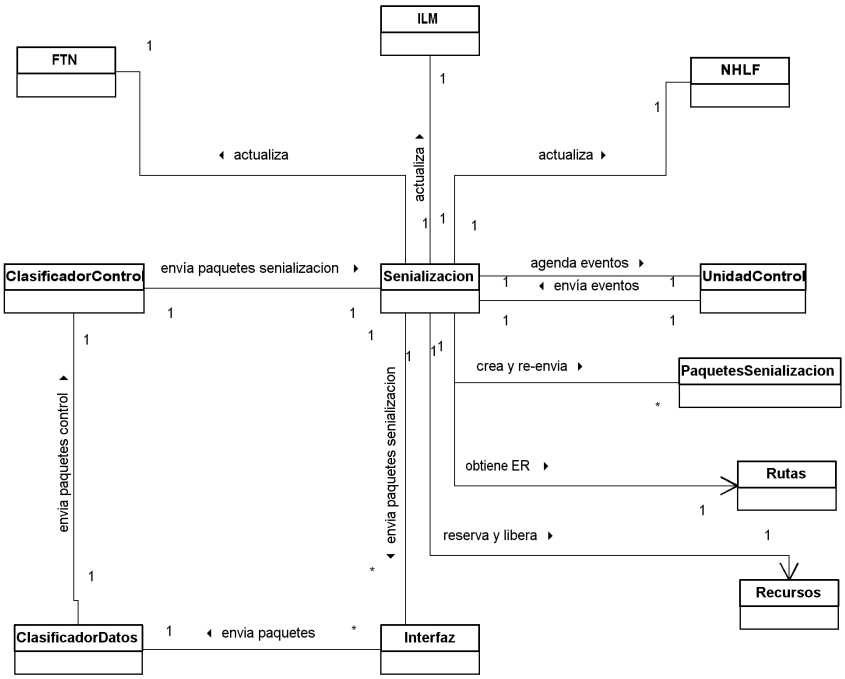


Figura 4.12: Diagrama de dominio del módulo Señalización

4.3.2. Diseño

Diagrama de Clases

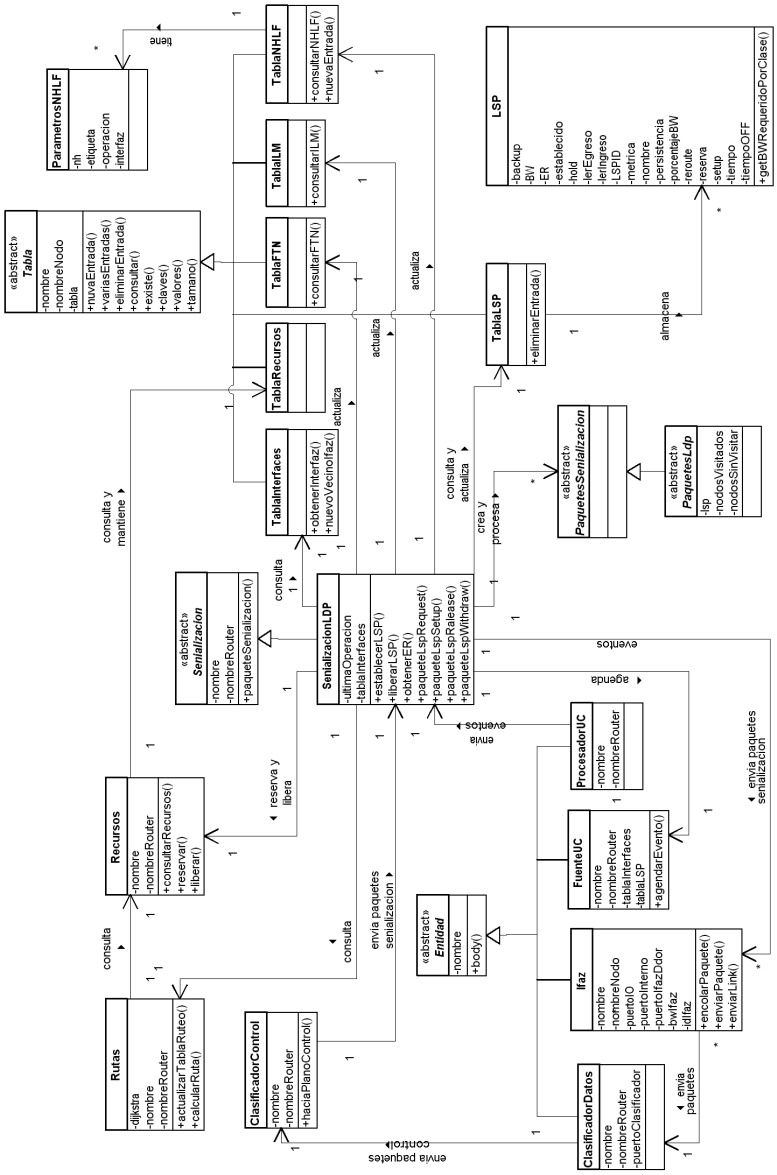


Figura 4.13: Diagrama de clase del módulo Señalización

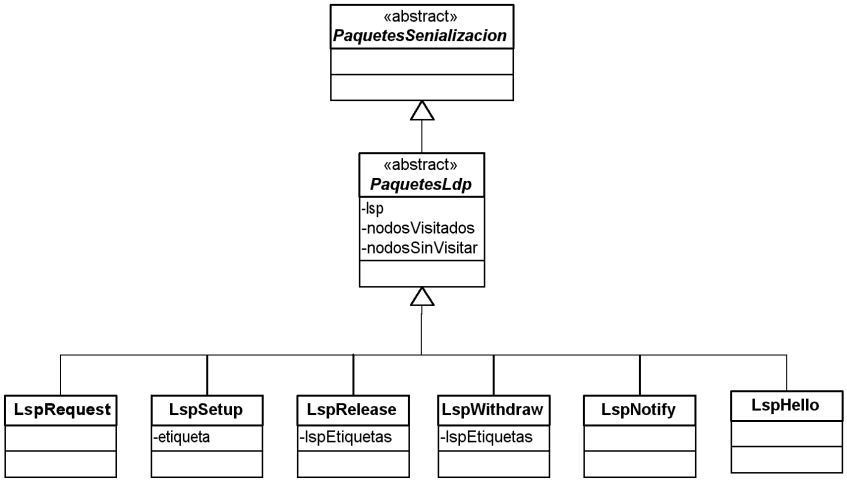


Figura 4.14: Diagrama de clase de los paquetes de Señalización

Diagramas de Colaboración

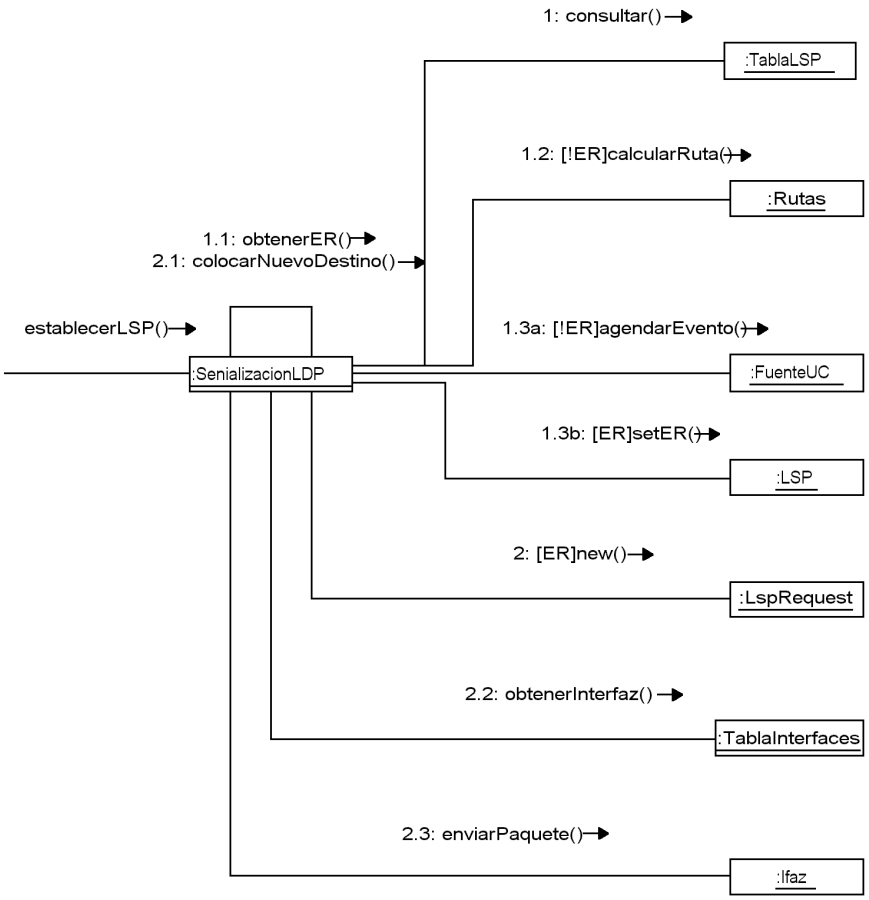


Figura 4.15: Diagrama de colaboración de la creación de un paquete lsp request.

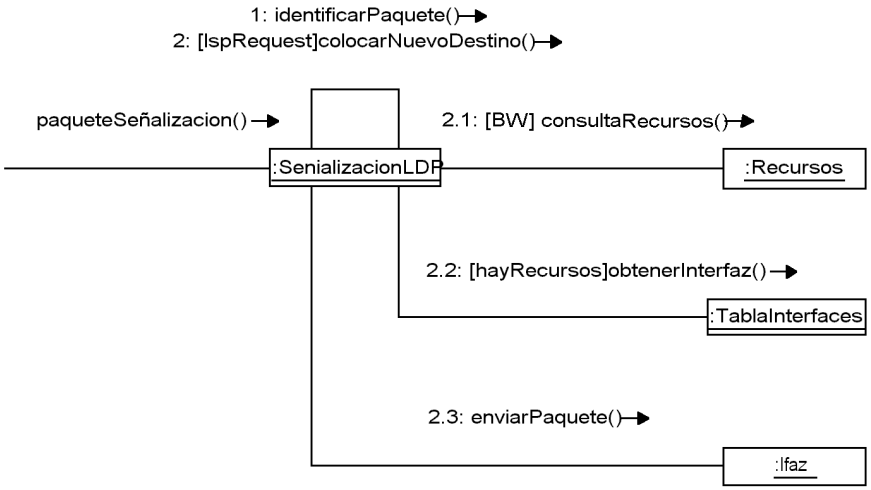


Figura 4.16: Diagrama de colaboración del arribo de un paquete lsp request a un nodo interno de la red.

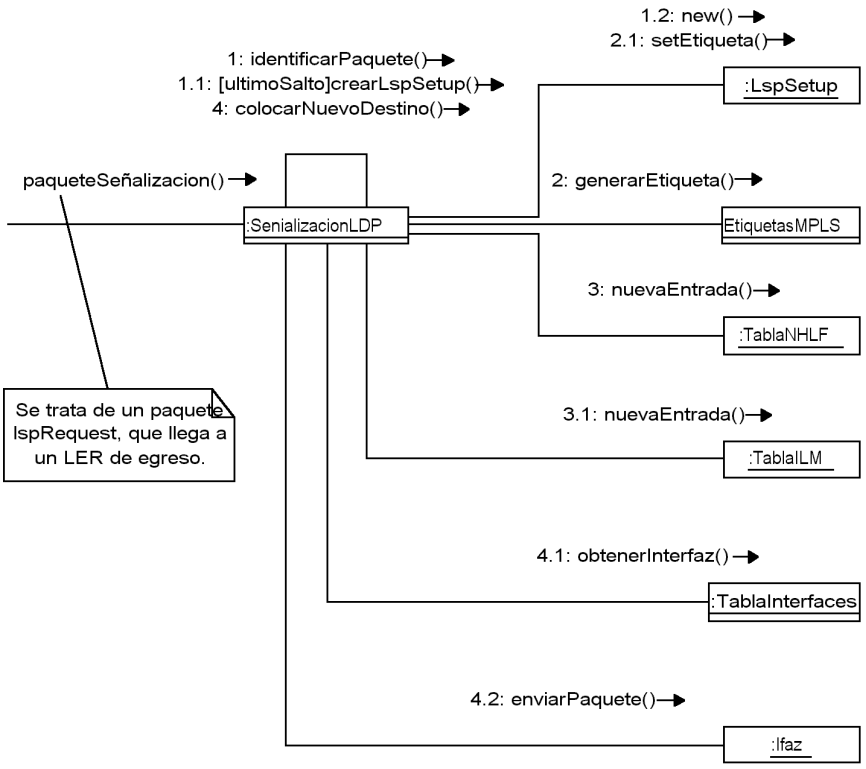


Figura 4.17: Diagrama de colaboración del arribo de un paquete lsp request a un LER de egreso.

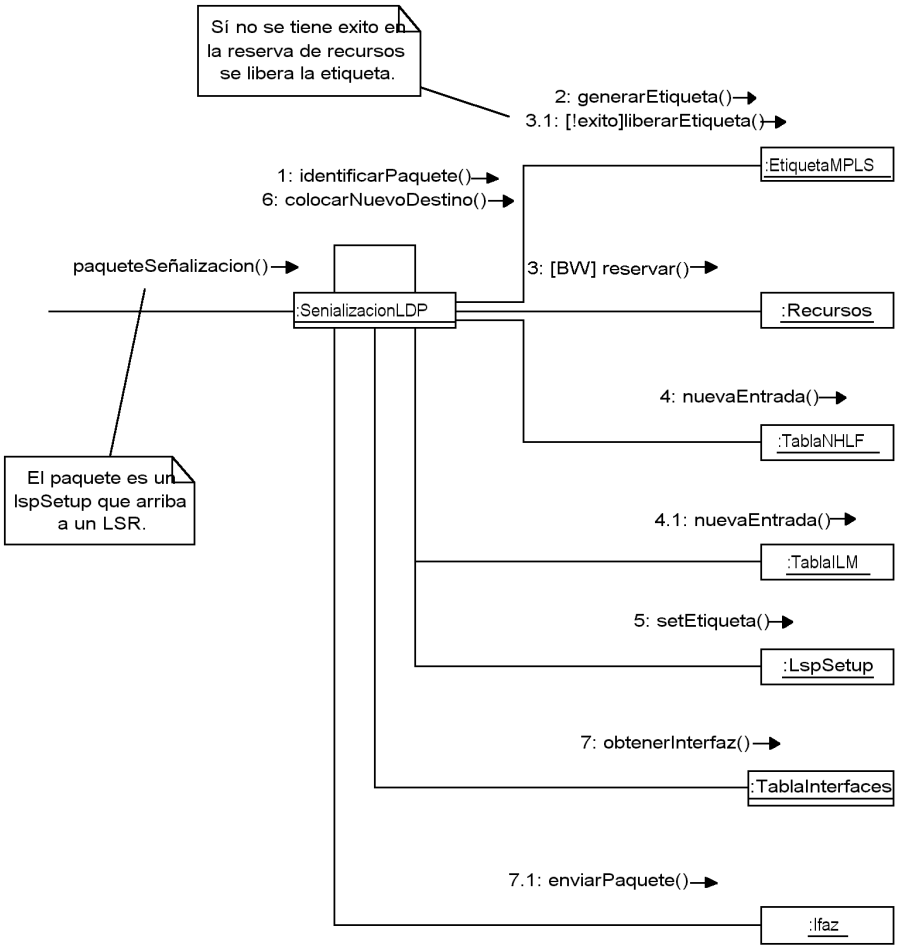


Figura 4.18: Diagrama de colaboración del arribo de un paquete lsp setup a un nodo interno de la red.

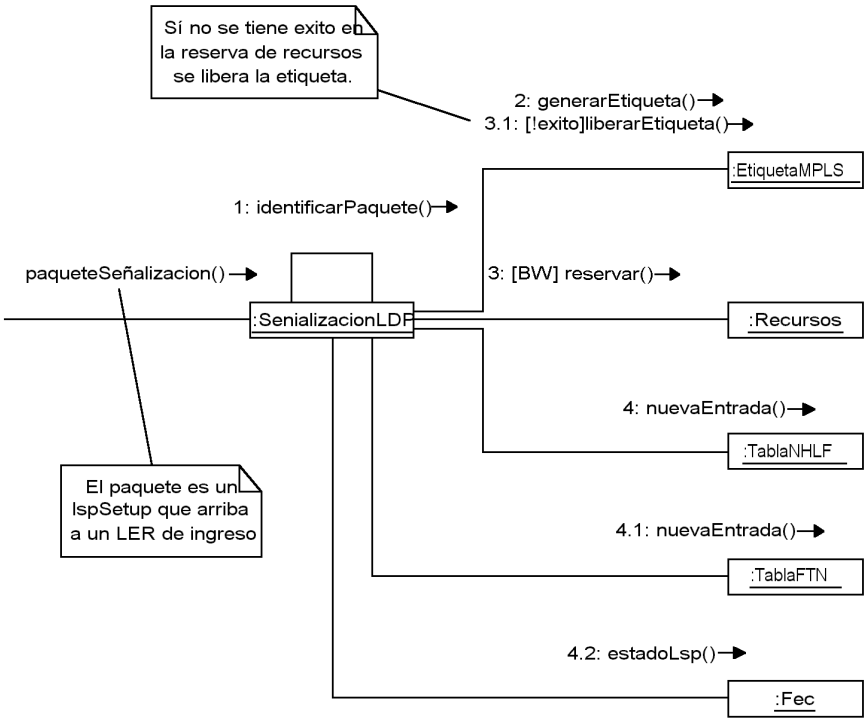


Figura 4.19: Diagrama de colaboración del arribo de un paquete lsp setup a un LER de ingreso

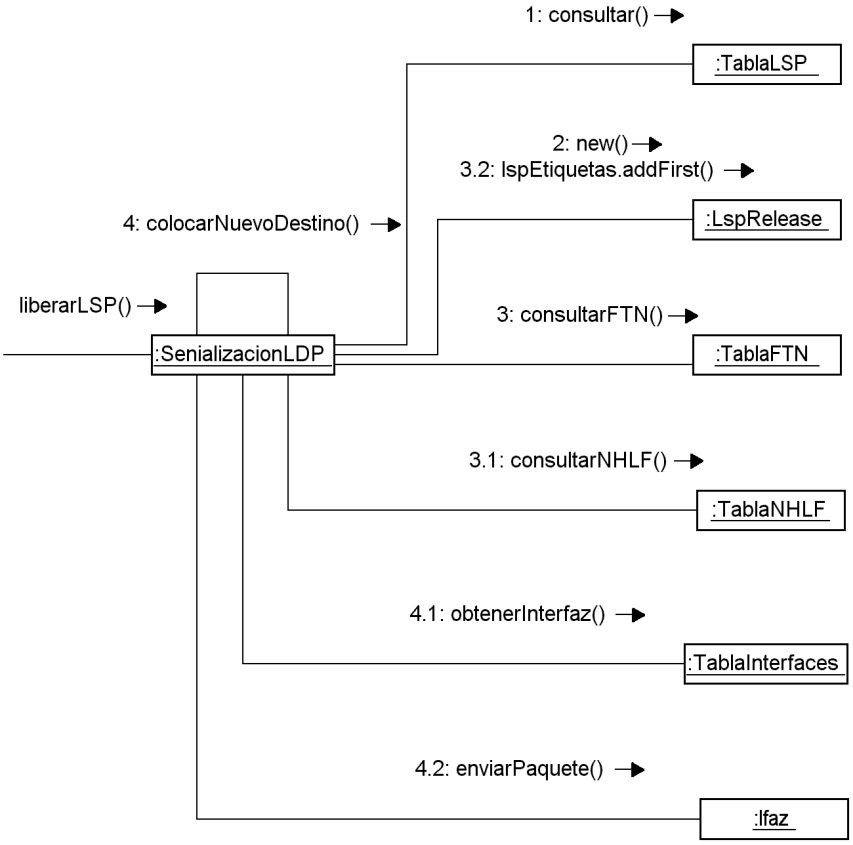


Figura 4.20: Diagrama de colaboración de la creación de un paquete lsp release.

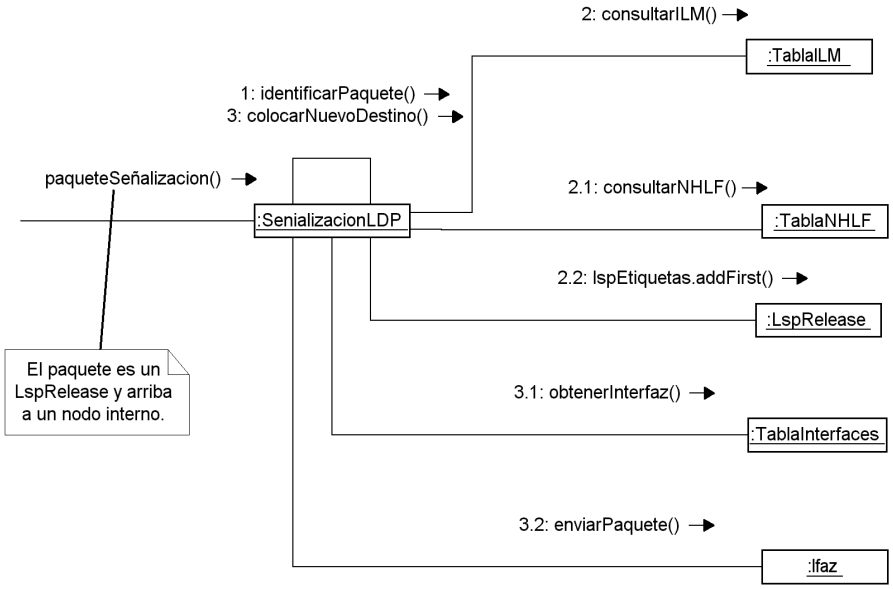


Figura 4.21: Diagrama de colaboración del arribo de un paquete lsp release a un nodo interno de la red.

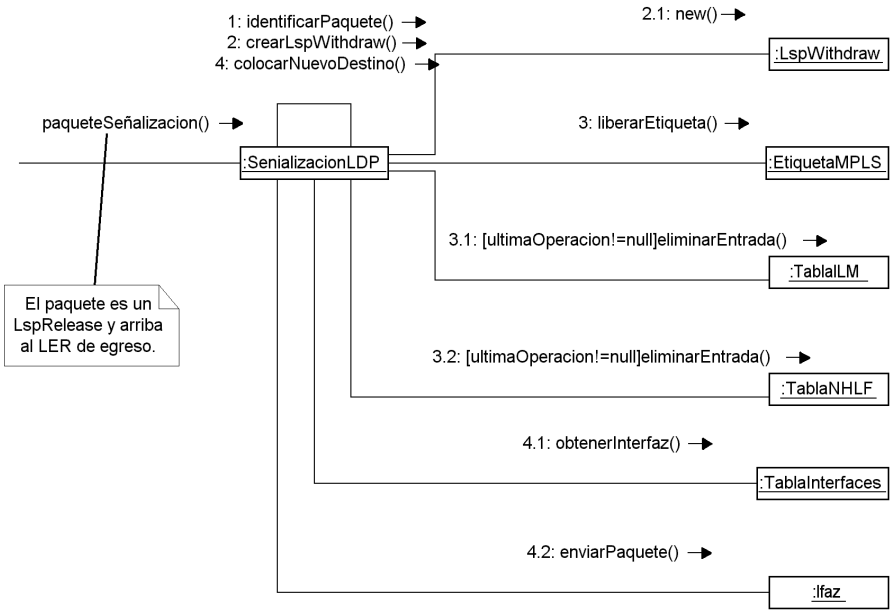


Figura 4.22: Diagrama de colaboración del arribo de un paquete lsp release a un LER de egreso.

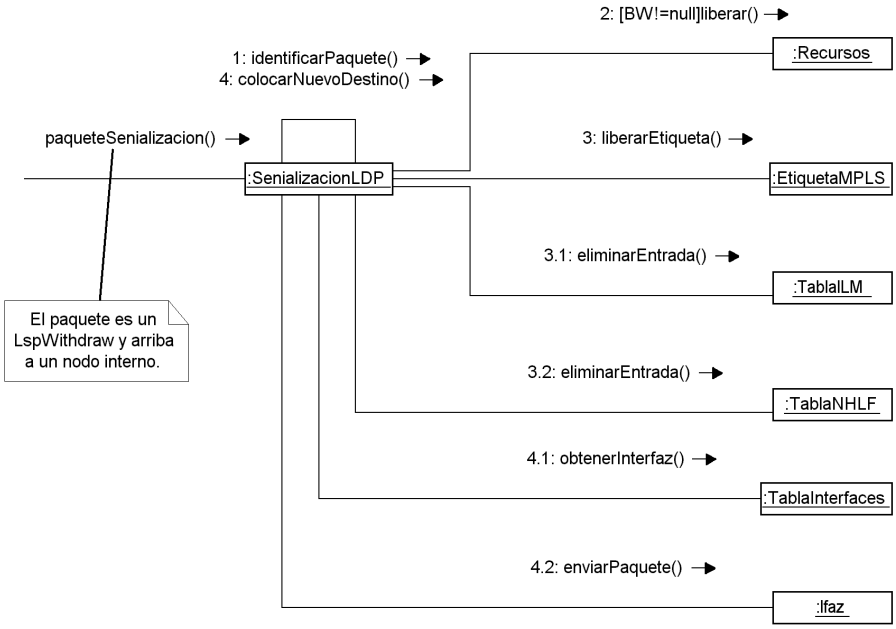


Figura 4.23: Diagrama de colaboración del arribo de un paquete lsp withdraw a un nodo interno.

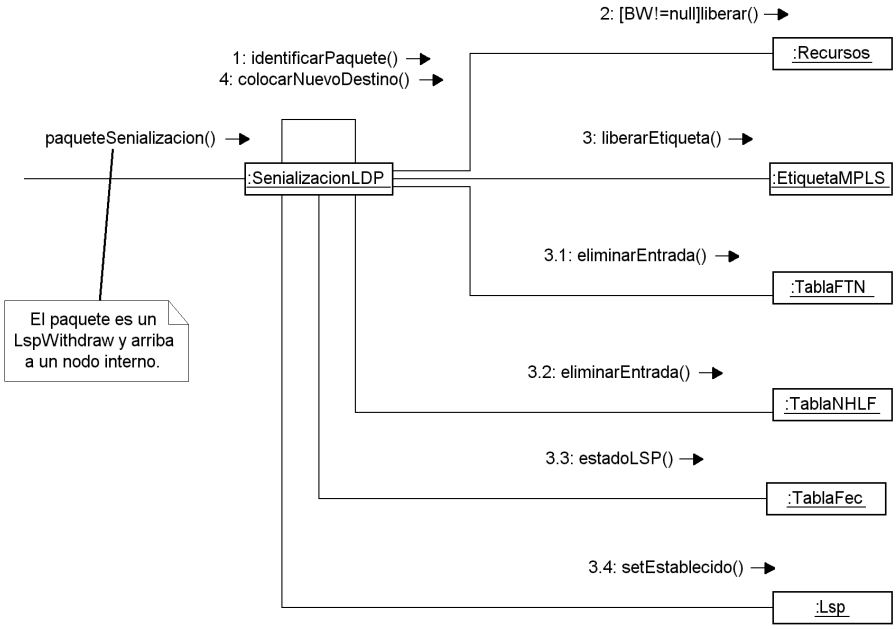


Figura 4.24: Diagrama de colaboración del arribo de un paquete lsp withdraw a un LER de ingreso.

4.4. Descubrimiento

4.4.1. Análisis de requerimientos

Cuando comienza la simulación, cada router de la red MPLS conoce los routers que son accesibles a través de sus interfaces. Empero, a medida que transcurre la simulación, la topología de la red cambia a causa de diversos factores tales como la caída de interfaces de los routers y links.

El módulo de Descubrimiento, tiene dos objetivos fundamentales:

- Aprender los cambios que sufre la red.
- Informar a los sus vecinos los cambios aprendidos.

Para que dichos objetivos sean cumplidos, el módulo Descubrimiento debe ser capaz de crear y procesar crear tres tipos de paquetes:

- paquete Hello.

- paquete LSU.
- paquete LSUACK.

Paquete Hello El objetivo de estos paquetes es el de comunicar al router receptor que el router que envió el paquete está "vivo´´. La información contenida en este paquete es:

- Tiempo de creación.
- Identificación del router que lo creó.

Paquete LSU Estos paquetes serán enviados por los routers a través de cada una de sus interfaces para informar los cambios sucedidos en las mismas. La información contenida dentro de éste paquete es:

- Tiempo de creación en segundos. Es el tiempo de la simulación en que fue creado el paquete.
- Identificación del router que lo creó.
- Identificación de la interfaz que sufrió los cambios.
- Los parámetros de dicha interfaz.
- Interfaz por donde será enviado el paquete.

Paquete LSUACK Se enviará un LSUACK por la interfaz que se haya recibido un paquete LSU. Su objetivo es el de comunicarle al router que envió un LSU que el mismo ha sido recibido. Estos paquetes tienen los siguientes campos:

- Identificación del router que lo creó.
- Interfaz por la que se enviará el paquete.
- El tiempo de creación el LSU por el cual se creó el LSUACK.

El módulo Descubrimiento tendrá que ser capaz de: generar cada uno de estos paquetes, introduciendo la información requerida por cada uno y procesarlos de manera de actualizar la información de la red en el router.

Generación de un paquete Hello El módulo Descubrimiento será capaz de mandar un paquete Hello a todas las interfaces del router cada vez que el módulo Unidad de Control (UC) lo desee. Una vez enviados los paquetes se le debe indicar al módulo UC que los paquetes ya han sido enviados.

Generación de un paquete LSU Cuando se requiera, el módulo Descubrimiento deberá generar un paquete LSU y enviarlo por todas las interfaces. Para ello se le debe indicar la interfaz cuyos datos se quieran difundir. Una vez enviado el paquete LSU por la interfaz debida, se le debe informar a UC dicha acción.

Generación de un paquete LSUACK Cada vez que un router reciba un LSU, enviará un LSUACK por la interfaz que recibió el mismo. La información del tiempo de creación del LSU se obtendrá del paquete LSU recibido.

Procesamiento de un paquete Hello Al llegar un paquete Hello se debe obtener la información del router que lo creó para avisar al módulo a UC quien mandó el paquete Hello . En caso que la interfaz del mismo esté considerada como caída, se levanta y se envía un paquete LSU por las demás interfaces para informarle a los demás routers los cambios sucedidos.

Procesamiento de un paquete LSU Al llegar un paquete LSU, se envía un LSUACK de respuesta. Luego se verifica el tiempo de creación y el origen del mismo. En caso que el paquete no haya sido creado antes del último paquete LSU que provino del mismo origen, se actualizan los datos del router y se envía el paquete LSU por todas las interfaces del router menos por la que arribó el mismo.

Procesamiento de un paquete LSUACK Al recibir un paquete LSUACK, el módulo Descubrimiento le informa al módulo UC la llegada del mismo junto con su tiempo de creación y la interfaz por la que llegó.

Casos de uso

Informar el estado del router.

Caso de uso: Informar el estado del router.

Actor: Módulo UC.

Tabla 4.9: Caso de uso informar el estado del router

Acción del actor	**Acción del sistema**
1. Pasa el intervalo de tiempo $HelloInterval$, por lo tanto se le avisa el módulo Descubrimiento que debe informar su estado a los vecinos.	

	2. Se crea un paquete Hello por cada interfaz que tenga el router. Cada paquete contiene la identificación del router que lo creó y el tiempo de creación del mismo. 3. Cada paquete creado es enviado por cada una de las interfaces del router. 4. Se le informa al módulo UC que nuevamente debe informar el estado del router pasado un tiempo $HelloInterval$.
5. Inicia un contador para avisarle al módulo Descubrimiento que debe informar su estado pasado un tiempo $HelloInterval$.	

Recepción de un paquete Hello.

Caso de uso: Recepción de un paquete Hello.

Actor: Clasificador Control.

Tabla 4.10: Caso de uso Recepción de un paquete Hello

Acción del actor	**Acción del sistema**
1. Al clasificar el paquete de control y verificar que es un paquete Hello, el paquete es enviado al módulo Descubrimiento.	
	2. Se obtiene la información del router vecino que creó el paquete. Si la interfaz que los conecta se considera caída, se actualiza la información de la topología, y se difunde dicho cambio a todos los routers de la red. La difusión se realiza a través de la generación de paquetes LSU. 3. Se le informa al módulo UC que debe avisar que se debe considerar el router vecino como caído en k HelloInterval.

4. Inicia un contador para avisarle al módulo Descubrimiento que debe considerar al vecino caído pasado un tiempo $HelloInterval$.	

Difundir Información.

Caso de uso: Difundir Información.

Actor: Módulo del Router.

Tabla 4.11: Caso de uso Difundir Información

Acción del actor	**Acción del sistema**
1. En el momento en que algún módulo del router detecte un cambio en alguna de las interfaces del router o en el estado de alguno de sus vecinos, se le comunica al módulo Descubrimiento dicho cambio.	2. Se verifican que routers vecinos son accesibles a través de sus interfaces. 3. Se crea un paquete LSU para cada destino accesible. Cada paquete contendrá la información necesaria para comunicarle al router que reciba el paquete los cambios acontecidos. 4. Cada paquete es enviado a las interfaces del router que corresponda. 5. Se le informa al módulo UC que le indique el momento en que debe reenviar el paquete LSU.

Recepción de un paquete LSU.

Caso de uso: Recepción de un paquete LSU.

Actor: Clasificador Control.

Tabla 4.12: Caso de uso Recepción de un paquete LSU

Acción del actor	**Acción del sistema**

1. Al clasificar el paquete de control y verificar que es un paquete LSU, el paquete es enviado al módulo Descubrimiento.	2. Se obtiene el origen y el tiempo de creación del paquete. 3. En caso que el último paquete recibido del router que mandó el paquete haya sido creado antes que el paquete en cuestión, se actualizan los datos de la tabla de recursos con los datos del paquete LSU. 4. Se crea un paquete LSUACK, y se envía por la interfaz por la que ingresó el LSU. 5. Se envía el paquete LSU por todas las interfaces del router menos por la que ingresó.

Reenvío de un paquete LSU.

Caso de uso: Reenvío de un paquete LSU.

Actor: Módulo UC.

Tabla 4.13: Caso de uso Reenvío de un paquete LSU

Acción del actor	**Acción del sistema**
1. Le avisa al módulo Descubrimiento que debe reenviar un paquete LSU.	2. Se verifica que no se haya recibido un paquete LSUACK reconociendo al paquete LSU próximo a reenviar. 3. En caso que no se haya recibido se reenvia. En caso contrario no. 4. Se le informa al módulo UC que le indique el momento en que debe reenviar el paquete LSU.

Recepción de un paquete LSUACK.

Caso de uso: Recepción de un paquete LSUACK.

Actor: Clasificador Control.

Tabla 4.14: Caso de uso Recepción de un paquete LSUACK

Acción del actor	**Acción del sistema**
1. Al clasificar el paquete de control y verificar que es un paquete LSUACK, el paquete es enviado al módulo Descubrimiento.	
	2. Se verifica a que router corresponde el paquete y el tiempo en que fue creado el paquete LSU por el que se está respondiendo.
	3. Se le informa al módulo UC los datos necesarios del paquete para que deje de reenviar los paquetes LSU correspondientes.

Considerar un router vecino caído.

Caso de uso: Tirar Router.

Actor: Módulo UC.

Tabla 4.15: Caso de uso Considerar un router vecino caído

Acción del actor	**Acción del sistema**
1. Se informa que uno de los contadores destinados a controlar los vecinos caídos llegó a un tiempo $kHelloInterval$.	
	2. Se actualiza la topología conocida.
	3. En caso que se cambie algo, se difunde la información correspondiente a la interfaz que conecta ambos routers.

Modelo de dominio

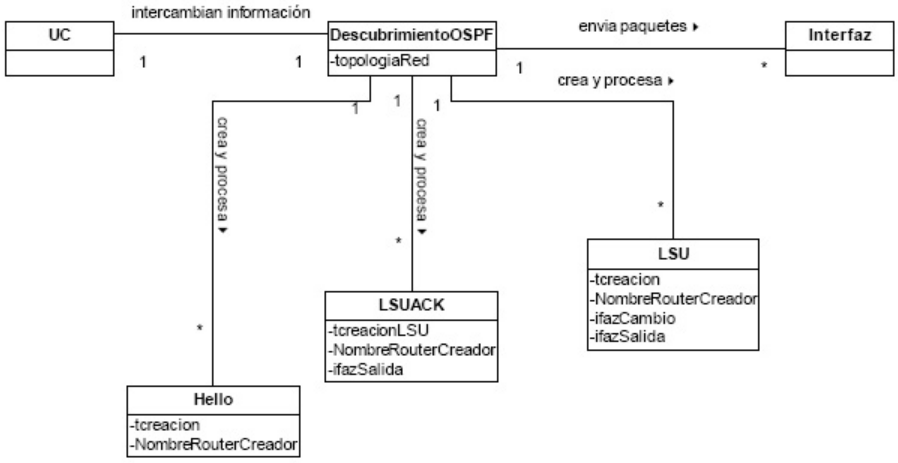


Figura 4.25: Modelo de Dominio del módulo Descubrimiento

4.4.2. Diseño

Diagrama de Clases

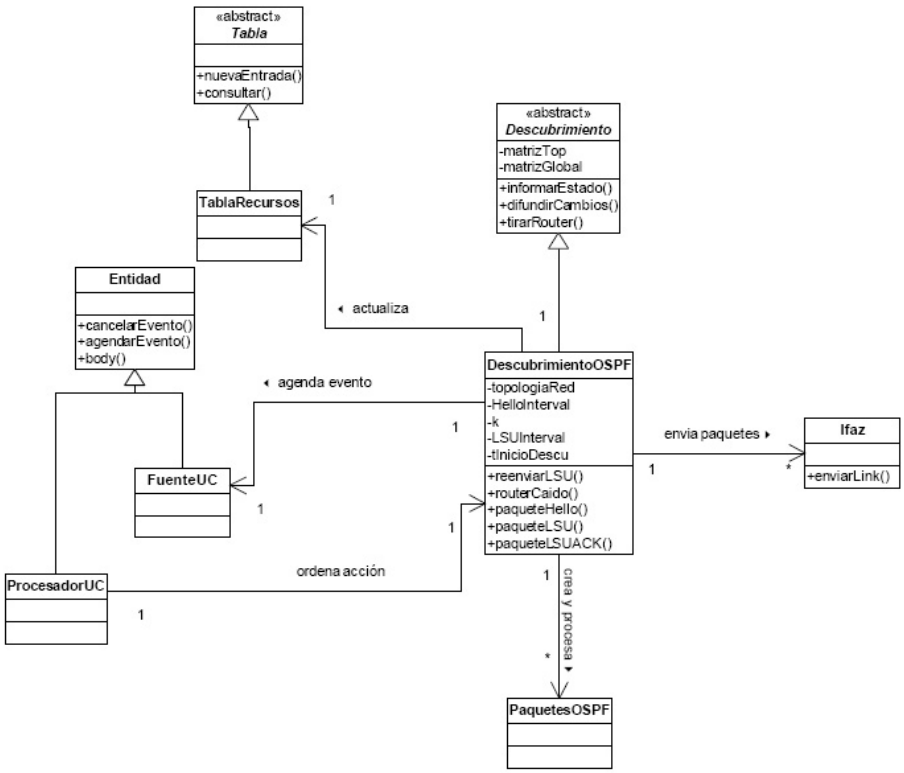


Figura 4.26: Diagrama de Clases del módulo Descubrimiento

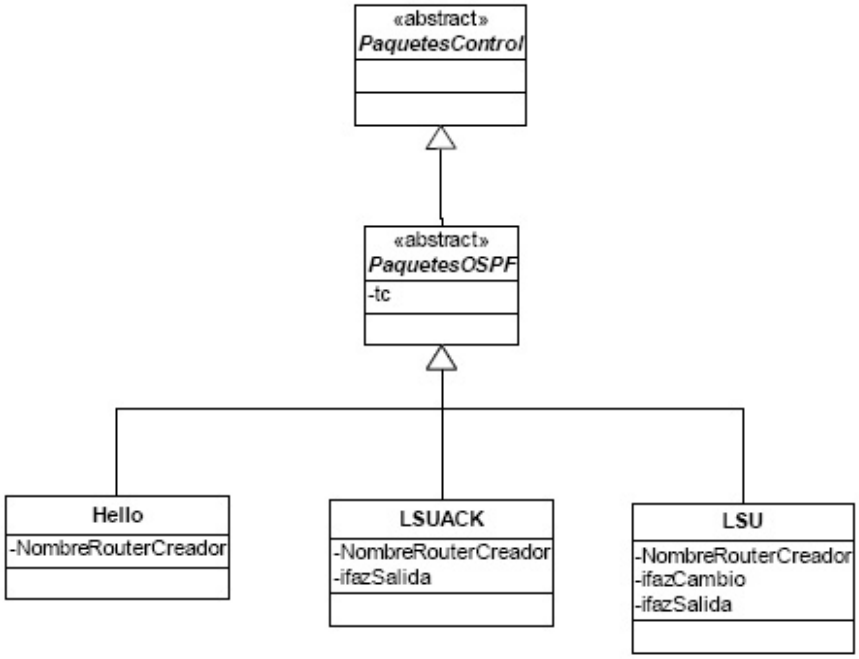


Figura 4.27: Diagrama de Clases de los paquetes usados por el módulo Descubrimiento

Diagramas de Colaboración

Informar el estado del router

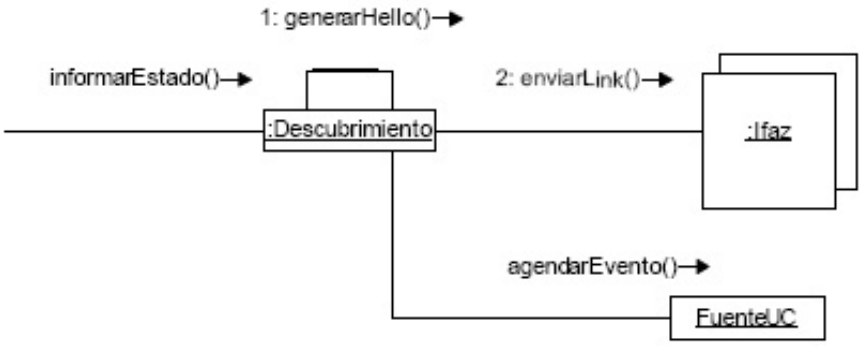


Figura 4.28: Diagrama de colaboración informar estado del router

Recepción de un paquete de control del módulo Descubrimiento

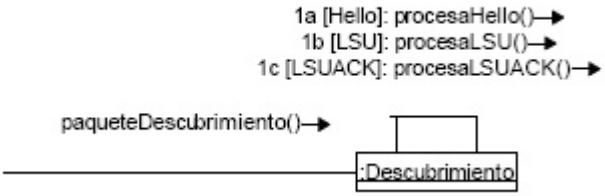


Figura 4.29: Diagrama de colaboración recepción de un paquete Descudrimiento

Recepción de un paquete Hello

Figura 4.30: Diagrama de colaboración del procesamiento de un paquete Hello

Recepción de un paquete LSU

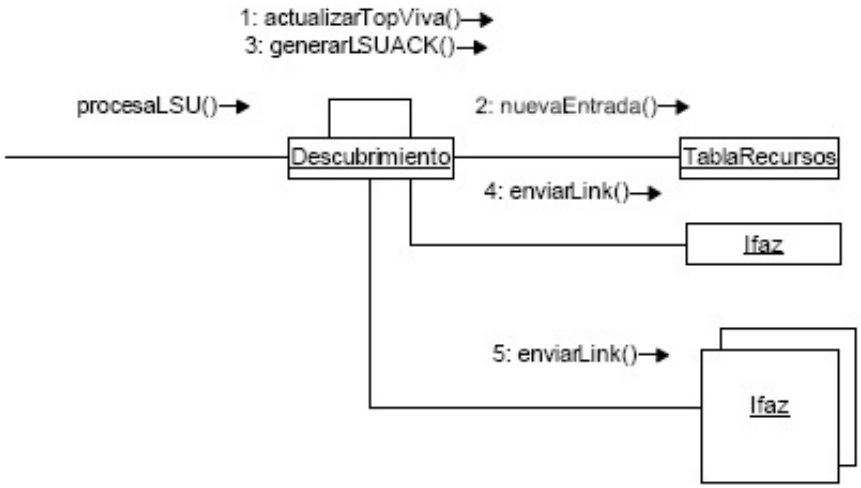


Figura 4.31: Diagrama de colaboración del procesamiento de un paquete LSU

Recepción de un paquete LSUACK

Figura 4.32: Diagrama de colaboración del procesamiento de un paquete LSUACK

Difundir Información.

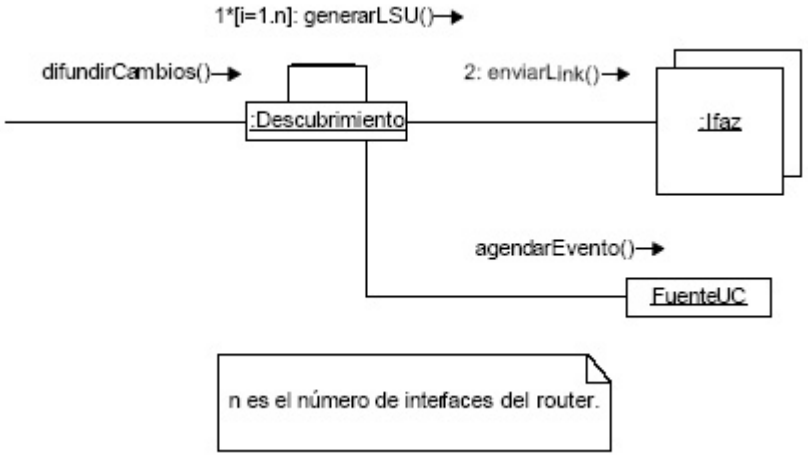


Figura 4.33: Diagrama de colaboración correspondiente a difundir cambios

Reenvío de un paquete LSU.

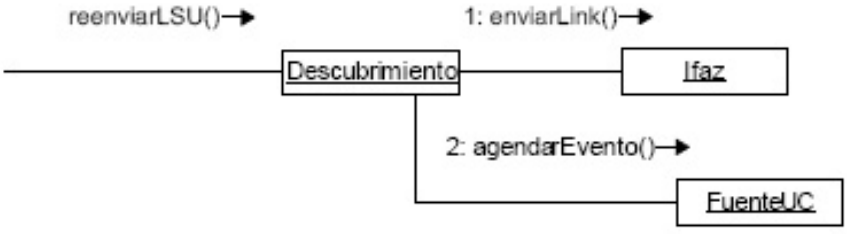


Figura 4.34: Diagrama de colaboración para reenviar paquetes LSU

Considerar un router vecino caído.

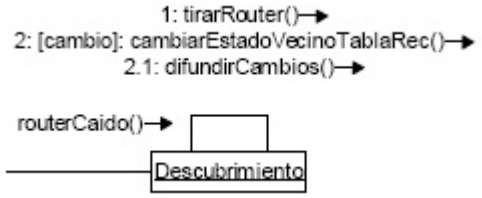


Figura 4.35: Diagrama de colaboración para considerar un router vecino caído

4.4.3. Implementación

La forma en que cada router realiza el descubrimiento de la red está basado en el protocolo OSPF. En caso que se desee descubrir los distintos nodos de la red usando otro protocolo, solo basta con crear una clase que herede de Descubrimiento y que implemente los métodos públicos que tiene.

La información de la topología de la red está contenida en una matriz de dimensión nxn, donde n es el número de routers. Dicha matriz se llama `matrizTop`.

La figura 4.36(a) muestra una red la cual se representa matricialmente según la tabla 4.36(b).

En cada elemento de la fila i se indica la accesibilidad del router Ri a los demás a través de alguna de sus interfaces. 1 indica que el router es accesible y -1 que no lo es.

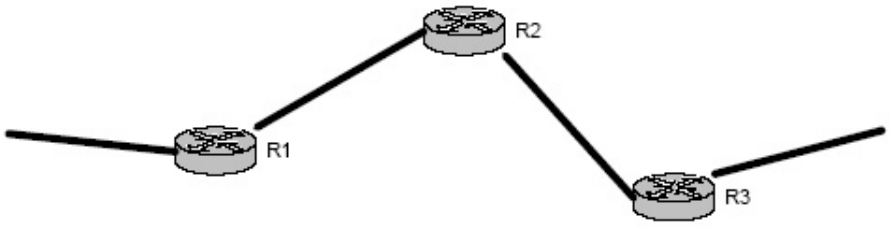
Así en la tabla 4.36(b), en la primer fila se indican los routers a los que R1 accede a través de alguna de sus interfaces. En este caso el único router accesible es R2.

4.5. Unidad de Control

4.5.1. Análisis de requerimientos

El objetivo del módulo UC es el de controlar las acciones de los routers. Cada módulo del router puede agendar distintas acciones o eventos a efectuarse en un tiempo especificado por los mismos. Cuando llega el momento de efectuarse alguna de las acciones agendadas, el módulo UC se encarga de avisarle al módulo que corresponda.

En ESR, los módulos con los que el módulo UC tiene relación son: Descubrimiento (ver 4.4), Rutas (ver 4.6) y Señalización (ver 4.3).



(a) Topología de la red

	R1	R2	R3
R1	-1	1	-1
R2	1	-1	1
R3	-1	1	-1

(b) Matriz que representa la topología

Figura 4.36: En establecimiento, mantenimiento y liberación

Relación con el módulo Descubrimiento Cuando inicia la simulación, el módulo Descubrimiento le informa al módulo UC que debe comenzar a informar el estado del router a partir de un determinado tiempo t_{0Desc} y repetir dicha acción cada un intervalo de tiempo $HelloInterval$. Tanto t_{0Desc} como $HelloInterval$ pueden ser especificados por el usuario.

Para ésto es que el módulo UC inicia un contador que a partir del tiempo t_{0Desc} le avisa al módulo Descubrimiento que debe informar el estado del router.

También se inicializa un contador por cada vecino del router para que a partir de t_{0Desc}, pasado un tiempo $k\,HelloInterval$ sin que el módulo Descubrimiento le avise que se recibió un paquete Hello de algún vecino, la interfaz que los une se considere caída. k es un número entero mayor a 1 que puede ser especificado por el usuario.

Al inicio de la simulación también se envía a los routers vecinos paquetes LSU con la información de todas sus interfaces.

Una vez que el módulo Descubrimiento envía dicho paquete, la UC se encarga de que cada paquete LSU sea reenviado hasta que se reciba un LSUACK que corresponda al paquete LSU. El reenvío se realiza pasado un tiempo $LSUInterval$.

Relación con el módulo Señalización En el caso que se haya configurado el router para que tenga señalización LDP, cuando comienza la simulación, se agendan los eventos de establecimiento y liberación de los LSP en la UC. Es decir se agendan los tiempos de establecimiento y liberación de cada LSP configurado por el usuario. Cuando llega el momento del establecimiento o liberación de un LSP la UC avisa al módulo de Senialzacion el correspondiente hecho.

En el caso en que no se pueda establecer un LSP por una falla en el proceso de establecimiento, el módulo de Senializacion agenda un tiempo de persistencia en el que la UC debe volver a avisar al módulo de Senialización que debe reintentar establecer el LSP.

Relación con el módulo Rutas Cuando comienza la simulación, el módulo Rutas le informa al módulo UC el tiempo en que se debe comenzar a actualizar las tablas de ruteo ($t_{0tablaRuteo}$) y el intervalo que debe pasar entre dos actualizaciones sucesivas ($\Delta t_{0tablaRuteo}$).

Cuando la simulación alcanza el tiempo $t_{0tablaRuteo}$, el módulo UC le empieza a avisar al módulo Rutas cada $\Delta t_{0tablaRuteo}$ que se debe actualizar la tabla de ruteo del router .

Casos de uso

Agendar evento para informar el estado del router

Caso de uso: Agendar evento para informar el estado del router.

Actor: Módulo Descubrimiento.

Tabla 4.16: Caso de uso Agendar evento para informar el estado del router

Acción del actor	**Acción del sistema**
1. Le informa al módulo UC el tiempo a partir del cual se debe comenzar a informar el estado y el intervalo de tiempo en que se debe realizar dicha acción.	
	2. Se inicia un contador para avisarle al módulo Descubrimiento cuando llegue el momento de informar el estado del router.

	3. Se inicia un contador por cada interfaz del router para avisarle al módulo Descubrimiento el momento en que se debe considerar como caído un router vecino. Un router vecino se considera caído cuando el contador correspondiente llega a $kHelloInterval$

Agendar evento para avisar cuando se debe difundir la información de las interfaces del router.

Caso de uso: Agendar evento para avisar cuando se debe difundir la información de las interfaces del router.

Actor: Módulo Descubrimiento.

Tabla 4.17: Caso de uso Agendar evento para avisar cuando se debe difundir la información de las interfaces del router

Acción del actor	**Acción del sistema**
1. Le informa al módulo UC el tiempo a partir del cual se debe a difundirla información de sus interfaces.	2. Se inicia un contador para avisarle al módulo Descubrimiento cuando llegue el momento de difundir la información de sus interfaces.

Aviso de recepción de un paquete Hello.

Caso de uso: Aviso de recepción de un paquete Hello.

Actor: Módulo Descubrimiento.

Tabla 4.18: Caso de uso Aviso de recepción de un paquete Hello

Acción del actor	**Acción del sistema**
1. Le informa al módulo UC que debe resetear el contador que avisa cuando el router que mandó el paquete Hello está caído.	2. Resetea dicho contador.

Aviso de recepción de un paquete LSUACK.

Caso de uso: Aviso de recepción de un paquete LSUACK.

Actor: Módulo Descubrimiento.

Tabla 4.19: Caso de uso Aviso de recepción de un paquete LSUACK

Acción del actor	**Acción del sistema**
1. Le informa al módulo UC que se recibió un paquete LSUACK confirmando la recepción de un paquete LSU.	2. Se elimina el contador encargado de avisar al módulo Descubrimiento para que reenvie el paquete LSU en cuestión.

Agendar establecimiento de LSP.

Caso de uso: Agendar establecimiento de LSP.

Actor: Módulo Señalización.

Tabla 4.20: Caso de uso Agendar establecimiento de LSP

Acción del actor	**Acción del sistema**
1. Le informa el momento y el ID en que se deben establecer los LSPs.	2. Inicia un contador por cada LSP. Llegado el momento de establecimiento de cada uno, se le avise al módulo Señalización que LSP se debe establecer.

Aviso para actualizar la tabla de ruteo del router.

Caso de uso: Aviso para actualizar la tabla de ruteo del router.

Actor: Módulo Rutas.

Tabla 4.21: Aviso para actualizar la tabla de ruteo del router.

Acción del actor	**Acción del sistema**
1. Le informa al módulo UC el tiempo a partir del cual se debe comenzar a actualizar la tabla de ruteo del router y el intervalo de tiempo en que se debe realizar dicha acción.	2. Inicia un contador para informar al módulo Rutas el momento en que se debe actualizar la tabla de ruteo del router.

Modelo de dominio

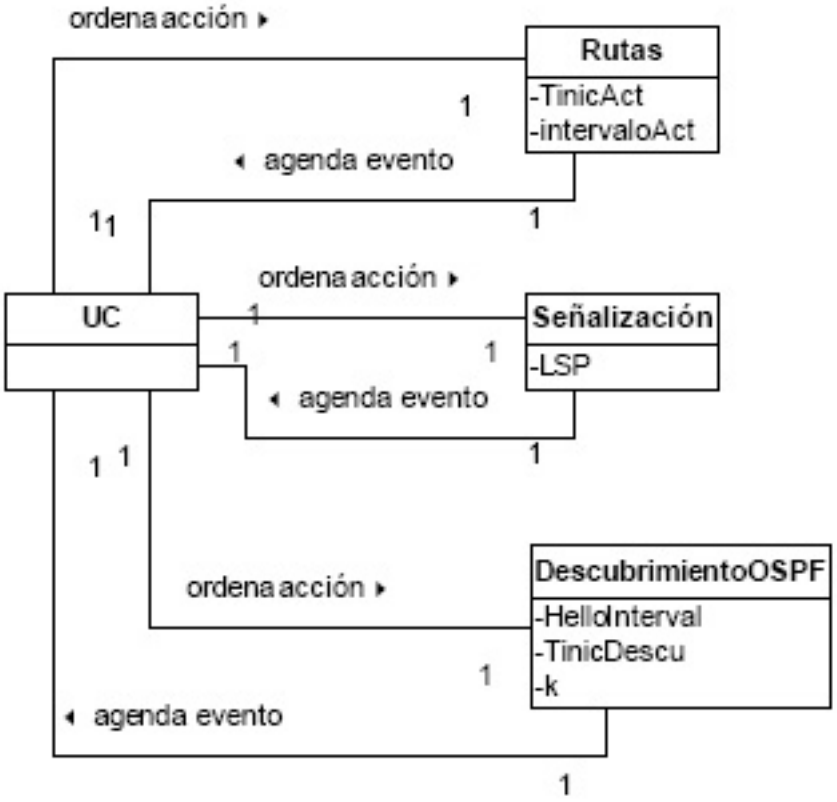


Figura 4.37: Modelo de Dominio del módulo Unidad de Control

4.5.2. Diseño

Diagrama de Clases

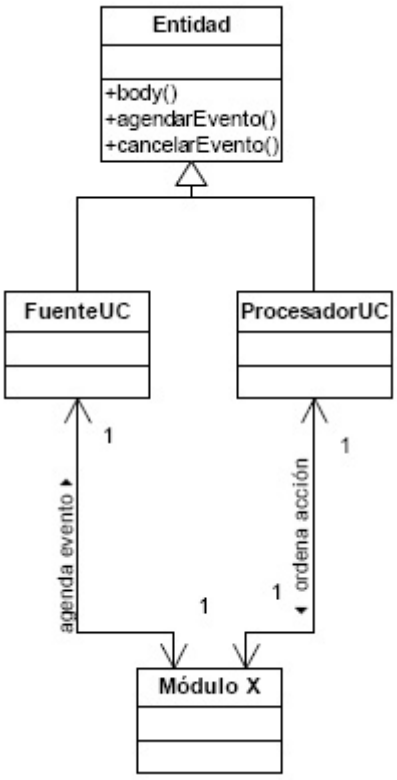


Figura 4.38: Diagrama de Clases del módulo UC

4.5.3. Implementación

Como se pude apreciar, el módulo UC está compuesto por una Fuente que genera los eventos y un Procesador que los procesa. La Fuente y el Procesador se implementan con las clases `FuenteUC` y `ProcesadorUC` respectivamente.

Las entidades `FuenteUC` y `ProcesadorUC`, están unidas a través de puertos `out` e `in` respectivamente.

El método `agendarEventos(double delay, int tag, Object dato)` de la clase `FuenteUC`, es el método que se debe usar para agendar nuevos eventos. Cuando se invoca, se envía al Procesador, el evento representado por un tag del tipo entero, para que sea ejecutado en el tiempo representado por el parámetro `delay`.

Cuando el Procesador recibe un evento, este obtiene el parámetro `tag` para usarlo en una estructura `switch case`, para que ejecute la acción debida.

Dada la importancia del parámetro `tag` para que todo funcione correctamente, para agregar nuevos eventos se debe declarar en la clase `FuenteUC` un atributo público del tipo `int` y constante que identifique el nuevo evento. En la clase `ProcesadorUC` se debe de agregar el correspondiente case con el identificador del nuevo evento dentro del método body() el cual define el comportamiento de la entidad.

Se reservaron rangos de números enteros para los eventos de cada módulo. Ver la

Tabla 4.22: Rangos de tags para cada módulo del Router.

Rango	**Módulo**
0 a 19	sin usar
20 a 29	Senializacion.
30 a 39	Descubrimiento.
40 a 49	Rutas.

4.6. Rutas

4.6.1. Análisis de requerimientos

Éste módulo le brinda al router la posibilidad de:

- calcular la ruta entre dos routers de la red.
- actualizar la tabla de ruteo del router.

Cálculo de ruta entre dos routers El cálculo de una ruta entre dos routers se puede realizar con y sin restricciones de BW. Se basa en el algoritmo de Dijsktra y se pueden usar las siguientes métricas:

- $MinimumHopRouting$,el algoritmo buscará el camino con la mínima cantidad de saltos.
- $MinimumAdministrativeWeightRouting$, en la cual la ruta resultante es aquella en la cual la suma de los pesos de los enlaces es menor.
- $\frac{1}{BWreservado}$, métrica cuyos pesos se basan en el inverso del BW reservado. Así el algoritmo elegirá aquel camino cuyos enlaces estén más usados.

- $\frac{1}{BW disponible}$, es una métrica opuesta a la anterior. El uso de está métrica elige aquel camino cuyos enlaces estén menos usados.

En ESR el único módulo que usa el cálculo de rutas con o sin restricciones es el módulo Señalización.

Actualización de la tabla de ruteo de un router En base a la métrica definida por el usuario para el cálculo de la tabla de ruteo del router, se le aplica el algoritmo de Dijsktra a la topología de la red conocida por el router para calcular la ruta a todos los nodos de la red.

El módulo UC es el encargado se avisar al módulo Rutas que se debe actualizar la tabla de ruteo.

Casos de uso

Cálculo de ruta entre dos routers sin restricciones de BW

Caso de uso: Cálculo de ruta entre dos routers sin restricciones de BW.

Actor: Módulo Señalización.

Tabla 4.23: Caso de uso Cálculo de ruta entre dos routers sin restricciones de BW

Acción del actor	**Acción del sistema**
1. Le pide calcular una ruta desde un origen a un destino con una métrica en particular.	
	2. Calcula la ruta aplicando el algoritmo de Dijsktra a la topología de la red conocida del router usando la métrica especificada.
	3. Entrega la ruta.

Cálculo de ruta entre dos routers con restricciones de BW

Caso de uso: Cálculo de ruta entre dos routers con restricciones de BW.

Actor: Módulo Señalización.

Tabla 4.24: Caso de uso Cálculo de ruta entre dos routers con restricciones de BW

Acción del actor	**Acción del sistema**
1. Le pide calcular una ruta desde un origen a un destino con una métrica en particular y un BW disponible en los enlaces mínimo.	
	2. Se eliminan todos aquellos enlaces con BW insuficiente.
	3. Calcula la ruta aplicando el algoritmo de Dijsktra a la topología de la red conocida del router usando la métrica especificada.
	4. Entrega la ruta.

Actualización de la tabla de ruteo del router

Caso de uso: Actualización de la tabla de ruteo del router.

Actor: Módulo UC.

Tabla 4.25: Actualización de la tabla de ruteo del router

Acción del actor	**Acción del sistema**
1. Le avisa que se debe actualizar la tabla de ruteo del router1.	
	2. Calcula la ruta mas corta aplicando el algoritmo de Dijsktra a la topología de la red conocida del router en base a la métrica especificada para las tablas de ruteo.
	3. Avisa al módulo UC que se realizó dicha actualización.

Modelo de dominio

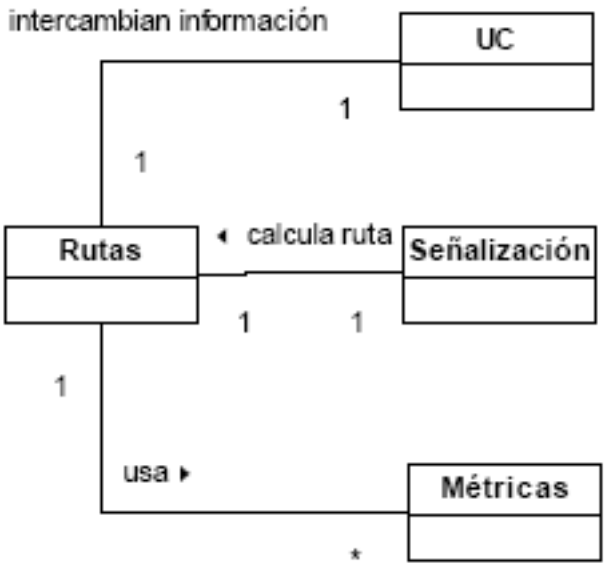


Figura 4.39: Modelo de Dominio del módulo Rutas

4.6.2. Diseño

Diagrama de Clases

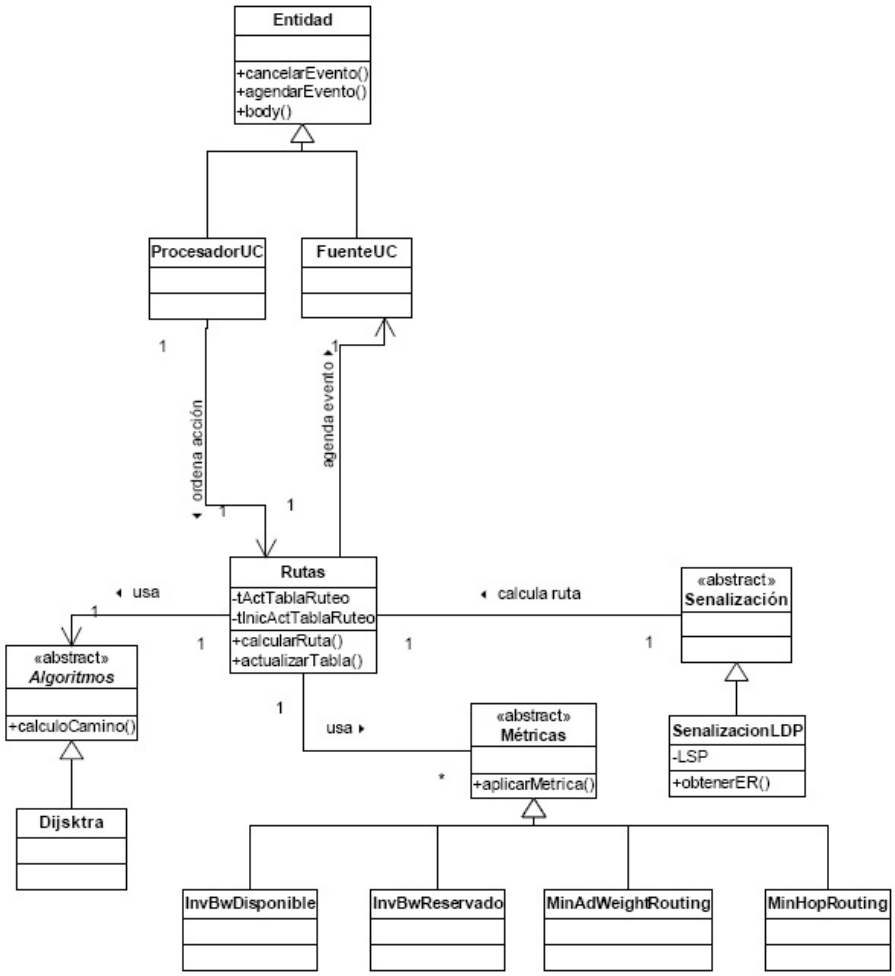


Figura 4.40: Diagrama de Clases del módulo Rutas

Diagramas de Colaboración

Cálculo de ruta entre dos routers sin restricciones de BW

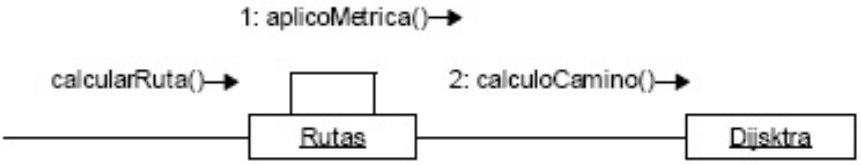


Figura 4.41: Diagrama de colaboración para calcular una ruta sin restricciones

Cálculo de ruta entre dos routers con restricciones de BW

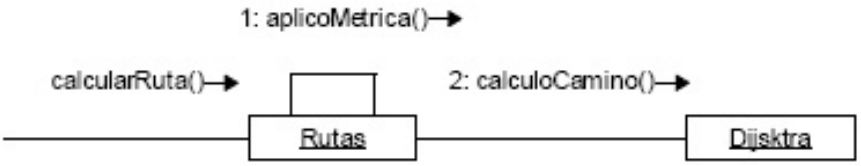


Figura 4.42: Diagrama de colaboración para calcular una ruta con restricciones

Actualización de la tabla de ruteo del router

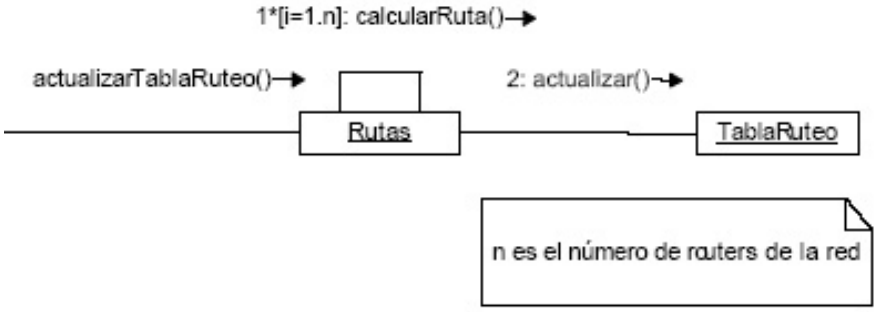


Figura 4.43: Diagrama de colaboración para actualizar la tabla de ruteo del router

4.6.3. Implementación

El módulo rutas es usado por los routers básicamente para calcular la ruta entre dos routers usando una métrica y un algoritmo determinado.

En ésta versión de ESR, el algoritmo usado es Dijsktra. En el caso que se desee usar otro algoritmo, lo único que se debe hacer es crear una clase que herede de la clase `Algritmoy` que implemente el método `calculoCamino( int origen,int destino,double[][]matrizTop,int enlaces))`.

El parámetro `origen` indica donde debe comenzar el camino, `origen` el fin

del camino y el parámetro `matrizTop`

4.7. Recursos

4.7.1. Análisis de requerimientos

Éste módulo brinda la posibilidad a los demás módulos del router a:

- consultar los recursos de BW de una de las interfaces del router.
- reservar recursos de BW de una de las interfaces del router.
- liberar recursos de BW de una de las interfaces del router.

Para realizar una consulta de recursos de BW, se debe especificar la interfaz a la cual se desean consultar los recursos y el ancho de banda requerido. Al consultante se le informa si el $BW_{disponible}$ de la interfaz sea mayor al $BW_{requerido}$ o no.

En ésta versión de ESR, el único módulo capaz de reservar o liberar recursos es el módulo de Senializacion. Para realizar alguna de éstas acciones, es necesario indicar la interfaz a la cual se desean reservar recursos y el LSP que pasará por la misma. Vale aclarar que el LSP contiene la información del BW a reservarse por clase.

Cada vez que se reserve o libere BW de una interfaz, se le informa al módulo Descubrimiento que debe difundir los cambios acontecidos.

Casos de uso

Consulta de recursos de BW disponible

Caso de uso: Consulta de recursos de BW disponible.

Actor: Módulo del router.

Tabla 4.26: Consulta de recursos de BW disponible

Acción del actor	**Acción del sistema**
1. Consulta si hay BW disponible en una interfaz de un router determinado.	

	2. Se fija en la tabla de recursos del router el BW disponible de la interfaz especificada. 3. Informa si hay suficiente BW .

Reserva de recursos

Caso de uso: Reserva de recursos.

Actor: Módulo Señalización.

Tabla 4.27: Reservación de recursos

Acción del actor	**Acción del sistema**
1. Informa la interfaz en la cual se quiere hacer la reserva y el LSP que se establecerá por la misma.	2. Se fija en la tabla de recursos del router el BW disponible de la interfaz especificada. 3. Si hay suficiente BW para el LSP, se reservan recursos. 4. Se le informa al módulo Descubrimiento que han habido cambios en la interfaz. 5. Se le informa al módulo Señalización los resultados de la reserva .

Liberación de recursos

Caso de uso: Liberación de recursos.

Actor: Módulo Señalización.

Tabla 4.28: Liberación de recursos

Acción del actor	**Acción del sistema**
1. Informa la interfaz en la cual se quiere liberar recursos y el LSP que se levantará.	

	2. Se fija en la tabla de recursos del router el BW disponible de la interfaz especificada. 3. En caso que el LSP esté establecido, se liberan los recursos usados por el mismo. 4. Se le informa al módulo Descubrimiento que han habido cambios en la interfaz. 5. Se le informa al módulo Señalización los resultados de la reserva .

Modelo de dominio

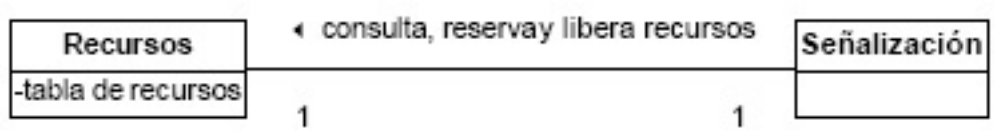


Figura 4.44: Modelo de Dominio del módulo Recursos

4.7.2. Diseño

Diagrama de Clases

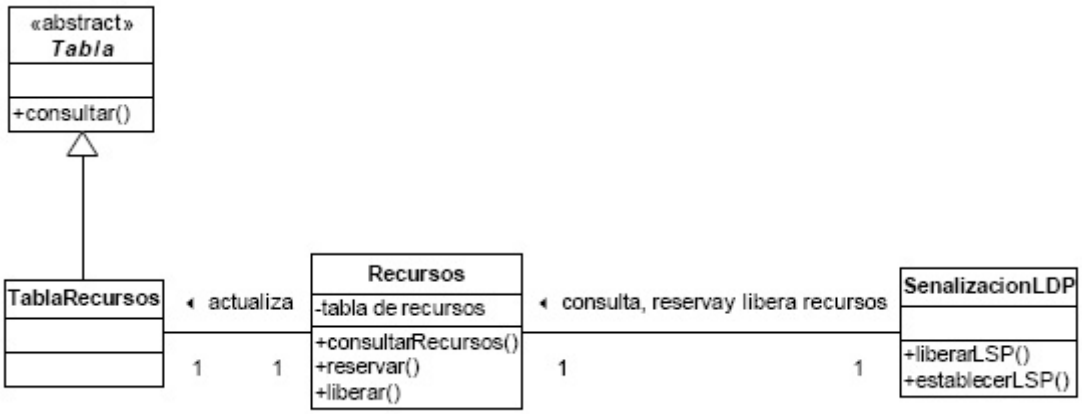


Figura 4.45: Diagrama de Clases del módulo Recursos

Diagramas de Colaboración

Consulta de recursos de BW disponible

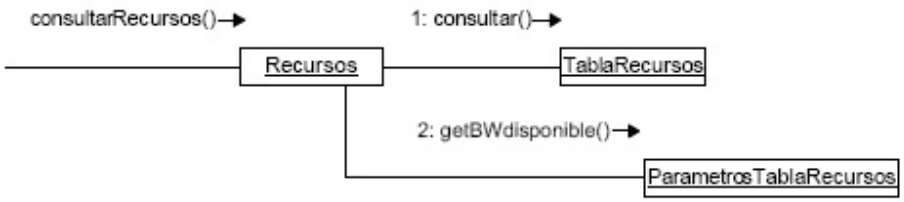


Figura 4.46: Diagrama de colaboración para consultar los recursos

Reserva de recursos

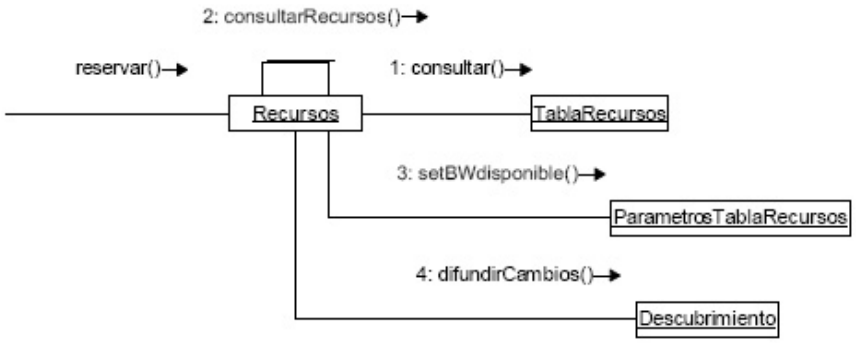


Figura 4.47: Diagrama de colaboración para reservar recursos de una interfaz

Liberación de recursos

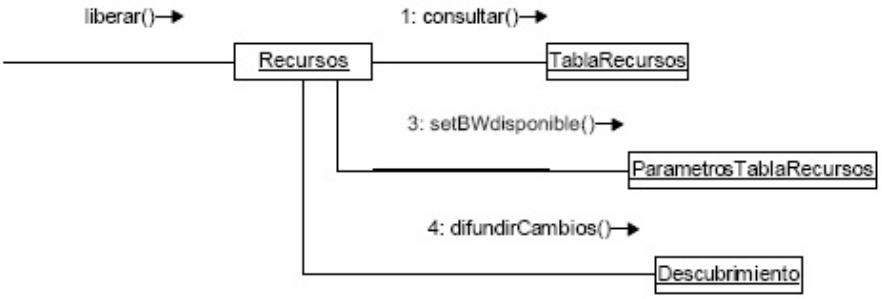


Figura 4.48: Diagrama de colaboración para liberar recursos de una interfaz

Capítulo 5

Configuración

5.1. Análisis de requerimientos

Una simulación tiene una red la cual esta compuesta por nodos y links. Los nodos que existen en esta versión del simulador son Router y Host. Los routers se subclasifican en LER y LSR. Todos los nodos de la red tienen al menos una interfaz de entrada/salida, clasificadores, módulos de control. Los módulos a diferencia de los demás componentes del nodo, pueden estar ó no y de estos depende la funcionalidad del router. Además los módulos pueden tener que consultar y crear tablas las cuales también pueden compartirse entre los diferentes módulos. Los nodos de la red son independientes entre sí, es decir, la configuración de los nodos se realiza de manera individual, sin afectar los parámetros del otro nodo. Así también, todos los parámetros se deben inicializar en algún valor por defecto, de manera de simplificar la configuración de los mismos.

La configuración de una simulación es parte esencial del presente simulador, mediante la cual se arma la topología de la red, se configuran los routers, host y link de la red,los parámetros de la simulación y se agendan acciones.

La función de configuración más importante, es la creación de los componentes que forman la red, es decir la creación de los hosts, routers y links. A su vez, dada la arquitectura adoptada de los nodos de la red, [1] requiere que se creen los módulos, interfaz, clasificadores y demás componentes, los que luego podrán ser configurados.

Entre los parámetros a configurar se encuentran las variables temporales usadas en los protocolos, propiedades de alguna parte de la red, como por ejemplo ancho de banda de los links. También se encuentran comprendidos los parámetros de la simulación, como el tiempo de duración de la misma. Entre las acciones que se pueden configurar se encuentran el establecimiento de LSP, caídas de link,

[1] La arquitectura de los nodos se detalla en el capítulo **?? ??**

etc. Cuando se configura algún elemento de la red, es posible que no sea necesario establecer el valor de todos sus parámetros, por lo tanto, debe ser posible solo configurar aquellos parámetros que el usuario necesite y los demás queden establecidos a valores definidos por defecto.

En ESR, esta tarea se debe realizar de manera que sea posible configurar desde cualquier tipo de interfaz gráfica que interactúe con el usuario, como ser mediante la escritura de comandos predefinidos en un archivo de texto, gráficamente, desde una consola ó de donde la imaginación del programador alcance. Por otra parte también debe ser posible realizar la configuración de una simulación mediante una clase de java.

Se deben de poder configurar los parámetros de la simulación. Éstos parámetros no tienen ninguna relación con los parámetros de una red de datos. En principio el parámetro que no debe faltar es el tiempo de simulación, el cual se establece como condición de parada de la simulación.

La red se configura indicando las interconexiones entre los nodos. Las conexiones posibles son router-router, router-host y host-host.

En un host, se debe configurar la interfaz, indicando algunos o todos sus parámetros, los agentes y los generadores. En esta versión del simulador, el único agente que existe es el UDP, pero hay que tener en cuenta que podrían existir otros como TCP. En un host pueden existir múltiples generadores, cada uno conectado a un sólo agente.

En un router, la cantidad de elementos a configurar es mayor que en el host y puede variar mucho más. Al igual que en el host, se deben configurar las interfaces, todas de manera independiente. También se deben configurar los módulos que utilizan tablas igualmente configurables. Hay módulos que siempre tienen que estar presentes y otros que pueden estar ó no presente. Esto hace que los elementos a configurar de un router sean variables.

La configuración de los links, se basa en establecer los parámetros del mismo, tales como el ancho de banda, retardo, probabilidad de pérdidas y tantos como el modelo del link defina.

5.1.1. Casos de uso

Configuración de la simulación

Caso de uso: Configuración de la simulación

Actor: Usuario, Configurador.

Tabla 5.1: Caso de uso, configuración de la simulación

Acción del actor	**Acción del sistema**

1. El usuario ingresa la configuración de la simulación y da la orden para que comience la misma.	
	2. El sistema toma la configuración y la procesa pasándole los datos al Configurador
3. El Configurador arma la red, crea los nodos, configura los parámetros de la simulación, de los nodos y establece las acciones.	
	4. Cuando finaliza la configuración sin errores, el sistema da la orden para que comience la simulación.
4a. En caso de que se encuentren errores en la configuración se detiene el proceso y se avisa al usuario lo ocurrido, para que este tome alguna acción correctiva y comience nuevamente el proceso de configuración.	

5.1.2. Modelo de dominio

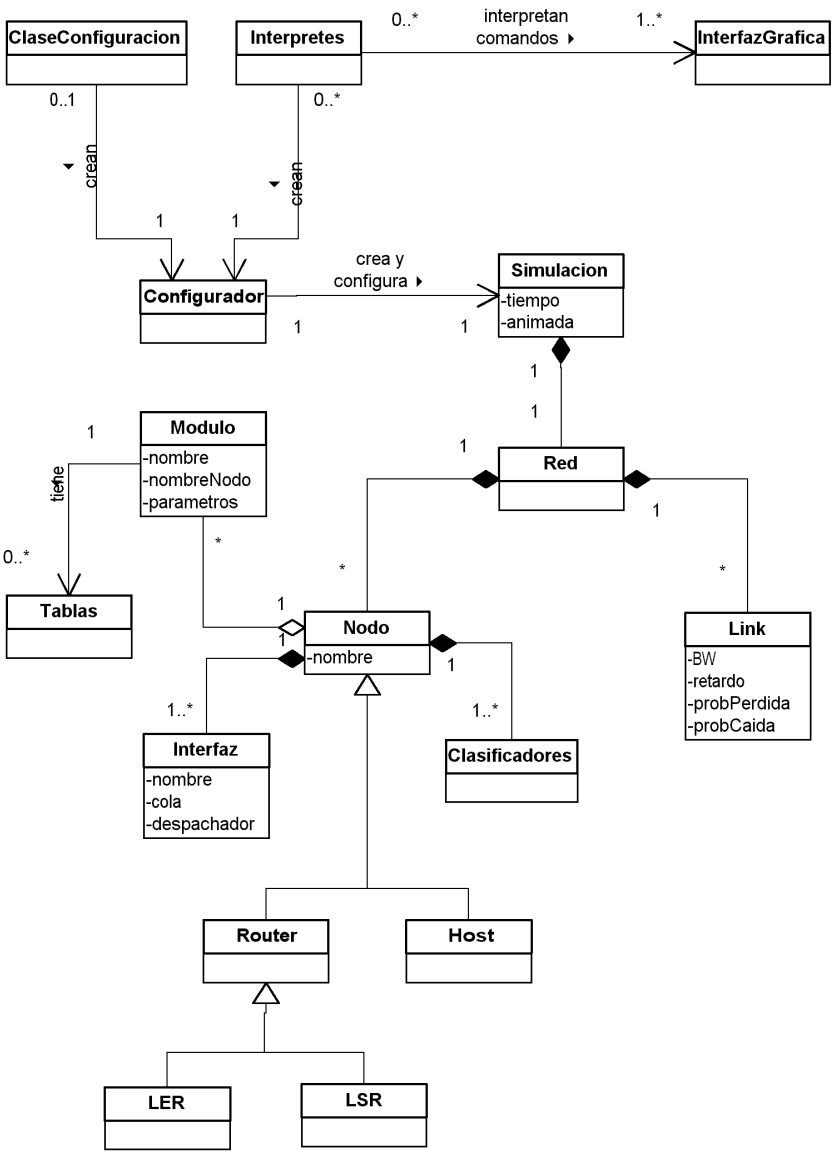


Figura 5.1: Diagrama de dominio de la configuración de la simulación

5.2. Diseño

5.2.1. Diagrama de Clases

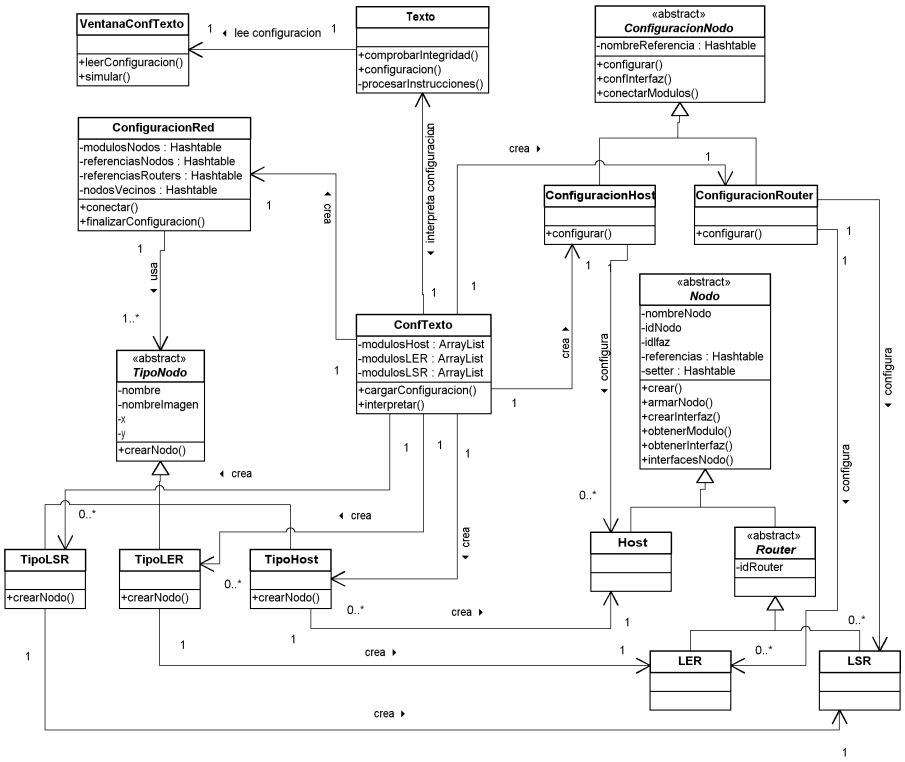


Figura 5.2: Diagrama de clase de la configuración de la simulación

5.2.2. Diagramas de Colaboración

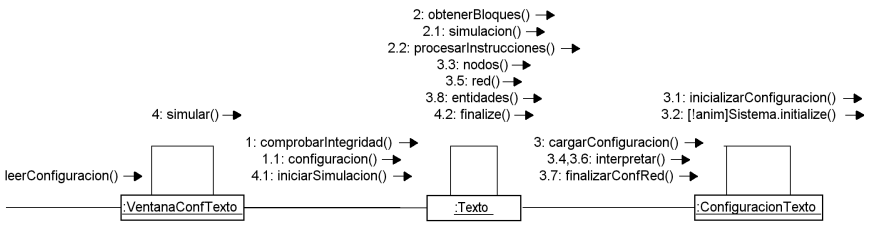


Figura 5.3: Diagrama de colaboración de la configuración de la simulación

5.3. Implementación

La configuración de la simulación es una parte central del presente simulador. En esta sección se explica como se implementa y se detallan los algoritmos usados para el llevar a cabo el proceso de creación y configuración de la red.

Como ya es sabido easySim, utiliza el simulador de eventos discretos llamado "SimJava" (ver [2]). Para realizar una simulación SimJava define pasos que no se pueden evitar lo que impone que se invoquen ciertos métodos en un determinado orden. La configuración de la simulación sigue estos pasos. Brevemente se resumen en llamar a un método que inicializa la simulación, luego se crean y se interconectan las entidades, se define la condición de parada de la simulación y se invoca al método que inicia la simulación.

Como se puede apreciar en los diagramas de dominio (figura 5.1) y diseño (figura 5.2), existen clases encargadas de la configuración (Configuradores) de la simulación y otras encargadas de interpretar(Interpretes) las órdenes dadas por el usuario para configurar la simulación. A continuación se describen los configuradores.

Los configuradores son las clases encargadas de armar la red, crear los nodos, establecer sus parámetros, configurar los módulos, crear las acciones, etc. Aquí se explica con más detalle que hace cada clase de configuración y como lo hace.

5.3.1. Configuración de la Red

La clase `ConfiguracionRed` es la encargada de proveer los métodos necesarios para armar la red. Esto implica crear los nodos y realizar las interconexiones entre ellos. El principal método de esta clase es :

```
conectar(TipoNodo nodo1, String ifazNodo1,TipoNodo nodo2, String ifazNodo2)
```

Es usado como su nombre lo indica para realizar las conexiones, para lo cual crea los nodos de la red en caso de que no existan y crea las interfaces asignándole el nombre que se pasa como parámetro en este método. El método conecta el `nodo1` por la interfaz `ifazNodo1` con el `nodo2` por la interfaz `ifazNodo2`.

Los parámetros `nodo1` y `nodo2` son instancias de las clases que heredan de `TipoNodo`, las cuales no son un nodo de la red, si no que representan un nodo. La idea de usar clases tipo es para que la creación de las conexiones de la red no este únicamente ligadas a usar sintaxis predefinidas, si no que la clase `ConfiguracionRed` solo se ocupe de la configuración de la red y sea una clase interprete que se encargue de fijar la sintaxis. De esta manera se brinda flexibilidad para crear el interprete. Por ejemplo, se podrían ingresar la red desde un archivo de texto, desde una interfaz gráfica ó creando una clase que configure una simulación invocando los métodos de las clases configuradoras de manera totalmente independiente.

Creación de los nodos

Dentro del método `conectar()` de la clase `ConfiguracionRed`, se chequea si los nodos que se pasan como parámetros se encuentran creados. En caso de que no sea así, se invoca al método `crearNodo()` de la clase `TipoNodo`, el cual crea una instancia del respectivo nodo. Si se consulta el diagrama de clases de diseño de la figura 5.2, se aprecia que existen tres clases tipo: `TipoLER`, `TipoLSR` y `TipoHost` que heredan de la clase abstracta `TipoNodo`, cada una de estas crean instancias de las clases `LER`, `LSR` y `Host` respectivamente, donde todas heredan de la clase `Nodo`.

Cuando se crea una instancia de algún nodo, se debe pasar como parámetro en el constructor el nombre del nodo y un `ArrayList` conteniendo una lista de los componentes del nodo que se desea que se creen. Dicho de otra manera el nodo esta compuesto por módulos, tablas, clasificadores y más componentes los cuales se seleccionan agregando en un `ArrayList` el nombre completo de la clase, es decir en el nombre también se especifica el package.

En el constructor de la clase `Nodo`, se recorre el `ArrayList` que contiene los componentes del `Nodo` y para cada elemento se invoca al método `crear()` el cual se adjunta a continuación.

```
private void crear(String modulo, ArrayList modulosRouter) {
 //campos del API Reflect
 Class clase;
 Constructor constructor;
 Method metodos[], unMetodo;
 Object objeto;
 Object[] parametro;
 Class[] tipoParametro;

 //se crea una copia del ArrayList de los nombres de las clases.
 ArrayList copiaModulosRouter = new ArrayList(modulosRouter);
 modulosRouter = null;
 copiaModulosRouter.remove(modulo); //no se necesita

 //contenedor de los metodos setter
 ArrayList metodosSet = new ArrayList();

 try {
   //se carga la clase
   clase = Class.forName(modulo);
   //se obtiene el constructor de la clase
   tipoParametro = new Class[2];
   tipoParametro[0] = Class.forName("java.lang.String");
   tipoParametro[1] = Class.forName("java.lang.String");
   constructor = clase.getConstructor(tipoParametro);

   //parametro que se la pasa a todas las clases
```

```
   parametro = new Object[2];
   parametro[1] = nombreNodo;
   parametro[0] = nombreClase(modulo);

   //se crea una instancia
   objeto = constructor.newInstance(parametro);

   //se guarda una referencia al objeto creado
   referencias.put(nombreClase(modulo), objeto);
   //se obtienen todos los métodos de la clase cargada
   metodos = clase.getMethods();

   //se buscan los metodos setter de la clase y se guardan en la tabla.
   for (int i = 0; i < metodos.length; i++) {
     Iterator iter = copiaModulosRouter.iterator();
     while (iter.hasNext()) {
       String nombreMetodo = "set" + nombreClase((String)iter.next());
       StringBuffer metodoCargado = new StringBuffer(metodos[i].getName());
       if (nombreMetodo.contentEquals(metodoCargado)) {
         metodosSet.add(metodos[i]);
         break;
        }
     }
   }
   if(!metodosSet.isEmpty()){
     setter.put(nombreClase(modulo), metodosSet);
     return;
   }
 }catch(ClassNotFoundException e){
             .
             .
             .
              }
}
```

Este método se basa en el API Reflection de Java, donde las clases que lo definen se encuentran en el package `java.lang.reflect`. El API Reflection provee de métodos para trabajar con las clases y objetos de java, entre las operaciones que se pueden realizar se incluyen invocar de métodos a partir de cadenas de texto, creación de instancias de las clases, modificar valores de los atributos de algún objeto, obtener información acerca de los atributos y métodos de una clase, y más. Existe una basta colección de métodos muy potentes en este package, los cuales se pueden consultar en la documentación de java. No se explicarán en detalle cada línea de código, pero se pueden leer los comentarios del método que pueden dar una buena idea de que hace cada línea.

Se destacan dos puntos muy importantes en el código del método. Uno es que este método crea las instancias de los componentes del nodos mediante la línea `objeto = constructor.newInstance(parametro);`, donde previamente

se había cargado el constructor de la clase correspondiente. Es importante saber que para cargar un constructor al igual que cualquier método se debe indicar la clase de parámetros que este tiene, por lo cual una vez definidos estos parámetros todos los componentes del Nodo tienen que tener el mismo constructor. Este es un requerimiento bastante fuerte pero más adelante se verá su gran potencial. Cuando se crean las instancias de los componentes, se guardan las referencias en un `Hashtable` cuyo nombre es `referencias`.

El otro punto muy importante, es que lógicamente los componentes de los nodos necesitan interactuar entre sí, esto se traduce a nivel de programación, que cada componente que interactue con otros necesita las referencias de dichos objetos. Para establecer las referencias cruzadas, como es normal se usan los métodos `set()`, denominados "setter". El método `crear()` luego de crear una instancia de un componente, obtiene todos los métodos setter de los componentes con los cuales interactuá. Solo se queda con los métodos setter que establecen las referencias cruzadas entre los componentes que se pasan en el `ArrayList` (es decir solo los componentes activos en el momento) y los almacena en un `ArrayList`, dentro de un `Hashtable` denominado `setter` indexado por la clase (variable de tipo `Class`) del componente.

Luego de que se crean todas las instancias de los componentes del Nodo, en el constructor de la clase `Nodo` se invoca al método `armarNodo()`, el cual es el encargado de establecer todas las referencias cruzadas. A continuación se presenta el método para luego realizar una breve descripción del mismo.

```
private void armarNodo() {
  //en caso de que no sea necesario establecer ninguna referencia cruzada
  //sale del método.
  if(setter.isEmpty()) return;

  //campos del API Reflect
  Class clase;
  Method unMetodo;
  Object[] parametro;
  Object objeto;
  //contenedor de metodos setter
  ArrayList coleccionMetodosSet;

  //se recorren todos los objetos creados que tienen métodos setter.
  Enumeration enum = setter.keys();
  while (enum.hasMoreElements()) {
    //se obtiene la primer dupla(nombreModulo, metodos) de la tabla que almacena
    //los métodos setter
    String nombreClaseSetter = (String) enum.nextElement();
    coleccionMetodosSet = (ArrayList) setter.get(nombreClaseSetter);

    //se recorren los métodos setter de cada objeto creado
    Iterator iter = coleccionMetodosSet.iterator();
    while (iter.hasNext()) {
      unMetodo = (Method) iter.next();
```

```
    //luego de obtener un metódo setter se busca entre las Referencias el
    //handle del objeto que se tiene que pasar como parámetro. El nombre del
    //método esta compuesto por la palabra set más el nombre de la clase de
    //la cual se quiere establecer una referencia cruzada.
    Enumeration enumReferencias = referencias.keys();
    while (enumReferencias.hasMoreElements()) {
      String nombreClaseReferencias = (String) enumReferencias.nextElement();
      String nombreMetodoSet = "set" + nombreClase(nombreClaseReferencias);
      StringBuffer metodoCargado = new StringBuffer(unMetodo.getName());

      if (nombreMetodoSet.contentEquals(metodoCargado)) {
        objeto = referencias.get(nombreClaseSetter);
        parametro = new Object[1];
        parametro[0] = referencias.get(nombreClaseReferencias);

        //se invoca el metodo set de la clase cuyo nombreRouter lo indica
        //la variable nombreClaseSetter, y el parámetro lo indica
        //nombreClaseReferencias
        try {
          unMetodo.invoke(objeto, parametro);
          break;
            } catch (IllegalArgumentException e) {
              .
              .
            }
          }
      }
  }
```

El método `armarNodo()` recorre el `Hashtable setter` y en cada iteración obtiene los métodos de un componente dado, esto se hace mediante la línea:

```
coleccionMetodosSet=(ArrayList)setter.get(nombreClaseSetter);
```

Se recorre la colección de métodos setter donde se toma cada uno de los métodos contenidos en `coleccionMetodosSet` y a partir del nombre(este punto quedará más claro al final de la sección) se busca en el `Hashtable referencias`, la referencia al componente que se quiere interactuar. Luego mediante el API Reflection de Java se invoca al método setter del componente en cuestión pasándole la referencia adecuada. Para encontrar la referencia del componente en el `Hashtable referencias`, a partir del nombre del método setter, se forma el nombre de dicho método como la composición de la palabra "set" + nombre del componente del cual se quiere obtener una referencia.

En este punto debe quedar claro que todos los componentes de un Nodo menos la interfaz, se crean a partir de los nombres de la clase que lo implementa, los cuales se pasan en un `ArrayList` al constructor de la clase `Nodo`. Por otro lado también debe quedar en claro que las referencias cruzadas se establecen sin necesidad de invocar de manera explícita al método setter en cuestión. Este

procedimiento de creación de los componentes de los nodos implementado con los métodos `crear()` y `armarNodo()` establece ciertos requerimientos fuertes sobre las clases que implementan los componentes de los nodos.

Con un ejemplo final se pretenden fijar ideas sobre el proceso de creación de los componentes de los nodos y el establecimiento de las referencias cruzadas. A continuación se muestra parte del código de las clases `Rutas` y `Recursos`.

```
public class Rutas{

    //Referencia Recursos
    private Recursos recursos;

    //Constructor de la clase
    public Rutas(String nombre, String nombreRouter){ //Primer restricción
        .
        .
        .
    }

    //Método setter para establecer la referencia cruzada entre la
    //instancia de la clase Recursos.
    public void setRecursos(Recursos recu){//Segunda restricción
        this.recursos = recu;
    }
}//fin de la clase Rutas


public class Recursos{
    //Constructor de la clase
    public Recursos(String nombre, String nombreRouter){ //Primer restricción
        .
        .
    }

}//Fin de la clase Recursos
```

En este ejemplo, ambas clases definen su constructor como lo indican los requerimientos; mediante dos parámetros de tipo `String`, los cuales son el nombre del componente y el nombre del nodo al que pertenece. Por otro lado la clase `Rutas` necesita interactuar con la clase `Recursos`, por lo cual `Rutas` tiene una referencia a `Recursos`. Para realizar el estableciemiento de la referencia cruzada, en la clase `Rutas` se debe implementar el método setter donde su nombre debe estar compuesto por la palabra "set"+"Recursos", según los requerimientos.

En otra parte del programa se definen los componentes que se crearan para cada tipo de nodo, colocando en un `ArrayList` el nombre completo de los mismos. Supongamos que las clases `Rutas` y `Recursos` pertenecen a los packages `easySim.core.nodo.rutas` y `easySim.core.nodo.recursos` respectivamente.

Entonces en alguna clase del programa se encuentran las siguientes líneas:

```
    .
ConfiguracionRed confRed;
ArrayList componentesLER = new ArrayList();
ArrayList componentesLSR = new ArrayList();
ArrayList componentesHost = new ArrayList();
    .
    .
    .
componentesLER.add("easySim.core.nodo.rutas.Rutas");
componentesLER.add("easySim.core.nodo.recursos.Recursos");
    .
    .
    .

//se crea una instancia de la clase ConfiguracionRed
confRed = new ConfiguracionRed(componentesLER, componentesLSR, componentesHost);
confRed.conectar(new TipoLER("LER1"), "i1", new TipoLER("LSR1"), "i2");
    .
    .
```

En las líneas de código anterior se muestra a modo de ejemplo los elementos necesarios para realizar la conexion de dos nodos. Con estas líneas es suficiente para que se ejecuten todos los pasos descritos hasta el momento en esta sección.

Se comienza incluyendo los componentes de un LER, donde están presentes las clases `Rutas` y `Recursos`. Luego se muestra como se usa el `ArrayList` que contiene los nombres de los componentes, los cuales se pasan en el constructor de la clase `ConfiguraciónRed`. La última línea muestra como se realiza una conexión entre dos nodos invocando al método `conectar()` y es aquí donde comienza el proceso de conexión en donde se van a crear los nodos (un LER y un LSR) por medio del método `crear()`. Por último se establecerá la referencia cruzada entre `Rutas` y `Recursos`, mediante el método `armarNodo()`.

Se espera que en este punto no existan dudas del proceso de creación y conexión de nodos de la red. Por otra parte se quiere hacer énfasis en la relativa sencillez para agregar nuevos componentes a los nodos. Simplemente se debe crear una clase que respete los requerimientos impuestos, los cuales se vuelven a describir para que queden claros y agregrar el nombre completo de la clase en el `ArrayList` del tipo de nodo que se desee incorporar el nuevo componente.

Constructor El constructor de la clase que implemente el componente debe contener dos parámetros de tipo `String`, donde el primero se usa para asignarle un nombre al componente y el segundo se usa para pasar el nombre del nodo al que pertenece el componente.

Referencias cruzadas Cuando se necesita establecer una referencia a otro componente del nodo, se debe crear el método setter cuyo nombre este compuesto por la palabra “set” más el nombre del componente al cual se quiere

establecer la referencia y como parámetro debe recibir la referencia al componente en cuestión (ej . `setRecursos(Recursos recu)`).

Capítulo 6

Datos

Uno de los objetivos de realizar una simulación, es obtener datos acerca de alguna propiedad de la red, como puede ser obtener el retardo entre dos puntos de la red, la cantidad de memoria ocupada en el buffer de la cola a lo largo del tiempo y tantas otras propiedades mensurables como el analista necesite.

6.1. Análisis de requerimientos.

No es un requerimiento que el programa deba ofrecer la posibilidad de graficar cualquier propiedad que el usuario defina en la red, alcanza con que el programa ofrezca las propiedades básicas que se definen en esta sección. Pero si es un requerimiento que un usuario avanzado pueda fácilmente agregar una nueva propiedad, de la cual se pueden extraer datos de la red para procesarlos. El hecho de la escalabilidad del programa debe estar presente en el mayor contexto posible del mismo.

No existe una restricción con respecto de donde se deben obtener los datos de la simulación, pero si deben de ser independientes con respecto a la propiedad que se esté registrando. Es decir en caso de que se este almacenando el tamaño del buffer de la cola de una interfaz de un router, estos datos no se deben mezclar con los datos de otras interfaces del mismo router.

Los datos muestreados pueden ser almacenados tanto en memoria ram, como en el disco duro u otro dispositivo de almacenamiento. El objetivo de esto es que se pueda adecuar el simulador a las necesidades de cada usuario y PC.

Además de almacenar datos durante la simulación, el programa debe de ser capaz de procesarlos para obtener resultados gráficos y estadísticos. Los resultados principales que deben existir son los que se listan a continuación.

rate taza de los paquetes medida en kbps, en punto de la red.

tamaño paquetes tamaño de los paquetes en función del tiempo en el algún punto de la red. También debe ser posible mostrar la distribución acumulada de los tamaños.

tamaño buffer tamaño del buffer de las colas en función del tiempo.

retardo retardo de los paquetes entre dos puntos de la red. Esto abarca la posibilidad de capturar el retardo en las colas en el caso de que los puntos de muestreo sean a la entrada y salida de la cola de una interfaz.

jitter variación del retardo entre dos puntos de la red. Al igual que para el retardo, se puede determinar el jitter en la cola.

tiempo inter-paquete tiempo entre la llegada de paquetes en un punto de la red. También debe ser posible obtener la distribución acumulada de este parámetro.

cantidad paquetes cantidad de paquetes almacenados en una cola a lo largo del tiempo.

Según lo mencionado hasta el momento, el procesamiento de datos se da en dos etapas. La primer etapa consiste en la recaudación de los datos, los cuales podrán almacenarse tanto en memoria ó en disco según el usuario escoja. La segunda etapa consiste en la presentación de resultados tanto gráficos como estadísticos.

Para fijar ideas, se presenta el siguiente ejemplo. Se considera una red a la cual se conectan los "Host1"y "Host2", donde el generador G1 del "Host1", envía paquetes de datos con destino el "Host2". El usuario selecciona obtener el rate de los paquetes que provienen del generador G1 del "Host1", tomando los datos en el "Host2". De esta manera se estaría obteniendo el throughput de los datos enviados por el generador G1.

Una observación válida del ejemplo anterior, es que hay que tener en cuenta que en el Host de destino, (Host2) no solo pueden llegar paquetes de datos enviados por el generador G1, por lo cual hay que filtrar los paquetes.

Diagrama de dominio

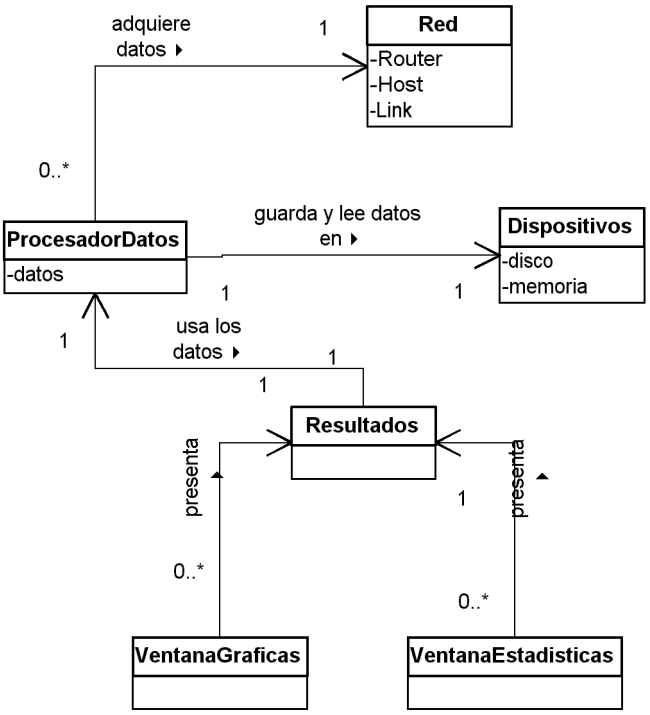


Figura 6.1: Diagrama de dominio del procesamiento de datos

6.2. Diseño

6.2.1. Diagrama de Clases

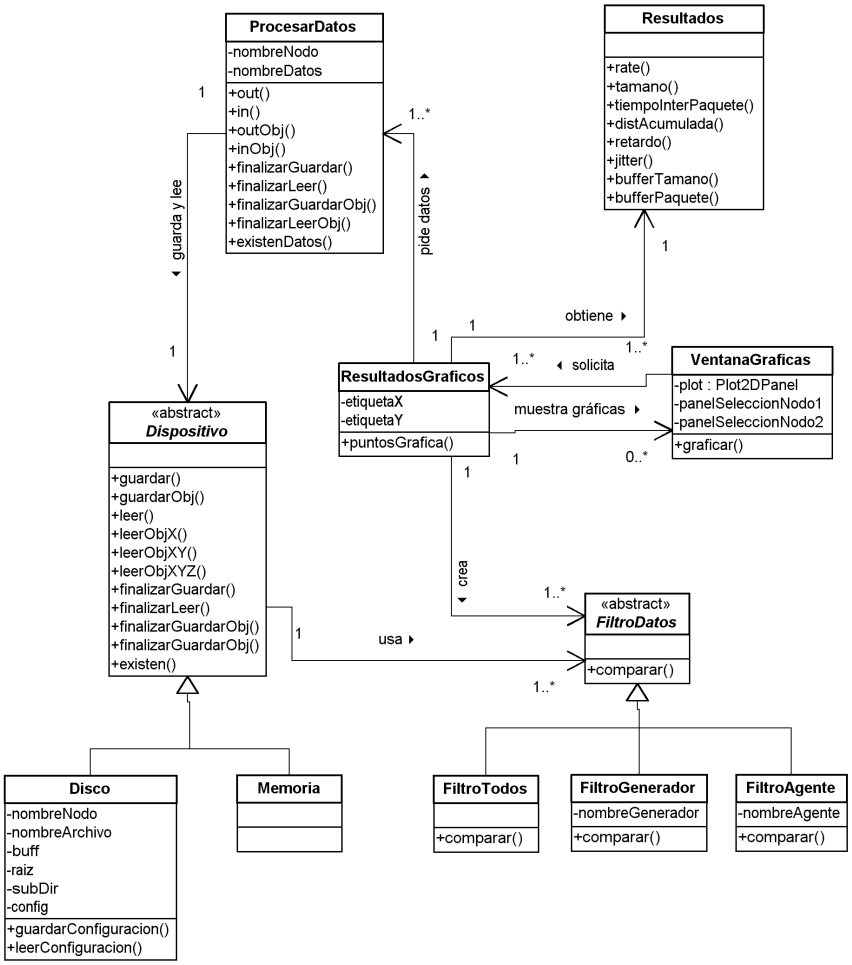


Figura 6.2: Diagrama de clase de Diseño del procesamiento de datos

6.2.2. Diagramas de Colaboración

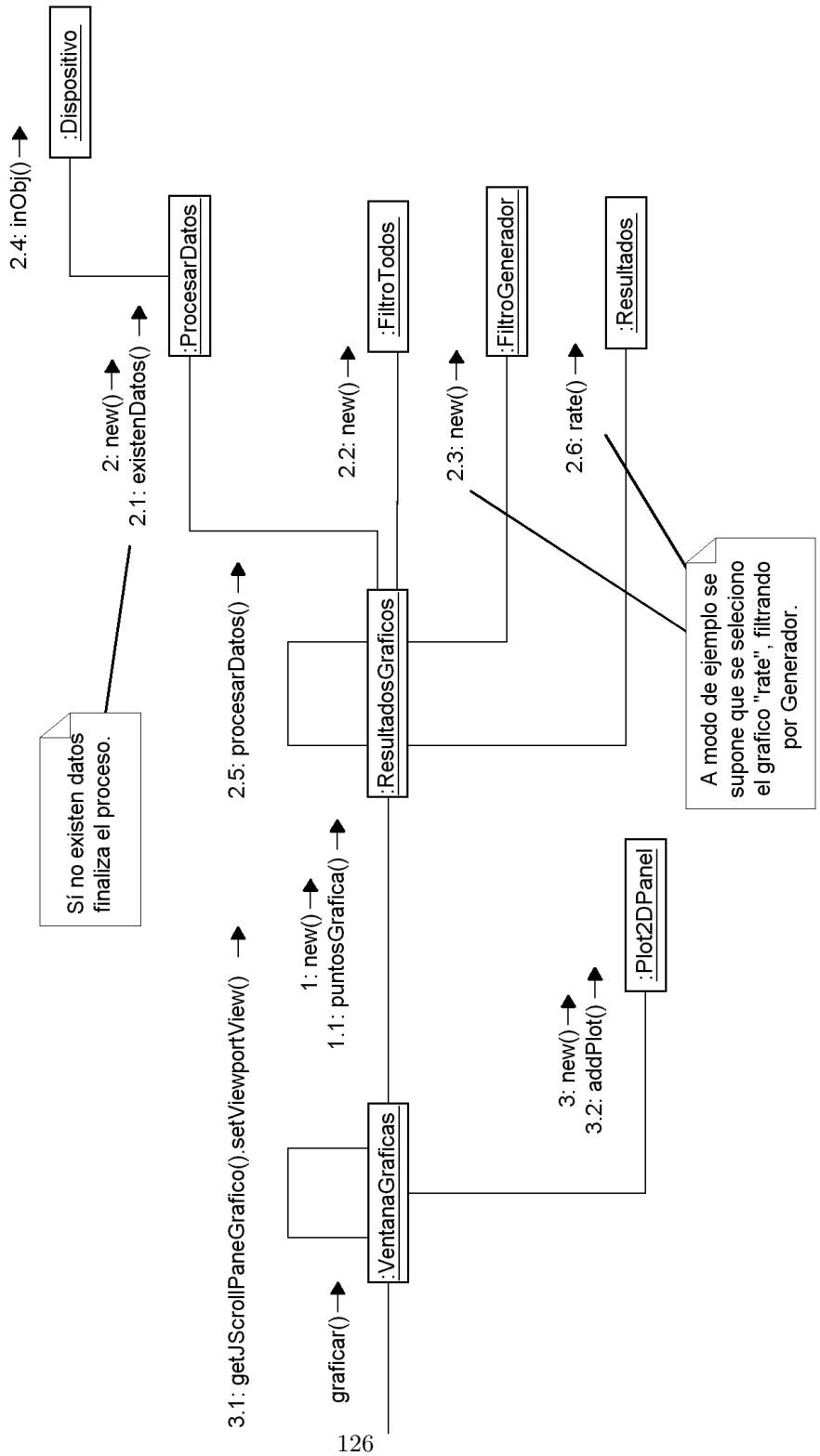


Figura 6.3: Diagrama de colaboración del procesamiento de datos

6.3. Implementación

Capítulo 7

Animaciones

Capítulo 8

Comparación de Resultados

Capítulo 9

Conclusiones

Capítulo 10

Trabajos Futuros

Apéndice A

Simulación

En este apéndice se intenta brindar al lector una breve introducción al concepto de simulación. Toda esta sección fue extraída de [1], de modo que el lector puede recurrir a la citada referencia por mas información al respecto.

Una simulación es la imitación de la operación de un proceso en el mundo real sobre el tiempo. Ya sea mediante una computadora o hecha a mano, una simulación involucra la creación de una historia artificial del sistema, y mediante la observación de esa historia es que se infieren ciertas características de operación del sistema.

El comportamiento de un sistema que involucra al tiempo, es estudiado mediante un modelo de simulación. Este modelo generalmente toma la forma de un conjunto de hipótesis concernientes a la operación del sistema. Estas hipótesis son expresadas en términos matemáticos, lógicos, y relaciones simbólicas entre las diferentes entidades, u objetos de interés del sistema. Una vez desarrollado y validado, un modelo puede ser usado para contestar preguntas del estilo "¿qué pasa si...?."cerca del sistema del mundo real. Los cambios potenciales del sistema pueden ser primero simulados para predecir el comportamiento en la performance del mismo. La simulación también puede ser usada para estudiar sistemas en la etapa del diseño antes que éstos sean construidos. Por lo tanto, la simulación puede ser usada como una herramienta de análisis o para predecir cambios de sistemas ya existentes, y como una herramienta de diseño para predecir la performance de nuevos sistemas bajo un conjunto dado de circunstancias.

En muchos casos, un sistema puede ser modelado matemáticamente y se puede llegar a analizar de esa manera. Sin embargo, la complejidad de algunos sistemas impide tal desarrollo y la simulación se vuelve una herramienta muy valiosa. Durante una simulación se recolectan datos como si estos fuesen adquiridos en el mundo real, que luego serán útiles para medir la performance del sistema.

A.1. Componentes de un sistema

Con el objeto de comprender y analizar un sistema, es necesario definir algunos conceptos.

- Una entidad es un objeto de interés en el sistema.
- Un atributo es una propiedad de la entidad.
- Una actividad representa un período de tiempo de largo especificado.

Muchas veces el sistema total puede ser dividido en subsistemas para un estudio particular. El estado de un sistema es definido como la colección de variables necesarias para describir al sistema en un momento específico del tiempo, relativo a los objetos de estudio. Un evento es definido como una ocurrencia instantánea que puede llegar a cambiar el estado del sistema.

A.2. Sistemas discretos y continuos

Los sistemas pueden ser clasificados en discretos o continuos. Un sistema discreto es aquel en donde las variables cambian solo en un conjunto de puntos discretos del tiempo. En contraposición, un sistema se dice que es continuo cuando las variables cambian continuamente en el tiempo.

A.3. Modelo de un sistema

Muchas veces es de interés estudiar un sistema para entender la relación entre las entidades que lo componen o predecir el comportamiento del mismo bajo la aplicación de nuevas políticas. Sin embargo, muchas veces no es posible experimentar con el sistema real. En el caso de una red de datos, sería impensable para investigar el comportamiento de alguna cola particular de la red, cambiar los enlaces de fibra por inalámbricos y luego volver a hacia atrás. En estos casos resulta muy valiosa la herramienta de simulación. Un modelo se define como una representación de un sistema con el propósito de estudiarlo. Para muchos estudios, solo es necesario considerar los aspectos del sistema que afectan al sistema bajo investigación. Estos aspectos están representados en un modelo del sistema y el modelo, es por definición, una simplificación del sistema. Por otro lado, el modelo del sistema podría ser lo suficientemente detallado como para permitir deducir conclusiones válidas para el sistema real.

A.4. Tipos de modelos

Los modelos pueden ser clasificados como matemáticos o físicos. Un modelo matemático usa notación simbólica para representar al sistema. Un modelo de simulación es un tipo particular de modelo matemático de un sistema. A su vez, los modelos de simulación pueden ser representados como estáticos o dinámicos, determinísticos o estocásticos y discretos o continuos. Un modelo estático, muchas veces llamado simulación Monte Carlo, representa al sistema en un punto particular del tiempo. Un modelo de simulación dinámico representa el sistema conforme éste cambia con el tiempo. Si el modelo de simulación no contiene variables aleatorias, se dice que es un modelo determinístico. Estos modelos contienen un único conjunto de entradas bien conocidas que producen un único conjunto de salidas. Si el modelo tiene una o más variables aleatorias, es un modelo estocástico. Entradas aleatorias producen salidas aleatorias. Ya que las salidas son aleatorias, ellas solo pueden ser consideradas como estimaciones de las salidas reales del sistema.

En la sección A.2, se definieron los sistemas continuos y discretos. Los modelos continuos y discretos se definen de manera análoga. Sin embargo, no siempre un modelo de simulación continuo es usado para modelar un sistema continuo ni un modelo de simulación discreto es usado para simular un sistema discreto. Además, los modelos de simulación pueden ser mixtos, es decir, continuos y discretos. La elección de modelar un sistema con un modelo continuo o discreto depende de la aplicación del modelo.

Apéndice B

SimJava

En SimJava, cada sistema es un conjunto de entidades que interactúan entre sí. Éstas entidades se conectan una a otra a través de puertos y se comunican pasándose eventos. Una clase

The approach to simulating systems adopted by SimJava is similar to other process based simulation packages. Each system is considered to be a set of interacting processes or entities as they are referred to in SimJava. These entities are connected together by ports and communicate with each other by passing events. A central system class controls all the threads, advances the simulation time, and delivers the events. The progress of the simulation is recorded through trace messages produced by the entities and saved in a file.

Apéndice C

SimJava

En SimJava, cada sistema es un conjunto de entidades que interactúan entre sí. Éstas entidades se conectan una a otra a través de puertos y se comunican pasándose eventos. Una clase

The approach to simulating systems adopted by SimJava is similar to other process based simulation packages. Each system is considered to be a set of interacting processes or entities as they are referred to in SimJava. These entities are connected together by ports and communicate with each other by passing events. A central system class controls all the threads, advances the simulation time, and delivers the events. The progress of the simulation is recorded through trace messages produced by the entities and saved in a file.

Apéndice D

Teoría de Colas

D.1. Modelos de Colas

Apéndice E

Políticas de Descarte

Apéndice F

Disciplinas de Despache

Glosario

BW Band Width

ER Explicit Route

FEC Forwarding Equivalent Class

FTN FEC to NHLF

ILM Incoming Label Map

LER Label Edge Router

LSR Label Switch Router

LSP Label Switch Path

LSPID LSP Identifier

MHR Minimum Hop Routing

MPLS Multi Protocol Label Switching

NHLF Next Hop Label Forwarding

NHLFE Next Hop Label Forwarding Entry

OSPF Open Short Path First

UC Unidad de Control

ESR EasySim Revolution

BW Ancho de Banda

Índice Clases

Nombre de la clase	Package
AbsoluteCoord.java	\src\org\jmathplot\gui\plotObjects
Agente.java	\src\easySim\core\nodo\agentes
Algoritmos.java	\src\easySim\core\nodo\moduloRutas\algoritmos
Anim_applet.java	\src\eduni\simanim
Anim_entity.java	\src\eduni\simanim
Anim_event.java	\src\eduni\simanim
Anim_param.java	\src\eduni\simanim
Anim_port.java	\src\eduni\simanim
Animacion.java	\src\easySim\interfazSimJava
AnimacionLSP.java	\src\easySim\core\nodo\test
ASCIIFile.java	\src\org\jmathplot\io\files
Axe.java	\src\org\jmathplot\gui\plotObjects
back.png	\src\org\jmathplot\gui\icons
BarPlot.java	\src\org\jmathplot\gui\plots
Base.java	\src\org\jmathplot\gui\plotObjects
Base2D.java	\src\org\jmathplot\gui\plotObjects
Base3D.java	\src\org\jmathplot\gui\plotObjects
BaseLabel.java	\src\org\jmathplot\gui\plotObjects
BaseScalesDependant.java	\src\org\jmathplot\gui\plotObjects
Bernoulli.java	\src\easySim\interfazSimJava\distribuciones

Beta.java	\src\easySim\interfazSimJava\distribuciones
BetaPrima.java	\src\easySim\interfazSimJava\distribuciones
Binomial.java	\src\easySim\interfazSimJava\distribuciones
BoxPlot2D.java	\src\org\jmathplot\gui\plots
BoxPlot3D.java	\src\org\jmathplot\gui\plots
Calculo.java	\src\easySim\core\procesamiento\matematicas
Cancelador.java	\src\easySim\interfazSimJava
Cauchy.java	\src\easySim\interfazSimJava\distribuciones
center.png	\src\org\jmathplot\gui\icons
ChiCuadrado.java	\src\easySim\interfazSimJava\distribuciones
ClasificadorControl.java	\src\easySim\core\nodo\clasificadores
ClasificadorDatos.java	\src\easySim\core\nodo\clasificadores
ClasificadorHost.java	\src\easySim\core\nodo\clasificadores
ClipBoardPrintable.java	\src\org\jmathplot\io
Cola.java	\src\easySim\core\nodo\interfaz
CommandLinePrintable.java	\src\org\jmathplot\io
Comment.java	\src\org\jmathplot\gui\plotObjects
ConfGenerador.java	\src\easySim\core\procesamiento\texto
ConfiguracionHost.java	\src\easySim\core\configuracion
ConfiguracionNodo.java	\src\easySim\core\configuracion
ConfiguracionRed.java	\src\easySim\core\configuracion
ConfiguracionRouter.java	\src\easySim\core\configuracion
ConfiguracionTexto.java	\src\easySim\core\procesamiento\texto
ConfInterfaz.java	\src\easySim\core\procesamiento\texto
ConfLSP.java	\src\easySim\core\procesamiento\texto
ContinuousGenerator.java	\src\eduni\simjava\distributions
ControladorAgentes.java	\src\easySim\core\nodo\tablas
ControladorGenerador.java	\src\easySim\core\nodo\tablas

Coord.java	\src\org\jmathplot\gui\plotObjects
CriterioFEC.java	\src\easySim\core\nodo\moduloFec
data.png	\src\org\jmathplot\gui\icons
DataFile.java	\src\org\jmathplot\io\files
DataPanel.java	\src\org\jmathplot\gui
DatasFrame.java	\src\org\jmathplot\gui\components
DataToolBar.java	\src\org\jmathplot\gui\components
Descubrimiento.java	\src\easySim\core\nodo\moduloDescubrimiento
DescubrimientoOSPF.java	\src\easySim\core\nodo\moduloDescubrimiento
Despachador.java	\src\easySim\core\nodo\interfaz\disciplinasDeDespache
Dijsktra.java	\src\easySim\core\nodo\moduloRutas\algoritmos
DisciplinaDeDespache.java	\src\easySim\core\nodo\interfaz\disciplinasDeDespache
Disco.java	\src\easySim\core\procesamiento\datos
DiscreteGenerator.java	\src\eduni\simjava\distributions
Dispositivo.java	\src\easySim\core\procesamiento\datos
Distribucion.java	\src\easySim\interfazSimJava\distribuciones
DistribucionContinua.java	\src\easySim\interfazSimJava\distribuciones
DistribucionDiscreta.java	\src\easySim\interfazSimJava\distribuciones
DoubleArray.java	\src\org\jmathplot\util
DropTail.java	\src\easySim\core\nodo\interfaz\politicasDeColas
DuplaClaveValor.java	\src\easySim\core\nodo\tablas
DWRR.java	\src\easySim\core\nodo\interfaz\disciplinasDeDespache
easySim.core.nodo.moduloRutas.svg	\src\easySim\core\nodo\moduloRutas
EncabezadoIP.java	\src\easySim\core\nodo\moduloRuteoCapaRed
EncabezadoMPLS.java	\src\easySim\core\nodo\moduloMPLS
Entity.java	\src\easySim\interfazSimJava
EnvioMPLS.java	\src\easySim\core\nodo\moduloMPLS
Erlang.java	\src\easySim\interfazSimJava\distribuciones

EtiquetaMPLS.java	\src\easySim\core\nodo\moduloMPLS
Evento.java	\src\easySim\interfazSimJava
Evqueue.java	\src\eduni\simjava
Exponencial.java	\src\easySim\interfazSimJava\distribuciones
F.java	\src\easySim\interfazSimJava\distribuciones
Fec.java	\src\easySim\core\nodo\moduloFec
Fifo.java	\src\easySim\core\nodo\interfaz\disciplinasDeDespache
FilePrintable.java	\src\org\jmathplot\io
Filter.java	\src\easySim\interfazSimJava
FiltroCancelarLSP.java	\src\easySim\interfazSimJava
FiltroDatos.java	\src\easySim\core\procesamiento\datos
FiltroGenerador.java	\src\easySim\core\procesamiento\datos
FiltroReenviarLSU.java	\src\easySim\interfazSimJava
FiltroTirarRouter.java	\src\easySim\interfazSimJava
FiltroTodos.java	\src\easySim\core\procesamiento\datos
FQ.java	\src\easySim\core\nodo\interfaz\disciplinasDeDespache
FrameView.java	\src\org\jmathplot\gui
FuenteUC.java	\src\easySim\core\nodo\unidadControl
Gamma.java	\src\easySim\interfazSimJava\distribuciones
GammaInversa.java	\src\easySim\interfazSimJava\distribuciones
Generador.java	\src\easySim\core\nodo\generadores
GeneradorBasico.java	\src\easySim\core\nodo\generadores
Generator.java	\src\eduni\simjava\distributions
Geometrica.java	\src\easySim\interfazSimJava\distribuciones
GraphClearObject.java	\src\eduni\simdiag
GraphData.java	\src\eduni\simdiag
GraphDiagram.java	\src\eduni\simdiag
GraphDisplay.java	\src\eduni\simdiag

GraphEqn.java	\src\eduni\simdiag
GraphEventObject.java	\src\eduni\simdiag
GraphListener.java	\src\eduni\simdiag
GraphLoader.java	\src\eduni\simdiag
GraphSetAxes.java	\src\eduni\simdiag
GraphSetScale.java	\src\eduni\simdiag
GraphWindow.java	\src\eduni\simdiag
Grid.java	\src\org\jmathplot\gui\plotObjects
Hello.java	\src\easySim\core\paquetes\ospf
HistogramPlot2D.java	\src\org\jmathplot\gui\plots
HistogramPlot3D.java	\src\org\jmathplot\gui\plots
Host.java	\src\easySim\core\entidades
Ifaz.java	\src\easySim\core\nodo\interfaz
ImagenHost.caido.gif	\src\easySim\gui\ventanas\bitmaps
imagenHost.gif	\src\easySim\gui\ventanas\bitmaps
imagenHost.levantado.gif	\src\easySim\gui\ventanas\bitmaps
imagenPuerto.gif	\src\easySim\gui\ventanas\bitmaps
imagenRouter.caido.gif	\src\easySim\gui\ventanas\bitmaps
imagenRouter.gif	\src\easySim\gui\ventanas\bitmaps
imagenRouter.levantado.gif	\src\easySim\gui\ventanas\bitmaps
InfoAdicional.java	\src\easySim\core\nodo\moduloFec
InvBwdisponible.java	\src\easySim\core\nodo\moduloRutas\metricas
InvBwReservado.java	\src\easySim\core\nodo\moduloRutas\metricas
Label.java	\src\org\jmathplot\gui\plotObjects
LabelPop.java	\src\easySim\core\nodo\moduloMPLS
LabelPush.java	\src\easySim\core\nodo\moduloMPLS
LabelSwap.java	\src\easySim\core\nodo\moduloMPLS
LER.java	\src\easySim\core\entidades

Lifo.java	\src\easySim\core\nodo\interfaz\disciplinasDeDespache
Line.java	\src\org\jmathplot\gui\plotObjects
LinePlot.java	\src\org\jmathplot\gui\plots
Logistica.java	\src\easySim\interfazSimJava\distribuciones
LSP.java	\src\easySim\core\nodo\moduloMPLS
LspNotify.java	\src\easySim\core\paquetes\ldp
LspRelease.java	\src\easySim\core\paquetes\ldp
LspRequest.java	\src\easySim\core\paquetes\ldp
LspSetup.java	\src\easySim\core\paquetes\ldp
LspWithdraw.java	\src\easySim\core\paquetes\ldp
LSR.java	\src\easySim\core\entidades
LSU.java	\src\easySim\core\paquetes\ospf
LSUACK.java	\src\easySim\core\paquetes\ospf
MatrixString.java	\src\org\jmathplot\io
MatrixTablePanel.java	\src\org\jmathplot\gui
Metrica.java	\src\easySim\core\nodo\moduloRutas\metricas
MinAdWeightRouting.java	\src\easySim\core\nodo\moduloRutas\metricas
MinHopRouting.java	\src\easySim\core\nodo\moduloRutas\metricas
Nodo.java	\src\easySim\core\entidades
Normal.java	\src\easySim\interfazSimJava\distribuciones
NormalLog.java	\src\easySim\interfazSimJava\distribuciones
Noteable.java	\src\org\jmathplot\gui\plotObjects
Operacion.java	\src\easySim\core\nodo\moduloMPLS
OrigenDestino.java	\src\easySim\core\nodo\moduloFec
OrigenIfazEntrada.java	\src\easySim\core\nodo\moduloFec
Out_File.java	\src\easySim\interfazSimJava
Out_File_CSV.java	\src\easySim\interfazSimJava
package.html	\src\easySim\core\configuracion

package.html	\src\easySim\core\entidades
package.html	\src\easySim\core\nodo\agentes
package.html	\src\easySim\core\nodo\clasificadores
package.html	\src\easySim\core\nodo\generadores
package.html	\src\easySim\core\nodo\interfaz
package.html	\src\easySim\core\nodo\interfaz\disciplinasDeDespache
package.html	\src\easySim\core\nodo\interfaz\politicasDeColas
package.html	\src\easySim\core\nodo\moduloDescubrimiento
package.html	\src\easySim\core\nodo\moduloFec
package.html	\src\easySim\core\nodo\moduloMPLS
package.html	\src\easySim\core\nodo\moduloRecursos
package.html	\src\easySim\core\nodo\moduloRutas
package.html	\src\easySim\core\nodo\moduloRutas\algoritmos
package.html	\src\easySim\core\nodo\moduloRutas\metricas
package.html	\src\easySim\core\nodo\moduloRuteoCapaRed
package.html	\src\easySim\core\nodo\moduloSenializacion
package.html	\src\easySim\core\nodo\tablas
package.html	\src\easySim\core\nodo\test
package.html	\src\easySim\core\nodo\unidadControl
package.html	\src\easySim\core\paquetes
package.html	\src\easySim\core\paquetes\datos
package.html	\src\easySim\core\paquetes\ldp
package.html	\src\easySim\core\paquetes\ospf
package.html	\src\easySim\core\procesamiento\datos
package.html	\src\easySim\core\procesamiento\matematicas
package.html	\src\easySim\interfazSimJava
package.html	\src\easySim\interfazSimJava\distribuciones
package.html	\src\eduni\simanim

package.html	\src\eduni\simdiag
package.html	\src\eduni\simjava
package.html	\src\eduni\simjava\distributions
PanelSelecionGrafico.java	\src\easySim\gui\dialogo
PaqueteBasico.java	\src\easySim\core\paquetes\datos
PaqueteIP.java	\src\easySim\core\paquetes\datos
Paquetes.java	\src\easySim\core\paquetes
PaquetesControl.java	\src\easySim\core\paquetes
PaquetesDatos.java	\src\easySim\core\paquetes
PaquetesDescubrimiento.java	\src\easySim\core\paquetes
PaquetesLdp.java	\src\easySim\core\paquetes\ldp
PaquetesOspf.java	\src\easySim\core\paquetes\ospf
PaquetesSenializacion.java	\src\easySim\core\paquetes
Param_type.java	\src\eduni\simanim
Param_type_list.java	\src\eduni\simanim
ParamConfig.java	\src\easySim\core\procesamiento\datos
ParametersPanel.java	\src\org\jmathplot\gui
ParametrosNHLF.java	\src\easySim\core\nodo\tablas
ParametrosTablaRecursos.java	\src\easySim\core\nodo\tablas
ParametrosTablaRuteo.java	\src\easySim\core\nodo\tablas
Pareto.java	\src\easySim\interfazSimJava\distribuciones
Pascal.java	\src\easySim\interfazSimJava\distribuciones
Plot.java	\src\org\jmathplot\gui\plots
Plot2DPanel.java	\src\org\jmathplot\gui
Plot3DPanel.java	\src\org\jmathplot\gui
Plotable.java	\src\org\jmathplot\gui\plotObjects
PlotPanel.java	\src\org\jmathplot\gui
PlotToolBar.java	\src\org\jmathplot\gui\components

Poisson.java	\src\easySim\interfazSimJava\distribuciones
PoliticaDeDescarte.java	\src\easySim\core\nodo\interfaz\politicasDeColas
position.png	\src\org\jmathplot\gui\icons
PriorEstrictFifo.java	\src\easySim\core\nodo\interfaz\disciplinasDeDespache
PriorEstrictLifo.java	\src\easySim\core\nodo\interfaz\disciplinasDeDespache
ProcesadorUC.java	\src\easySim\core\nodo\unidadControl
ProcesarDatos.java	\src\easySim\core\procesamiento\datos
Puerto.java	\src\easySim\interfazSimJava
Randomica.java	\src\easySim\interfazSimJava\distribuciones
Recursos.java	\src\easySim\core\nodo\moduloRecursos
RecursosAbs.java	\src\easySim\core\nodo\moduloRecursos
RelativeCoord.java	\src\org\jmathplot\gui\plotObjects
Resultados.java	\src\easySim\core\procesamiento\datos
ResultadosGraficos.java	\src\easySim\core\procesamiento\datos
rotation.png	\src\org\jmathplot\gui\icons
Router.java	\src\easySim\core\entidades
Rutas.java	\src\easySim\core\nodo\moduloRutas
RutasAbs.java	\src\easySim\core\nodo\moduloRutas
Ruteo.java	\src\easySim\core\nodo\moduloRuteoCapaRed
RuteoCapaDosYMedio.java	\src\easySim\core\nodo\moduloMPLS
RuteoCapaRed.java	\src\easySim\core\nodo\moduloRuteoCapaRed
scale.png	\src\org\jmathplot\gui\icons
ScatterPlot.java	\src\org\jmathplot\gui\plots
Semaphore.java	\src\eduni\simjava
Senializacion.java	\src\easySim\core\nodo\moduloSenializacion
SenializacionLDP.java	\src\easySim\core\nodo\moduloSenializacion
SetScalesFrame.java	\src\org\jmathplot\gui\components
Sim_accum.java	\src\eduni\simjava

Sim_anim.java	\src\eduni\simanim
Sim_any_p.java	\src\eduni\simjava
Sim_bernoulli_obj.java	\src\eduni\simjava\distributions
Sim_beta_obj.java	\src\eduni\simjava\distributions
Sim_betaprime_obj.java	\src\eduni\simjava\distributions
Sim_binomial_obj.java	\src\eduni\simjava\distributions
Sim_cauchy_obj.java	\src\eduni\simjava\distributions
Sim_chisquare_obj.java	\src\eduni\simjava\distributions
Sim_entity.java	\src\eduni\simjava
Sim_erlang_obj.java	\src\eduni\simjava\distributions
Sim_event.java	\src\eduni\simjava
Sim_exception.java	\src\eduni\simjava
Sim_f_obj.java	\src\eduni\simjava\distributions
Sim_file.java	\src\easySim\interfazSimJava
Sim_from_p.java	\src\eduni\simjava
Sim_gamma_obj.java	\src\eduni\simjava\distributions
Sim_geometric_obj.java	\src\eduni\simjava\distributions
Sim_invgamma_obj.java	\src\eduni\simjava\distributions
Sim_logistic_obj.java	\src\eduni\simjava\distributions
Sim_lognormal_obj.java	\src\eduni\simjava\distributions
Sim_negexp_obj.java	\src\eduni\simjava
Sim_negexp_obj.java	\src\eduni\simjava\distributions
Sim_none_p.java	\src\eduni\simjava
Sim_normal_obj.java	\src\eduni\simjava
Sim_normal_obj.java	\src\eduni\simjava\distributions
Sim_not_from_p.java	\src\eduni\simjava
Sim_not_type_p.java	\src\eduni\simjava
Sim_outfile.java	\src\eduni\simjava

Sim_output.java	\src\eduni\simjava
Sim_parameter_exception.java	\src\eduni\simjava\distributions
Sim_pareto_obj.java	\src\eduni\simjava\distributions
Sim_pascal_obj.java	\src\eduni\simjava\distributions
Sim_poisson_obj.java	\src\eduni\simjava\distributions
Sim_port.java	\src\eduni\simjava
Sim_predicate.java	\src\eduni\simjava
Sim_random_obj.java	\src\eduni\simjava\distributions
Sim_reporter.java	\src\eduni\simjava
Sim_reportfile.java	\src\eduni\simjava
Sim_stat.java	\src\eduni\simjava
Sim_stat_exception.java	\src\eduni\simjava
Sim_system.java	\src\eduni\simjava
Sim_tstudent_obj.java	\src\eduni\simjava\distributions
Sim_type_p.java	\src\eduni\simjava
Sim_uniform_obj.java	\src\eduni\simjava
Sim_uniform_obj.java	\src\eduni\simjava\distributions
Sim_weibull_obj.java	\src\eduni\simjava\distributions
SimuAnimada.java	\src\easySim\core\nodo\test
Simulacion.java	\src\easySim\core\configuracion
SimulacionBasica.java	\src\easySim\core\nodo\test
SimulacionDWRR.java	\src\easySim\core\nodo\test
SimulacionFish.java	\src\easySim\core\nodo\test
SimulacionLSP.java	\src\easySim\core\nodo\test
Sistema.java	\src\easySim\interfazSimJava
StaircasePlot.java	\src\org\jmathplot\gui\plots
StringPrintable.java	\src\org\jmathplot\io
Tabla.java	\src\easySim\core\nodo\tablas

TablaDefinicionFEC.java	\src\easySim\core\nodo\tablas
TablaFTN.java	\src\easySim\core\nodo\tablas
TablaILM.java	\src\easySim\core\nodo\tablas
TablaInterfaces.java	\src\easySim\core\nodo\tablas
TablaLSP.java	\src\easySim\core\nodo\tablas
TablaNHLF.java	\src\easySim\core\nodo\tablas
TablaRecursos.java	\src\easySim\core\nodo\tablas
TablaRuteo.java	\src\easySim\core\nodo\tablas
Texto.java	\src\easySim\core\procesamiento\texto
TimingDiagram.java	\src\eduni\simdiag
TimingWindow.java	\src\eduni\simdiag
TipoHost.java	\src\easySim\core\configuracion
TipoLER.java	\src\easySim\core\configuracion
TipoLSR.java	\src\easySim\core\configuracion
TipoNodo.java	\src\easySim\core\configuracion
toclipboard.png	\src\org\jmathplot\gui\icons
tofile.png	\src\org\jmathplot\gui\icons
Traceable.java	\src\eduni\simdiag
TraceEventObject.java	\src\eduni\simdiag
TraceListener.java	\src\eduni\simdiag
TraceLoader.java	\src\eduni\simdiag
TraceSaver.java	\src\eduni\simdiag
TStudent.java	\src\easySim\interfazSimJava\distribuciones
UDP.java	\src\easySim\core\nodo\agentes
Uniforme.java	\src\easySim\interfazSimJava\distribuciones
VentanaAnimacion.java	\src\easySim\gui\ventanas
VentanaConfTexto.java	\src\easySim\gui\ventanas
VentanaEstadoTablas.java	\src\easySim\gui\ventanas

VentanaGraficas.java	\src\easySim\gui\ventanas
VentanaPrincipal.java	\src\easySim\gui\ventanas
VentanaTablas.java	\src\easySim\gui\ventanas
Weibull.java	\src\easySim\interfazSimJava\distribuciones
WFQ.java	\src\easySim\core\nodo\interfaz\disciplinasDeDespache
WRED.java	\src\easySim\core\nodo\interfaz\politicasDeColas
zoom.png	\src\org\jmathplot\gui\icons

Bibliografía

[1] Discrete-Event System Simulation (Second Edition), Jerry Banks, John S. Carson II, Barry L. Nelson. PRENTICE HALL . ISBN 0-13-217449-9

[2] Manual de SimJava.

[3] Redes de computadoras. Andrew S. Tanenbaum

[4] ÜML y Patrones". Craig Larman.

[5] Random Early Detection Gateways for Congestion Avoidance. Sally Floyd and Van Jacobson. http://www.icir.org/floyd/papers/red/red.html

[6] DiffServ: The Scalable End to End QoS Model http://www.cisco.com/en/US/tech/tk543/tk766/technologies_white_paper09186a00800a3e2f.shtml

[7] The Java API Documentation Generator. http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/javadoc.html

[8] http://www.sun.com

[9] Constraint Shortest Path First ; TED - Traffic Engineering Database. http://www.omnetpp.org/doc/INET/neddoc/TED-id2295824.html