# PROYECTO FIN DE CARRERA

# SIMULADOR PARA REDES DE TELECOMUNICACIONES BASADAS EN TECNOLOGÍA ATM DESARROLLADO MEDIANTE PROGRAMACIÓN ORIENTADA A OBJETOS

Trabajo presentado por
José María Carazo Abolafia

Director: Prof. Dr. D. Pablo Cortés Achedad
Departamento de Organización Industrial
Escuela Superior de Ingenieros de Sevilla

# Índice de contenido

# 1 OBJETIVO Y ALCANCE

El objeto de estudio de este proyecto fin de carrera son las redes de telecomunicaciones basadas en tecnología de modo de transferencia asíncrono, o ATM[1]. En la actualidad, este tipo de redes está adquiriendo gran difusión, y son las que soportan servicios de comunicación como los de ADSL[2] o UMTS[3].

Debido a la multitud de criterios de optimización que se podrían seleccionar en función del tipo de tráfico, la optimización en ATM se presenta como un problema de complicada resolución, y sin solución única. Por otro lado, se trabaja en tiempo real. Normalmente la solución ha de ser alcanzada en el tiempo que resta desde una petición de conexión hasta su admisión y establecimiento, fácilmente menos de un segundo.

Por todo ello, se hace necesario recurrir a métodos heurísticos apoyados en inteligencia artificial, como búsqueda tabú, algoritmos genéticos y sistemas multiagente. Con esto, se cambia el “encontrar la mejor solución”, por “encontrar una solución suficientemente adecuada, y en un tiempo suficientemente corto. Hay que añadir una obviedad: De nada sirve una buena solución si el algoritmo no es rápido.

[1] *Asinchronous Transfer Mode*. Hace referencia al hecho de que la transmisión de datos por parte de las fuentes no está sincronizado, a diferencia de las redes SDH o SONET, por ejemplo.

[2] *Asimetric Digital Subscriber Line*, Linea de Abonado Digital y Asimétrica. También conocida en el mercado de las telecomunicaciones como acceso a Internet de alta velocidad.

[3] *Universal Mobile Telecomunications System*, Sistema de Telecomunicaciones Móviles Universal. De reciente entrada en el mercado, también se conoce comercialmente como red de móviles de tercera generación.

No parece viable avalar un algoritmo y las soluciones que alcanza mediante una demostración teórica. Por otro lado resulta desproporcionado usar una red ATM real para probar los algoritmos que se propongan. Es necesario recurrir a la simulación.

Un simulador permite probar algoritmos de optimización y depurarlos. Permite contrastar los resultados de unos con otros. Y ello, cuantas veces se quiera, cambiando los escenarios o situaciones, y con un coste mínimo.

Siguiendo esta línea, la propuesta es un simulador que se comporte como una red ATM real, o al menos, que sea lo más fiel posible a la realidad, siempre con unos tiempos de ejecución admisibles.

El objetivo de este proyecto fin de carrera es aplicar un modelo de construcción para simuladores ATM. El primer paso para esto es una descripción de la propia realidad. Después hay que plasmar esa realidad en el simulador. Concretando el objetivo, este proyecto trata de aplicar las premisas de la programación orientada a objetos – OOP (*Object Oriented Programming*) al problema de la simulación de redes ATM, proponiendo la orientación a objetos como un buen método para modelar la realidad en estudio y simularla.

El trabajo continua con el apartado 2, que es un breve recorrido de lo que ha sido la evolución de las redes de telecomunicaciones hasta llegar a la tecnología ATM, a lo que sigue una breve introducción a dicha tecnología. Seguidamente, en el apartado 3, se propone una metodología para el modelado y se hace una descripción cualitativa de los elementos de la red que quieren modelarse. En el apartado 4 se presentan los modelos elaborados con la metodología propuesta. Se acompaña un análisis de los algoritmos que tienen interés y de la ejecución de los distintos elementos de red dotados de

autonomía. En el apartado 5 se muestran los resultados logrados con el simulador en diferentes escenarios. En el apartado 6 se exponen las conclusiones finales y lineas de ampliación. Por último se incluye un anexo con los diagramas UML de las clases diseñadas para el simulador.

# 2 INTRODUCCIÓN

## 2.1 EVOLUCIÓN DE LAS REDES DE TELECOMUNICACIONES

Uno de los desarrollos tecnológicos más importantes que el siglo XX ha visto la humanidad es, sin duda alguna, las redes de telecomunicaciones. Ciertamente, ya en la segunda mitad del XIX se probó una red de telefonía con unos cientos de abonados situados a no más de un kilómetro del edificio en el que se establecían las conexiones[4]. No obstante, ha sido en el siglo pasado cuando los servicios de telecomunicaciones han establecido nuevos hábitos de comportamiento, introduciéndose como uno de los modos de comunicación más básicos y cotidianos, si no el que más. Se puede decir que estos nuevos medios son una de las principales claves de la *tecnologización* de la vida cotidiana.

Primero fue la voz. Paralelamente se fueron desarrollando las comunicaciones de radio y la telefonía. Investigadores como Marconi, Bell o Edison, pusieron las bases físicas necesarias. La radio se estableció como uno de los grandes medios de difusión, hasta que fue posible transmitir imágenes con voz y la televisión se fue haciendo presente en todos los hogares.

Junto con esto, la telefonía adquiría más y mayor difusión. El número de abonados crecía, y esto provocaría el desarrollo de las primeras tecnologías de conmutación y los primeros estudios de modelado del tráfico. La aparición de centrales basadas en conmutadores mecánicos reemplazarían las grandes salas con las clásicas operadoras.

[4]Cfr. Sexton, M. y Reid (1997), A.: *Broadband Networkin: ATM, SDH and SONET,* Artech House. Norwood, MA. Pg. 2.

A mediados del siglo XX, tras el gran desarrollo tecnológico exigido por la II Guerra Mundial, empiezan a desarrollarse las comunicaciones digitales. No se trataba sólo de enviar señales analógicas de voz. Había necesidad de enviar datos. Así, un servicio parecido al télex empezaba a competir con el clásico telégrafo[5]. La necesidad y la capacidad de transmitir datos eran cada vez mayores por el desarrollo de las tecnologías de computación, a lo largo de la segunda mitad de dicho siglo[6].

Con la aparición de los datos y la tecnología digital se establece una profunda separación de las redes según la conmutación:

- ✓ Redes de conmutación de circuitos. Son las tradicionales redes de telefonía. Generalmente se emplean para transmitir voz. La denominación viene dada por el funcionamiento: para poder iniciar una comunicación es necesario establecer antes una trayectoria privada entre los usuarios que precisan comunicarse. Esa trayectoria se mantiene el tiempo que requieran los usuarios. Después se liberan todos y cada uno de los recursos de la red que componían el trayecto. Por lo explicado, en estas redes se tarifica por tiempo de conexión.

- ✓ Redes de conmutación de paquetes. Los paquetes son bloques de datos, generalmente binarios, que genera una fuente y los entrega a la red. Dentro de ésta, cada nodo puede ver especificado en el paquete quién es el destinatario. Según quien sea se lo pasará a un

[5]Idem, p. 7.

[6]Cfr. Schwartz, M. (1994): *Redes de Telecomunicaciones: protocolos, modelado y análisis*. Addison-Wesley Iberoamericana. Wilmington, Delaware. Pg. 3 y ss.

nodo o a otro, hasta que llegue al extremo de la red donde se encuentra el destino.

Como puede verse, no se establece ninguna trayectoria, no hay por tanto establecimiento de la conexión, e incluso cada paquete que genera la fuente puede ir por un camino diferente. Los paquetes, aunque son originados por fuentes distintas, comparten todos los mismos medios de distribución y transmisión. En esta tecnología es más propio que la tarificación sea por cantidad de datos o por la capacidad de transmisión.

Esta excisión en tipos de redes conlleva una dependencia mutua entre la red y el tipo de servicio que presta. Es entonces cuando empiezan a surgir los primeros intentos de unificar los tipos de red, haciendo una capaz de prestar cualquier tipo de servicio. Estos intentos están muy relacionados con la revolución digital, que usando sólo señales digitales permite transmitir cualquier señal analógica como la de voz o televisión. El modo de conseguir la unificación pasa por que la red sea enteramente digital.

Con este motivo, surge el estándar de redes de jerarquía digital plesiócrona – PDH basada en canales de 1540 Kbps en Norte América capaces de transportar 24 canales de voz, de 64kbps. En Europa el canal básico se fija a 2048 Kbps, siendo capaz de transportar 30 canales de voz. En ambos casos las velocidades de transmisión están normalizadas por la Unión Internacional de Telecomunicaciones – UIT.[7]

[7]Cfr. Recomendación UIT-T G.702 – Velocidades Binarias de la Jerarquía Digital.

Más tarde, también normalizado por la UIT, aparecen los estándares para jerarquía digital síncrona – SDH[8]. Usado en Europa, tiene su equivalente norteamericano llamado SONET. SDH se basa en contenedores llamados Módulos de Transporte Síncrono - STM con una determinada capacidad de datos y una velocidad de transmisión asociada. Estos módulos son generados en cada nodo de un modo síncrono. Si el nodo tiene que transmitir datos, los encapsula en el módulo. La capacidad básica corresponde a STM-1 (módulo de nivel 1) y su velocidad de transmisión es 155520 Kbps[9].

Además de conseguir la digitalización de la red, es necesario que la parte correspondiente del usuario sea también digital. A este fin responde el estándar Red Digital de Servicios Integrados – RDSI. En él, se normaliza la parte de red del abonado, sin llegar a la interfaz usuario – red. En RDSI todos los terminales son digitales. Los accesos RDSI se basan en canales de 64 Kbps que permiten utilizar diferentes servicios. Estos pueden disponer de uno o varios canales. Existen dos tipos de acceso RDSI: Básico con 2 canales, y Primario con 30. Todo ello, sin duda, recuerda a PDH.

Con la evolución de los servicios de comunicaciones, el crecimiento Internet, etc., surgen nuevas aplicaciones antes no tenidas en cuenta, como son la videoconferencia entre varios usuarios, aplicaciones en tiempo real como radio o televisión por Internet, redes privadas virtuales, aplicaciones de Tecnologías de la Información – IT (Information Technology) para grandes empresas, etc. Este crecimiento hace que rápidamente resulten escasos los 64

[8]Cfr. Recomendación UIT-T G.803 (03/2000) – Arquitectura de redes de transporte basadas en la jerarquía digital síncrona.

[9]Para una referencia de los distintos niveles de módulos STM y sus velocidades asociadas cfr. Sexton, M. y Reid, A (1997), op. cit., pg. 55.

Kbps sobre los que se estructura la RDSI. De este modo se comienza a trabajar en un nuevo estándar: RDSI de Banda Ancha o RDSI-BA. RDSI-BA está pensada teniendo en cuenta todos los tipos de servicios y comunicaciones, tanto tradicionales y avanzados: Voz, videoconferencia, contenidos multimedia, etc. Tanto transmisiones de datos como transmisiones en tiempo real. Comunicaciones punto a punto o punto a multipunto, como radio o televisión.

Por último, sólo queda saber qué red será la que soporte y tenga la capacidad de transmisión requerida por RDSI-BA. Debe servir para todos los servicios. La respuesta que da la UIT da a esta pregunta es ATM[10].

## 2.2 INTRODUCCIÓN A LAS REDES ATM

Las redes ATM están normalizadas por la UIT. También son reconocidas las recomendaciones del ATM Forum, formado por la unión de empresas multinacionales del sector de las telecomunicaciones.

Las redes ATM se presentan como una solución ante la disparidad de redes, absolutamente dependientes del servicio que prestan. Así por ejemplo, se puede mencionar la Red Telefónica Conmutada para comunicaciones de voz, la Red X.25 para comunicación de paquetes, o la red de radiodifusión. ATM es el estándar que pretende conseguir una red única válida para servicios con características de tráfico y requisitos muy distintos.

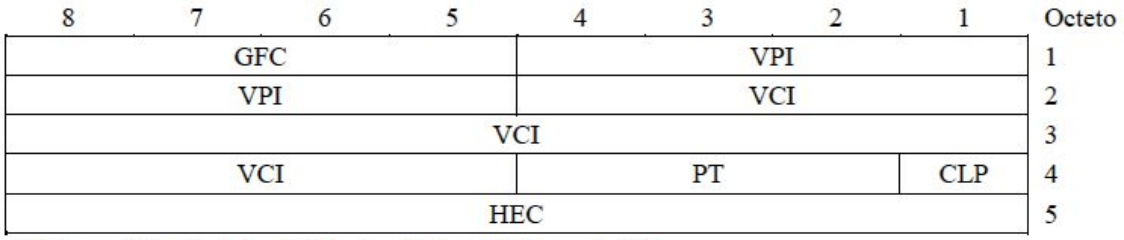
[10]Cfr. Recomendación UIT-T I.150 (09/1999) – Características funcionales del modo de transferencia asíncrono de la RDSI-BA. Pg. 1.

En el apartado 2 puede leerse: “El modo de transferencia asíncrono (ATM) es la solución en cuanto al modo de transferencia para implementar una RDSI-BA.”

### 2.2.1 LA CÉLULA, UNIDAD MÍNIMA DE DATOS EN ATM

ATM es una red en la que se transmiten de un modo asíncrono paquetes de información de tamaño fijo llamados células. La célula consiste en un paquete de 53 octetos, de los cuales 5 forman la cabecera y 48 el campo de datos.

La cabecera de célula varía según se trate de una interfaz UNI -*User to Network Interface*- o una interfaz NNI -*Network to Network Interface*. En las figuras 1 y 2 se ve la estructura para cada caso[11].




| | |
|---|---|
| CLP | Prioridad de pérdida de célula (*cell loss priority*) |
| GFC | Control de flujo genérico (*generic flow control*) |
| PT | Tipo de cabida útil (*payload type*) |
| HEC | Control de errores de encabezamiento (*header error control*) |
| VPI | Identificador de trayecto virtual (*virtual path identifier*) |
| VCI | Identificador de canal virtual (*virtual channel identifier* |

*Figura 1 Cabecera en la interfaz Usuario - Red*

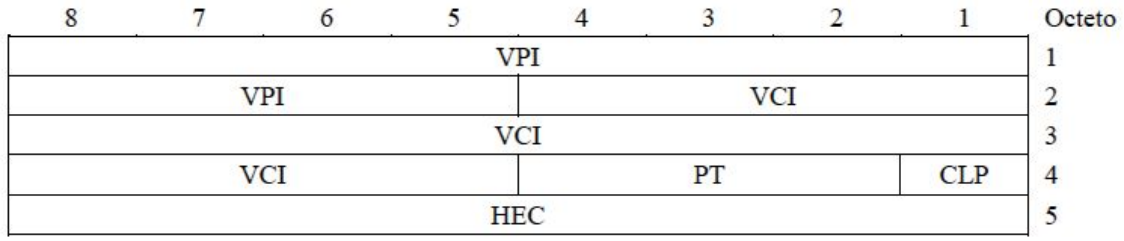



*Figura 2 Cabecera en la interfaz Red - Red*

[11]Tomada, al igual que las siguientes de la Recomendación UIT-T I.361 (02/99) – Especificación de la capa de modo asíncrono de la RDSI-BA. Pg 1 y ss.

Al comparar las dos figuras se observa que el campo VPI que en la UNI son 8 bits pasa a 12 en la NNI, y desaparecen los 4 bits del GFC. El sentido de esta diferencia es doble:

- ✓ El control de flujo sólo existe entre el usuario y la central a la que está conectado.
- ✓ Dentro de la red, en las NNI, se multiplexan multitud de células de diferentes pares origen-destino (diferenciables en cada interfaz por los pares VPI/VCI). Esto no ocurre en ninguna UNI. Por ello, en las NNI se encuentran muchos más valores de VPI en uso, respecto a las UNI.

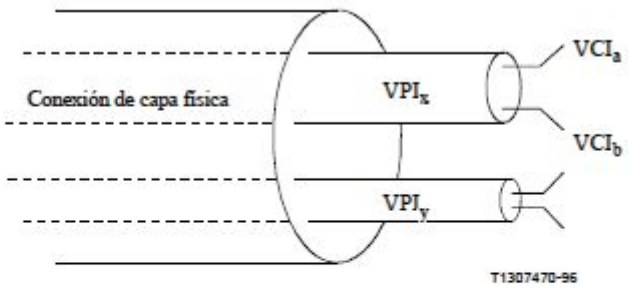
De los campos de la cabecera de las células, se analizan a continuación dos de ellos: el *Virtual Path Identifier* – VPI, y el *Virtual Circuit Identifier* – VCI[12].

### 2.2.2 TRAYECTOS Y CANALES VIRTUALES: CONEXIONES ATM

En ATM, una conexión "consiste en la concatenación de enlaces de capa ATM para proporcionar una capacidad de transferencia de extremo a extremo a puntos de acceso."[13]

[12]En castellano, identificador de trayecto virtual e identificador de canal virtual, respectivamente.

[13]Recomendación UIT-T I.150 (02/99), op. cit., pg. 2.




NOTA – VCI$_a$ y VCI$_b$ representan dos de los valores posibles del VCI dentro del enlace de VP de valor VPI$_x$. Igualmente, VPI$_x$ y VPI$_y$ son dos de los posibles valores del VPI dentro de la conexión de capa física.

*Figura 3 Identificadores VPI e VCI*

Cada concatenación de enlaces de la capa ATM la realiza un nodo. En cada enlace se distinguen tantos trayectos virtuales (VP) como VPI diferentes existan; y dentro de cada VP se distinguen tantos canales virtuales (VC) como VCI diferentes haya. Esto es lo que muestra la figura 3.

Por un lado, un VPI y un VCI tienen significado en cada enlace independientemente del resto, aunque esto requiere una matización posterior.

Por otro lado, y a grandes rasgos, los nodos tienen como función principal pasar las células que llegan por un enlace, con unos VPI/VCI concretos, a otro enlace, con unos nuevos VPI/VCI determinados por el enrutamiento.

La matización que se anunció es la siguiente: de acuerdo con la recomendación I.150 se pueden establecer conexiones de trayecto virtual – VPC, formado por una concatenación de enlaces en los que un VPI concreto identifica una conexión entre dos terminadores de trayecto virtual. Dentro de una VPC el valor VPI no puede ser cambiado.

De modo análogo, se pueden establecer conexiones de canal virtual – VCC, entre dos terminadores de canal virtual. Dentro de una VCC el valor VCI

no puede ser cambiado. A esto cabe añadir que dentro una VPC tampoco pueden cambiarse los VCI, como puede intuirse al ver la figura 3, mostrada anteriormente.

Esto nos lleva a establecer diferentes tipos de nodos. Por ejemplo, un terminador de trayecto virtual que no sea terminador de canal virtual sólo enrutará en función del VPI, pero ni si quiera leerá el VCI.

### 2.2.3 TIPOS DE TRÁFICO EN ATM

Distintos servicios, se caracterizan de modos distintos y presentan restricciones diferentes. Por ejemplo, el tráfico de voz es muy sensible al retraso de parte de la información, y prácticamente inmune a la pérdida de parte de ésta. Una red puede estar diseñada específicamente para un tráfico de estas características. Otro tipo de tráfico es el de comunicaciones de datos entre aplicaciones informáticas. En este caso, la información puede tener un retraso cualquiera, pero la pérdida de un paquete de información puede provocar la inutilidad del resto y/o el aumento de tráfico por retransmisiones.

#### 2.2.3.1 Parámetros y Descriptores de tráfico.

Como vemos, ATM presupone un estudio sobre caracterización y clasificación de los diferentes tráficos, provocado por los diferentes servicios, y que se pretenden soportar con ATM. Pero, yendo un poco más allá, la UIT plantea qué descriptores de tráfico usar para caracterizar los distintos tipos[14].

[14]Recomendación UIT-T I.371 (03/2000) – Control de tráfico y control de congestión en RDSI-BA, aptdo. 5 – Parámetros y descriptores de tráfico. Pg. 11 y ss.

En primer lugar, define “parámetro de tráfico” como “una especificación de un aspecto particular del tráfico. Puede ser cualitativo o cuantitativo.”[15]

A continuación define “descriptores de tráfico ATM”, como “la lista genérica de parámetros de tráfico que pueden utilizarse para captar las características de tráfico de una conexión ATM.”[16]

A renglón seguido define el “descriptor de tráfico fuente” como el subconjunto de parámetros de tráfico pertenecientes al descriptor de tráfico ATM, que ha sido “utilizado durante el establecimiento de la conexión para captar las características de tráfico intrínsecas de la conexión solicitada por la fuente.”[17]

Sin pretender abarcar completamente todas las definiciones restantes, hay que destacar que “los procedimientos de control de admisión de la conexión emplearán el descriptor de tráfico de fuente y las tolerancias de variación de retraso – CDV (*Cell Delay Vairation*)[18] asociadas, incluidas en el descriptor de tráfico de la conexión, para aceptar o rechazar solicitudes de conexión.”[19]

ATM no se especializa *a priori* en ningún tipo de tráfico. Acepta cualquiera. Evalúa en cada establecimiento de llamada si puede soportar esa comunicación, en función de los requisitos de calidad y CDV exigidos por la fuente.

---

[15]Idem.

[16]Idem.

[17]Idem.

[18]Su principal causa se encuentra en la multiplexación de células provenientes de distintas fuentes. Cfr. Rec. UIT-T I.371 (03/2000), op. cit., pg. 14.

[19]Rec. UIT-T I.371 (03/2000), op. cit.,. pg. 12.

En caso de ser aceptada una conexión, “la capacidad de transferencia ATM seleccionada (...), el descriptor de tráfico de fuente, las clases de calidad de servicio para una determinada conexión ATM y las tolerancias CDV asignadas al equipo de cliente (...) convenidas en la fase de establecimiento de la conexión definen el contrato de tráfico”[20].

Dos parámetros que la Recomendación señala para su especificación en cualquier descriptor de tráfico fuente son la velocidad de cresta (PCR, *peak cell rate*), y la velocidad de célula sostenible (SCR/IBT, *sustainable cell rate / intrinsic burst tolerance*)[21].

#### 2.2.3.2 Capacidades de transferencia ATM

Una capacidad de transferencia ATM – ATC (*ATM Transfer Capability*) tiene por finalidad soportar un modelo de servicio concreto, con unos descriptores de tráfico y parámetros, correspondiente a la calidad requerida por un determinado conjunto de aplicaciones[22]. Soportando diferentes ATCs, ATM es una red capaz de soportar tráficos diferentes de distintos tipos de comunicación, como pueden ser voz, datos, o difusión.

La ATC es solicitada por el usuario antes del establecimiento de la conexión, indicando el descriptor de tráfico fuente y tolerancias deseadas. Una vez que la conexión es aceptada, esa capacidad de transferencia es la misma en todas las interfaces normalizadas a lo largo de la conexión[23].

[20] Idem.

[21] Idem, pg. 17.

[22] Cfr. Rec. I.371 (03/2000). Pg. 26.

[23] Idem.

La norma I.371 (03/2000) de la UIT especifica cuatro capacidades de transferencia diferentes:

- ✓ Capacidad de transferencia determinista – DBR (*Deterministic Bit Rate)*. Su finalidad es satisfacer requisitos de calidad de tráfico de velocidad binaria constante con compromisos de calidad – QoS (*Quality of Service)* en términos de tasa de pérdida de células – CLR (*Cell Loss Ratio)*, retardo de transferencia de células – CTD (*Cell Transfer Delay)* y variación del retardo de transferencia de células – CDV. En definitiva, está pensado para cualquier aplicación que implique tráfico de tasa constante – CBR (*Constant Bit Rate).*

- ✓ Velocidad de transferencia estadística – SBR (*Statistic Bit Rate*). Puede no haber compromiso en cuanto al retardo, pero sí se establecen en el contrato de tráfico unos SCR e IBR determinados.

- ✓ Transferencia de bloques ATM – ABT (*ATM Block Transfer*). Esta capacidad establece una división en bloques del tráfico fuente, y establece unos compromisos estáticos para todos los bloques y otros dinámicos renegociables para cada bloque.

- ✓ Velocidad binaria disponible – ABR (*Available Bit Rate*). Como claramente explica la norma, la capacidad de transferencia ABR "está destinada a soportar aplicaciones elásticas que pueden adaptarse a la anchura de banda instantánea disponible dentro de la red y no tienen requisitos de retardo estrictos". Se puede pensar, por ejemplo, en tráfico de tipo HTTP (*Hyper Text Transfer Protocol*).

### 2.2.4 OPTIMIZACIÓN EN ATM

Toda red es objeto de estudios en orden a su optimización, entendiendo ésta como el modo de maximizar el uso de la red manteniendo una calidad de servicio; o si se prefiere, minimizar el coste de operación manteniendo la calidad.

En ATM este estudio se hace más complicado en cuanto que la red debe soportar diferentes servicios con restricciones variables, como se explicó anteriormente.

Una solución habitual en la práctica real pone el acento en la diferencia de tráficos y afirma que un modo natural de configuración para lograr las garantías del servicio es controlar cada tipo de tráfico individualmente y aislarlos. Que hubiera cambios en el comportamiento de un tipo de tráfico no debería afectar negativamente a los demás.[24] Muchos autores coinciden en establecer buffers separados para los diferentes tipos de tráfico[25].

Un hecho a favor de la separación de buffers es que para no introducir grandes retardos -como demandan DBR y a veces SBR- se necesitan buffers que no sean muy grandes, a costa incrementar las pérdidas. En cambio el tráfico de datos, insensible a los retardos, requiere que sean grandes para que haya pocas pérdidas.

Otra cuestión del problema de la optimización es el reparto de la capacidad de un nodo que debe atender diferentes tipos de tráficos, generalmente sensibles a las pérdidas e insensibles al retardo, o viceversa.

[24]Cfr. Balakrishnan, M., et al (1997): "Buffer losses vs. deadline violations for ABR traffic in an ATM switch: a computational approach". *Telecommunications Systems,* No. 7, pg. 106

[25]Entre otros: A. Demers, S. Keshav, S. Shenker, P.E. McKenney, A.K. Parekh y R.G. Gallager.

Una forma de abarcar este problema es formularlo mediante un modelo de programación lineal, que puede ser conceptualizado como una generalización del problema de la mochila con restricciones de capacidad[26].

En definitiva, plantear la optimización en ATM es plantear la pregunta de qué algoritmos de encaminamiento y qué directivas de servicio en los nodos son los que alcanzan un mayor rendimiento en la red: mínimo coste garantizando la calidad.

### 2.2.5 ESTUDIO DEL TRÁFICO EN ATM: EL TRÁFICO FRACTAL

Junto con el problema de la optimización, surge la cuestión relativa a que modelo emplear para caracterizar el tráfico en una red ATM.

En la Red Telefónica Conmutada el tráfico está perfectamente caracterizado mediante el modelo de Erlang. Se trata de tráfico de voz por conmutación de circuitos. Pero en ATM, como ya se ha visto, no se tiene tráfico de una única naturaleza, sino de varias. Por otro lado, se puede entender que sí hay conmutación de circuitos (VCC), pero todo tráfico se descompone en células, que procedentes de distintas fuentes se multiplexan en los nodos con otras células pertenecientes a tráficos muy distintos, como puede ser el ABR. Esto hace que se introduzcan retardos y pérdidas[27].

---

[26]Youngho, L. et. al. (2001): "An ATM switch capacity allocation problem". *Telecommunications Systems*, No. 18:4, pg. 301. Se propone una formulación del problema mediante programación lineal. Luego usan búsqueda tabú para encontrar una buena solución marcando un valor objetivo al 2% del óptimo. Con ello reducen considerablemente el tiempo de resolución.

[27]Cfr. UIT-T I.371 (03/2000), op. cit., aptdo. 5.3.5. "Efecto de la variación del retardo...", pgs. 14 y 16.

En definitiva, las dificultad consiste en caracterizar la naturaleza del tráfico de una red ATM, debido a la confluencia de tráficos muy distintos. Además, el modelo ha de servir para cualquier punto de la red ATM. Si existe esa naturaleza única, al multiplexar distintos agregados de tráfico ATM, la caracterización debe seguir siendo válida. Independientemente del tipo de servicio que provoque cada tráfico. Con esto se llega a que ese modelo ha de ser también válido para caracterizar el tráfico que provoca cualquier servicio soportado por ATM; es decir, todos -voz, SBR, ABR, etc.

Varios autores introdujeron un nuevo modelo que cumple con todos estos requisitos. Se trata del modelo de Tráfico Fractal[28], que se describe brevemente a continuación.

## 2.2.6 BREVE EXPOSICIÓN DEL MODELO DEL TRÁFICO FRACTAL[29]

### 2.2.6.1 Preliminares

Este modelo está precedido de estudios basados en mediciones de tráfico en redes de área local con fuentes de diversos comportamiento, así

[28]En literatura inglesa es denominado *Self-Similar Traffic*. El modelo ha sido desarrollado por Boris Tsybakov, Qualcomm Inc.; Nicolas D. Georganas, Multimedia Communications Research Laboratory (MCRLab).

Sobre este tema pueden consultarse de estos autores los siguientes artículos:

Tsybakiov, B y Georganas, N. D. (1997): "On Self-Similar Traffic in ATM Queues: Definitions, Overflow Probability Bound, and Cell Delay Distribution", *IEEE/ACM Transactions on networking*, vol. 5, nº 3.

Tsybakiov, B y Georganas, N. D. (1999): "Overflow and loss probabilities in a finite ATM buffer fed by self-similar traffic", *Queueing Systems*, No. 32, pgs. 233-256.

[29]Extraído de Tsybakiov, B y Georganas, N. D. (1997), op. cit.

como en análisis de cientos de millones de paquetes de datos en redes Ethernet en laboratorios de la compañía estadounidense Bellcore.

Para describirlo se define X como el segmento semi-infinito de un proceso estocástico con covarianza constante en el tiempo y argumento discreto *t* ≥ *1*. También se define $X^{(m)}$ como el segmento semi-infinito de un proceso estacionario de segundo orden y argumento discreto también *t* ≥ *1*. También se define x como sigue. Matemáticamente sería:

$$X = \{X_t\} = (X_1, X_2, X_3, \cdots)$$

$$x = \{x_t\},, \text{donde } x_t = X_t - \mu\ ,, \mu = E[X_t]$$

$$X^{(m)} = \{X_t^{(m)}\} = [X_1^{(m)}, X_2^{(m)}, X_3^{(m)}, \cdots],, \text{donde } X_t^{(m)} = \frac{1}{m}\sum_{j=1}^{m} X_{t(j)}\ ,, m \in I_2 = \{2,3,\cdots\}$$

Las propiedades establecidas en la descripción del proceso X implica varianza constante. Junto con esto se usarán las funciones de correlación de los procesos dados. Matemáticamente:

$$\text{var}(X) = \text{var}(x) = \sigma^2 = E[x_t^2]$$

$$\text{Autocorrelación de X y de x: } r(k) = \frac{E[x_t x_{t+k}]}{E[x_t^2]}$$

$$\text{Autocorrelación de } X^{(m)} = r^{(m)}(k)$$

A continuación se definen para su posterior empleo la función g(k), el parámetro de Hurst y el concepto de función con variación lenta – SVF (*Slowly Varying Function*).

$$g(k) \equiv \frac{1}{2}\left[(k+1)^{2-\beta} - 2k^{2-\beta} + (k-1)^{2-\beta}\right],, \beta \in (0,1)$$

$$\text{Hurst parameter: } H = 1 - \frac{\beta}{2}$$

$$\text{Definición de sowly varying function } L(k):$$

$$\frac{L(xt)}{L(t)} \xrightarrow[t\to\infty]{} 1\ ,, \forall x > 0$$

### 2.2.6.2 Definiciones y Teoremas de los procesos exactamente fractales de segundo orden – *esoss.*[30]

El proceso X es llamado esoss. con parámetro Hurst H=1-β/2 si:

$$\mathrm{var}\left[X^{(m)}\right] = \sigma^2 m^{-\beta} \qquad (1)$$

$$r^{(m)}(k) = r(k) \quad ,, \forall k \in I_0 \quad \text{y} \quad \forall m \in I_2 \qquad (2)$$

$$r(k) \propto k^{-\beta} L(k) \quad ,, \text{donde L(k) es una svf.} \qquad (3)$$

A esta definición hay que aplicar algunos teoremas con los que se comprueba que las 3 condiciones no son independientes entre sí.

### 2.2.6.3 Teoremas que siguen a la definición dada

**Teorema 1**

X satisface la condición (1) **si y sólo si**

$$r(k) = g(k) \qquad (4)$$

Luego (1) y (4) son equivalentes.

**Teorema 2**

**Si** r(k)=g(k) **entonces**

$$\lim_{k\to\infty} \frac{r(k)}{k^{-\beta}} = (2-\beta)(1-\beta) = H(2H-1) \qquad (5)$$

Esto implica que la SVF. de la condición (3) es una constante explícita dependiente del parámetro β.

---

[30] En inglés, *exactly second-order self-similar process*.

La definición que sigue es acorde con D. R. Cox (1984), "Long-range dependence: A review", en *Statistics: An Appraisal,* David, H. A. y David, H. T (eds.). Ames, IA: Iowa State Univ. Press. Iowa, pgs. 55-74.

**Teorema 3**

**Si** se cumple (1) o (4) **entonces**

$$r^{(m)}(k) = r(k) \quad ,,k \in I_0, m \in I_2 \qquad (6)$$

Luego por tanto, (1) implica (2). Lo mismo con (4).

### 2.2.6.4 Definición simplificada

Aplicando los teoremas vistos, es posible establecer una definición de proceso esoss más simplificada.

El proceso X es llamado esoss. con parámetro Hurst H=1-β/2 si:

$$r(k) = g(k) \equiv \frac{1}{2}\left[(k-1)^{2-\beta} - 2k^{2-\beta} + (k+1)^{2-\beta}\right] \qquad (7)$$

### 2.2.6.5 Definiciones y teoremas de los procesos asintóticamente fractales de segundo orden – asoss[31].

El proceso X es llamado asoss. con parámetro Hurst H=1-β/2 si:

$$\lim_{m\to\infty} r^{(m)}(k) = g(k) \equiv \frac{1}{2}\left[(k-1)^{2-\beta} - 2k^{2-\beta} + (k+1)^{2-\beta}\right] \qquad (8)$$

[31] En inglés *asymptotically second-order self-similar process*.

### 2.2.6.6 Teoremas derivados de esta definición

**Teorema 5**

Si

$$\lim_{k\to\infty} \frac{r(k)}{k^{-\beta}} = c \quad ,, \text{donde c es una cte.}, 0 < c < \infty \qquad (9)$$

entonces,

$$\lim_{k\to\infty} \frac{\text{var}(X^m)}{m^{-\beta}} = c \quad ,, \text{donde c también es una cte.} \quad (10)$$

**Teorema 6**

**Si** se cumple (12) **entonces:**

$$\lim_{m\to\infty} r^{(m)}(k) = g(k) \quad ,, k \in I_1 \quad (11)$$

Se aprecia aquí una importante semejanza entre esoss. y asoss., dada por los teoremas 2 y 5, concretamente, las ecuaciones (5), (9) y (11). En ambos, el límite es una constante; en ambos $r(k) \sim c \cdot k^{\beta}$, donde c es una cte., pero en esoss. $c=H(2H-1)$.

**Teorema 7**

**Si** X' y X'' son dos procesos no correlados en los que se cumple

$$r(k) \underset{k\to\infty}{\propto} c_1 k^{-\beta_1} \quad \text{para } X',$$
$$r(k) \underset{k\to\infty}{\propto} c_2 k^{-\beta_2} \quad \text{para } X''$$

**entonces** (X'+X'') es un asoss. con parámetro $H=1-\beta/2$ y $\beta=\min(\beta_1, \beta_2)$.

Es decir, la mezcla de procesos asoss., como la multiplexación de tráfico ATM en un switch da como resultado otro procesos que también es asoss.

**Teorema 8**

Sean X' y X'' dos procesos esoss. no correlados, X' con $H_1$, y X'' con $H_2$.

**Si** $H_1=H_2=H$, **entonces** X'+X'' es un proceso esoss. de parámetro H.

**Si** $H_1 \neq H_2$, **entonces** X'+X'' es un proceso asoss. de parámetro $H=\max(H_1,H_2)$; o lo que es lo mismo, $\beta=\min(\beta_1, \beta_2)$.

### 2.2.7 ARGUMENTOS DEL MODELO DE TRÁFICO FRACTAL

El modelo de tráfico fractal para caracterización del tráfico ATM permite la simulación de fuentes que generan tráfico, en las que variando unos pocos parámetros se pueden simular tráficos de distintas naturalezas: navegación por Internet, voz, imagen, etc.

Por otro lado, el tráfico agregado formado por la multiplexación de distintos flujos de tráfico fractal continúa manteniendo la naturaleza de tráfico fractal, y sólo varían los parámetros del mismo[32]. Es por ello, que en cualquier punto de la red el tráfico es fractal, siempre y cuando las fuentes generen tráfico fractal. Esto queda demostrado con los teoremas 7 y 8, según los cuales, en cualquier punto de la red se encontrarán procesos asintóticamente fractales de segundo orden.

[32]Cfr. Tsyakov, B. y Georganas, N. (1997), op. cit.

# 3 SOLUCIÓN PROPUESTA PARA EL MODELADO DE REDES ATM EN VISTAS A SU SIMULACIÓN

## 3.1 LOS PRINCIPIOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS COMO PUNTO DE PARTIDA

En la Programación Orientada a Objetos se pueden distinguir los siguientes principios[33]:

1. Cualquier cosa es un objeto.
2. Un programa es un conglomerado de objetos.
3. Cada objeto se compone de otros objetos.
4. Todo objeto pertenece a una clase de objetos.
5. Todos los objetos de una misma clase pueden recibir los mismos mensajes o entradas.

Todas las clases (de objetos) tienen sus propios datos (atributos) y sus propias funciones (métodos). Estas reciben determinados mensajes, que son propios de ese objeto.

La idea consiste en describir una red ATM desde un punto de vista "orientado a objetos". Así, se puede entender una red ATM como un conglomerado de objetos clasificables funcionando de un modo independiente, y que interactúan entre sí según estén conectados.

Así por ejemplo, una clase de objetos sería un nodo de red. Un nodo se podría descomponer en otras dos subclases de objetos: cola y servidor. Toda

[33]Cfr. Bruce Eckel (2000), *Thinking in Java*, Prentice Hall, pgs. 31 y 32.

cola recibe células, establece una disciplina de servicio, decide qué células se descartan, etc. Todo servidor recibe células ATM, procesa sus cabeceras, etc.

Igualmente, se podría modelar la clase fuente, enlace, nodo de admisión, incluso célula.

Una vez definidas las clases y subclases, la creación de objetos de esas clases podría ser flexible, de modo que, por ejemplo, se pudieran crear unos enlaces sí y otros no, estableciendo topologías diferentes. Incluso se podría programar la vida y destrucción de los objetos (nodos, enlaces, fuentes…) y simular la aparición y desaparición de fuentes, y la disponibilidad de enlaces y nodos, o también tasas de errores.

Otro aspecto implícito en el método buscado es la modularidad, entendiendo que es una característica muy deseable para un simulador de redes ATM. Un modelo de red supone tantos modelos como clases de objetos. Si se ven estas clases como cajas negras con entradas y salidas, su funcionamiento interno puede ser variado, mejorado, en definitiva cambiado, sin que ello afecte al resto del simulador. Muy deseable en la medida en que un simulador implica el problema de modelar cada elemento de red. Otro problema en el que no existe una solución única, ni en la descripción cualitativa ni en la transformación de esa descripción en código de programa.

Para la elaboración del simulador se pueden establecer los siguientes pasos:

- ✓ Determinar cuáles son las clases de objetos en una red ATM.
- ✓ Determinar las funciones que realiza cada clase de objeto y los datos que usa, sean parámetros de configuración o información que procesa.

- ✓ Programar cada clase.
- ✓ Programar el simulador haciendo uso de las clases programadas.

## 3.2 CLASIFICACIÓN DE OBJETOS DE UNA RED ATM Y SUS FUNCIONES

### 3.2.1 LA CÉLULA

Una célula, como ya se dijo en el apartado 1.1.1, tiene una primera división: cabecera, con 5 bytes (40 bits), y campo de datos con 48 bytes. Estableciendo la cabecera y el campo de datos como objetos en sí mismos, un objeto célula está compuesto por un objeto cabecera y por un objeto campo de datos.

Los elementos propios de una red ATM nunca harán operaciones sobre el campo datos de aquellas células originadas por usuarios de la red. No así si se trata de células de tráfico de gestión de red, creadas por elementos de ésta, como se verá más adelante. En ambos casos, se pueden establecer dos operaciones básicas sobre la clase de objetos “campo de datos” de célula: leer y escribir.

La clase de objetos “cabecera de célula” se puede entender simplemente como un array de 40 bits, en los que en general se realizan las operaciones de leer y escribir. No obstante, se pueden especializar estas dos operaciones y concretarlas más. Es decir, son operaciones típicas sobre una cabecera de célula: Leer y escribir el campo VPI, o el VCI, o el PT, o el bit CLP. En definitiva, resulta útil especificar una función para leer y otra para escribir para cada campo de la cabecera.

Atendiendo a lo dicho, se tiene que las funciones leer y escribir como tales, son operaciones típicas tanto en la cabecera como en el campo de datos. Por tanto será más propio especificarlo como operación sobre célula.

Por último, las operaciones de leer es lógico que devuelvan el resultado en decimal, por lo que una operación será pasar de binario a decimal. Y al revés para operaciones de escritura.

## 3.3 NODOS DE RED

En una red ATM, se puede hacer una primera clasificación de nodos de red:

- ✓ Nodos NNI o nodos internos: Todas sus interfaces son NNI, y por tanto, no tienen conectados bucles de abonados.
- ✓ Nodos UNI o centrales de usuarios: Son el punto de entrada de los abonados a la red. Todas sus interfaces son UNI, salvo una NNI que estará conectada a un nodo NNI.

### 3.3.1 NODOS NNI

Esta clase de objeto es, junto con la clase abonado, y si se quiere también la clase enlace, la más importante de cara a hacer el simulador lo más real posible. Los nodos son uno de los puntos clave en la red. En ellos se multiplexan los agregados de tráfico, provocando una parte importante de la pérdida de células en la red, y el retardo en las que no se pierden. En los nodos se llevan a cabo decisiones de encaminamiento cuando el algoritmo que se usa es local, no global. En los nodos se implementan las políticas de servicio para los diferentes tipos de tráfico.

En este proyecto no se lleva a cabo la implementación perfectamente detallada del nodo, pero sí lo suficientemente concisa como para mostrar las posibilidades que tiene la metodología seguida respecto al problema de simulación, en orden a los estudios de optimización de redes ATM.

Para la especificación, descomposición en partes y descripción de los nodos NNI se usa el modelo mostrado en la figura 4.[34]

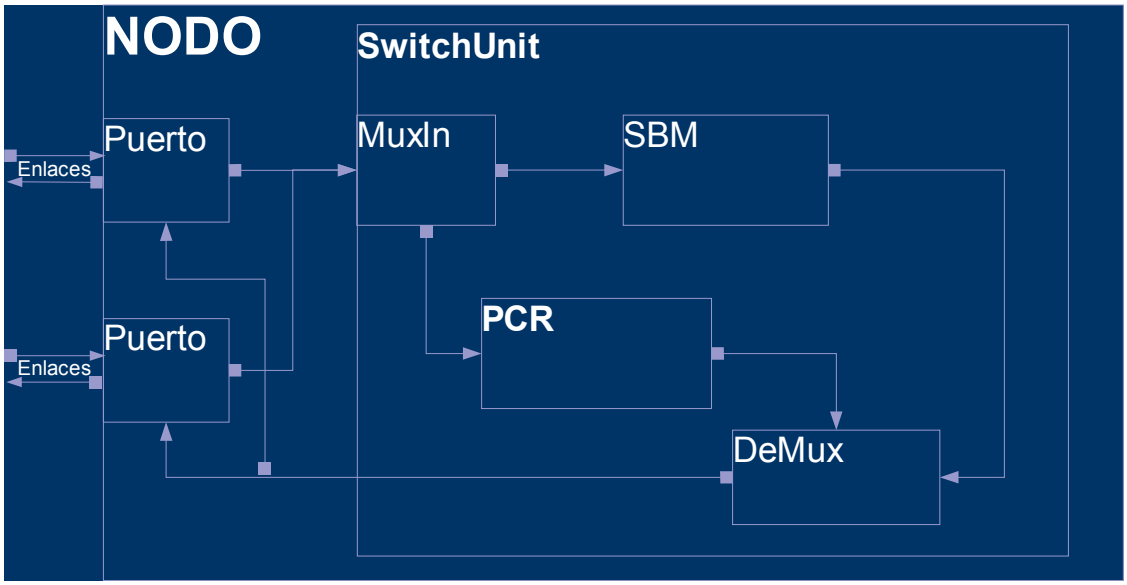



*Figura 4 Estructura interna del nodo*

El puerto en un nodo recibe células del enlace que tiene conectado y se lo pasa al SwitchUnit[35], o unidad de conmutación, y viceversa. En la figura 4 se

[34]Tomado y simplificado de Heo, J. W. et. al. (2000): "Cell-level/call-level ATM switch simulator", *Telecommunication Systems,* No. 14, pgs. 291 a 309.

[35]Todos los nombres en lengua inglesa están tomados del citado artículo. Por respetar la nomenclatura usada en el mismo, se usarán en su lengua original, aunque serán traducidos libremente en la medida en que aparezcan.

dibujan dos para indicar que son varios, frecuentemente en mayor número, todos ellos conectados al SwitchUnit.

Cabe preguntarse si en los puertos es posible la pérdida de células. Ciertamente los puertos cuentan en la realidad con un buffer de entrada y otro de salida.

En este proyecto el diseño del puerto se simplifica no proveyéndolo de buffer de entrada ni salida. Esto equivale a pensar que el enlace no es capaz de provocar el desbordamiento del buffer de entrada. Por el contrario, como se verá más adelante, se supondrá que el nodo puede provocar sobre escritura en el enlace. Es decir, el buffer queda asociado al enlace.

#### 3.3.1.2 SwitchUnit

Esta clase de objeto está presente en todo nodo. En él se realizan todas las operaciones del nodo.

En la figura 4 anterior se muestra una versión simplificada del modelo que presentado a continuación en la figura 5.[36]

[36]Tomada de Heo, J. W. et al. (2000), op. cit.

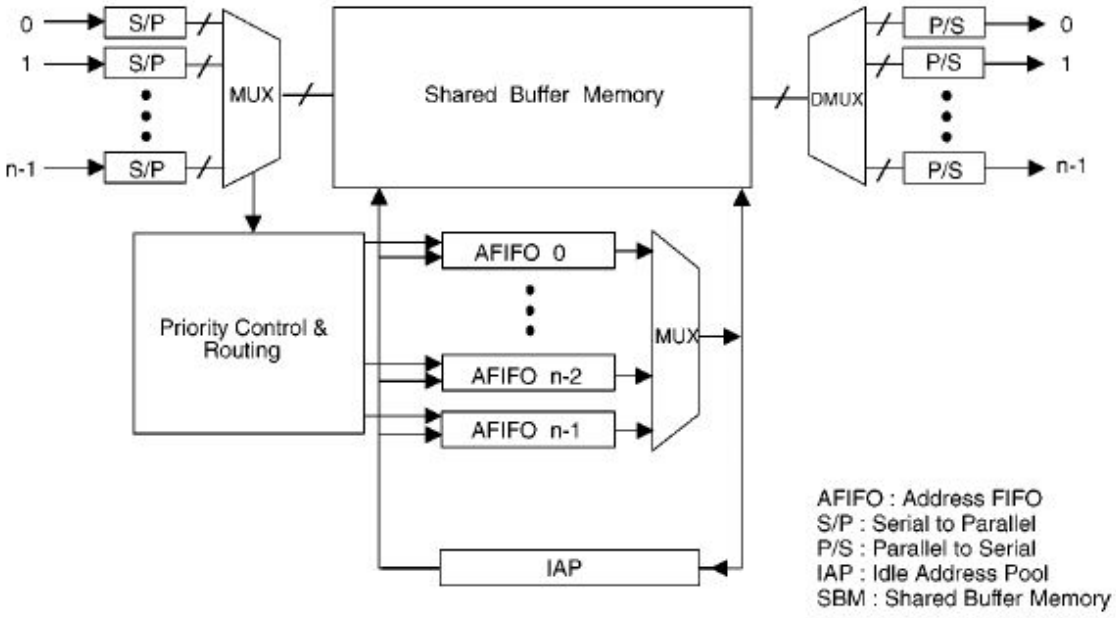



Figure 2. Switch element of shared buffer architecture.

*Figura 5 Unidad de conmutación del nodo*

De cara a la simulación se ha considerado innecesaria la inclusión de los convertidores paralelo/serie y viceversa que se observan en el modelo original. No tendrían sentido en el simulador, pues la transmisión bit a bit no se implementa. Sería necesario simular el mismo comportamiento de la señal eléctrica y los buses del hardware, y esto excede el ámbito de la simulación pretendida.

Otro elemento suprimido en el simulador es el IAP, cuya función es proveer de un mecanismo de relleno para cuando no hay células que atender. El funcionamiento de la cola, y concretamente del algoritmo de salida de la cola hace una función equivalente, como se verá.

Por otro lado, las funciones de multiplexación y demultiplexación que se ven también han sido simplificadas y o incluso puestas con carácter simbólico, en vistas a lo que es el objetivo y el alcance de este proyecto. Por el mismo motivo se han suprimido los buffers AFIFO – Address FIFO.

El funcionamiento del SwitchUnit es el siguiente[37]: El Multiplexor de entrada recibe por un puerto identificable una célula, y envía la cabecera al controlador de enrutamiento – PCR (*Priority and Control Routing*). Éste almacenará la cabecera con el identificador de puerto en la cola que tiene, siempre que sea posible, y le indicará al multiplexor en qué posición ha quedado almacenada. Por último, el multiplexor almacenará los datos de la célula en el buffer de memoria compartida – SBM (*Shared Buffer Memory*), en la misma posición en que el PCR. La cabecera de la célula queda en la cola del PCR en espera de ser procesada.

Cuando la cabecera es procesada, el PCR decide qué nuevo valor VPI/VCI ha de contener, y por qué puerto ha de retransmitirla. Tras configurar la cabecera, le anexa los datos tomándolos del SBM. Acto seguido le pasa al demultiplexor la nueva célula y el puerto por el que ha de ser retransmitida. El Demultiplexor de salida hará que sea transmitida por el puerto.

**Multiplexor de Entrada y Demultiplexor de Salida**

El modelo original realiza multiplexación en el tiempo. Por simplificaciones que se tratarán en la descripción del simulador, en este proyecto, multiplexor y demultiplexor tienen unas funciones mínimas, que ya han sido descritas. No obstante, su presencia como clases de objetos permite que dada la modularidad del método de trabajo puedan ser perfeccionados sin afectar a las demás clases.

[37]Cfr. Heo, J. W. et al. (2000), op. cit., pgs. 294 y 295.

### Shared Buffer Memory – SBM

Este elemento no es más que un buffer de memoria con tantas posiciones como tenga la cola del nodo, que se haya en el PCR. Complementa a la cola, almacenando los datos de la célula, mientras que aquélla almacena las cabeceras. Sobre estos datos no se realiza ninguna función de lectura ni escritura. Sólo se hace un ingreso en el SBM cuando la cola ha admitido el ingreso de la cabecera. Si por el contrario la cola rechaza la cabecera entrante, la célula es descartada y sus datos nunca son almacenados en el SBM.

### Priority and Control Routing – PCR

Recoge las cabeceras de las células, las almacena en la cola, que él posee y gestiona, junto con la información de por cuál de los puertos llegó. Periódicamente saca una cabecera de la cola para procesarla, aplica el enrutamiento, y se la remite al demultiplexor, indicando por qué puerto transmitirla y en qué posición del SBM están los datos correspondientes, tal y como se explicó y muestra la figura 6.

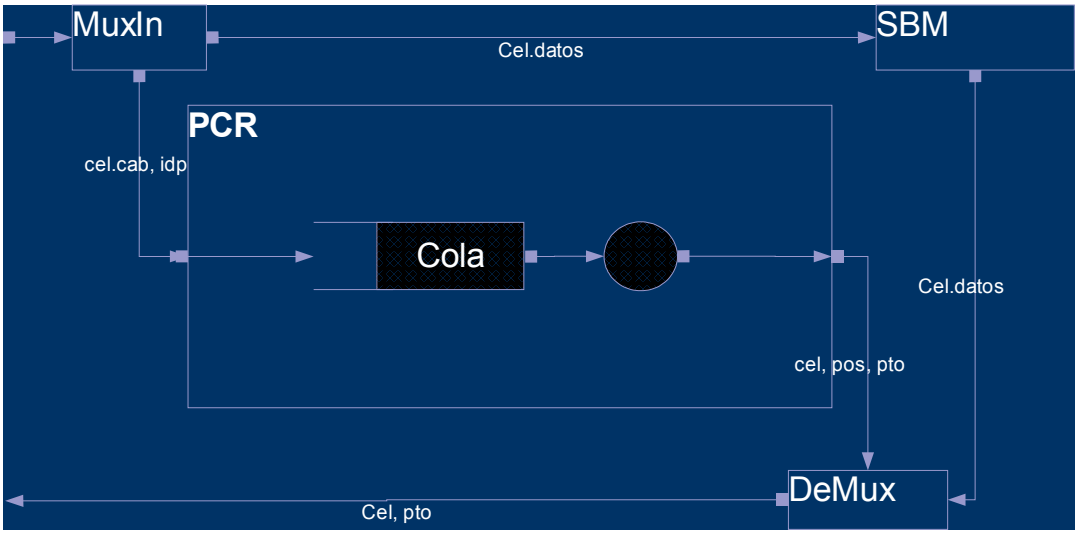



*Figura 6 Esquema del Switch Unit*

- ✓ En parte funciona autónomamente, en cuanto que procesa células. El procesamiento es independiente de la llegada. En otro caso, no tendría ningún sentido la existencia de una cola. No se puede simplificar este aspecto, pues es el lo que dota a un objeto nodo de funcionamiento independiente al resto de los objetos, de la misma clase o de otro tipo.

El nodo, por tanto, almacena información de enrutamiento, y en el caso de políticas de enrutamiento local, tendrá los procedimientos pertinentes.

La cola se puede considerar una clase de objeto específica, y el nodo, más concretamente el PCR, contendrá una, o varias, si se quiere hacer separación de tráfico, tal y como se vio en el estudio de redes ATM al principio. Por la importancia que tiene, sigue un epígrafe dedicado al tipo de objetos cola.

**La cola del PCR**

En primer lugar, una cola es un buffer de memoria, con una capacidad concreta, donde generalmente se sigue una política FIFO (*First Input, First Output*[38]), aunque esta disciplina puede variarse. De este modo, habrá una cola circular, donde los punteros o índices que indican la siguiente o última posición para sacar o introducir varían circularmente. Tres parámetros esenciales quedan así introducidos:

- ✓ Un puntero a la posición de la próxima cabecera a procesar, o bien a la posición de la última cabecera que se procesó.

[38]El primero que llegó sale primero.

- ✓ Un puntero a la posición donde se debe guardar la próxima cabecera que se almacene en la cola, o bien a la posición de la última cabecera que entró en la cola.
- ✓ Una constante que indica el tamaño de la cola. Cuando uno de los dos punteros alcanza este valor, ha de ser reinicializado para que continúe desde la primera posición.

En la cola se distinguen las siguientes operaciones:

- ✓ Entrada de un nuevo dato. Típicamente llamada *push* en la literatura de habla inglesa. Implica la gestión del puntero de entrada y su reinicialización si es el caso.
- ✓ Salida del siguiente dato. Conocida en inglés por *pop*. Implica la gestión del puntero de salida y su reinicialización si es el caso.
- ✓ Cálculo del número de datos almacenados en la cola y posiciones libres. Esta operación se puede entender como interna de la cola sobre sí misma y se hace en función del valor de los punteros.

Las operaciones sobre la cola tienen una exigencia crítica cuyo incumplimiento puede provocar la inutilidad del nodo: Se tiene que garantizar que nunca en ningún instante puedan estar ejecutándose dos operaciones simultáneas sobre la misma cola. Si no se garantizara, tampoco se podría garantizar la preservación de los datos mientras están en la cola, es decir: los punteros podrían dar valores incorrectos provocando la sobreescritura de datos almacenados, y sin mecanismos posibles que lo detectaran.

Por tanto, hay que implementar mecanismos de bloqueo de cola, que aseguren la atomicidad de las operaciones que se realicen sobre ella.

### 3.3.2 CENTRALES DE ABONADOS

En primer lugar cabe destacar la gran semejanza que existe entre el nodo y la central. Se puede entender como una variante de nodo que carece de funciones de enrutamiento, y por otro lado, tiene funciones propias de la interfaz UNI. Estas son, principalmente, algoritmos de control de admisión de la conexión, y algoritmos para control de tráfico de conexiones aceptadas, que vigilen el cumplimiento del contrato de tráfico establecido en la admisión.

La simplificación más arriesgada de las que se hacen en este proyecto está en las centrales. No se le da un funcionamiento propio, independiente del resto de los objetos. Consecuentemente, no tiene cola, ni PCR, ni SBM, ni en definitiva tiene un SwitchUnit. Además sólo está conectada a un nodo NNI, cosa que por otro lado no es irreal. En definitiva, no está modelado como objeto autónomo que se ejecute por su cuenta. Hay que analizar hasta qué punto esta simplificación puede ser grave.

En primer lugar, la central no es el elemento en el que más haya que fijarse. Los objetos esenciales para el análisis y modelado de una red son los nodos, los enlaces, y las fuentes de tráfico.

Por otro lado, si se desea que la central no tenga las simplificaciones que dichas, bastaría con colocar por cada central un conjunto Central-Nodo NNI. Se puede considerar este par como un objeto del tipo “central no simplificada”.

Para hacerlo más real, se podría asegurar que el enlace central-nodo no tuviera pérdidas. Bastaría conque su velocidad fuese mucho mayor que la de los demás objetos de la red.

## 3.4 ENLACES

Los enlaces en las redes ATM son otro punto clave, ya que junto con las colas de los nodos, es donde se producen la práctica totalidad de las pérdidas. Es por eso que en este proyecto se modela el enlace con un buffer de entrada para cada sentido de la transmisión, en el que se pueden producir pérdidas por sobre escritura.

Por otro lado, y del mismo modo que en la cola de los nodos, el buffer del enlace no tiene sentido si el enlace no funciona con un tope de velocidad, independientemente del resto de los objetos a los que está conectado.

El modelo de enlace que se desarrolla tiene así dos elementos que ya han sido introducidos en el PCR del SwitchUnit:

- ✓ Un buffer de entrada equivalente a una cola como la de los nodos pero de tamaño la unidad. Por tanto los punteros quedan sustituidos por un flag que indica si el buffer está libre o no.
- ✓ Un servidor que sin procesar las células -pues se trata de un enlace- las pasa de un extremo a otro. Este servidor es, como en el nodo, lo que da un funcionamiento autónomo e independiente al enlace, respecto al resto de los elementos de red.

El modelo que se empleará está reflejado en la figura 7.

Como se ve, el enlace ATM es doble, o si se prefiere, unidireccional. Esto se modela así por dos razones:

- ✓ Los enlaces, a nivel físico, son muchas veces unidireccionales. Es el caso, por ejemplo de la fibra óptica.

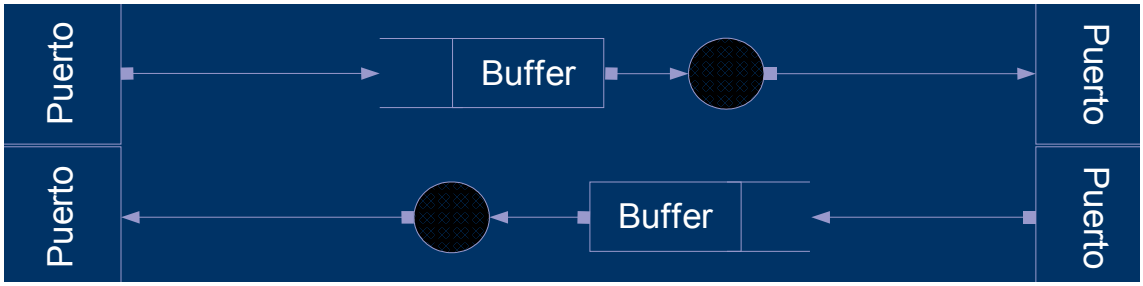



*Figura 7 Esquema del enlace*

- ✓ La UIT establece el concepto de conexión ATM con sentido unidireccional: "En esta Recomendación UIT-T, y en aras de la coherencia con las Recomendaciones UIT-T I.150 e I.113, las conexiones ATM son unidireccionales. Dos conexiones ATM se asocian para los dos sentidos de transmisión de una comunicación y se identifican por el mismo VPI/VCI en una interfaz dada"[39].

### 3.4.1 ENLACES DE RED VS. BUCLES DE ABONADO

Estructuralmente son dos clases de objetos idénticas, en las que cabe destacar dos diferencias no esenciales:

- ✓ La capacidad de transmisión. En los enlaces de red la velocidad será mucho mayor.
- ✓ Los bucles de abonado conectan puntos de acceso de la red -interfaz UNI- con centrales, mientras que los enlaces de red conectan siempre dos interfaces NNI.

Esta gran semejanza con pequeñas diferencias tendrá un efecto notable en la programación del simulador.

[39]Rec. UIT-T I.371 (03/2000), op. cit., pg. 7.

## 3.5 ABONADOS

En la descripción de los abonados este proyecto distingue dos capas que se muestran en la figura 8.

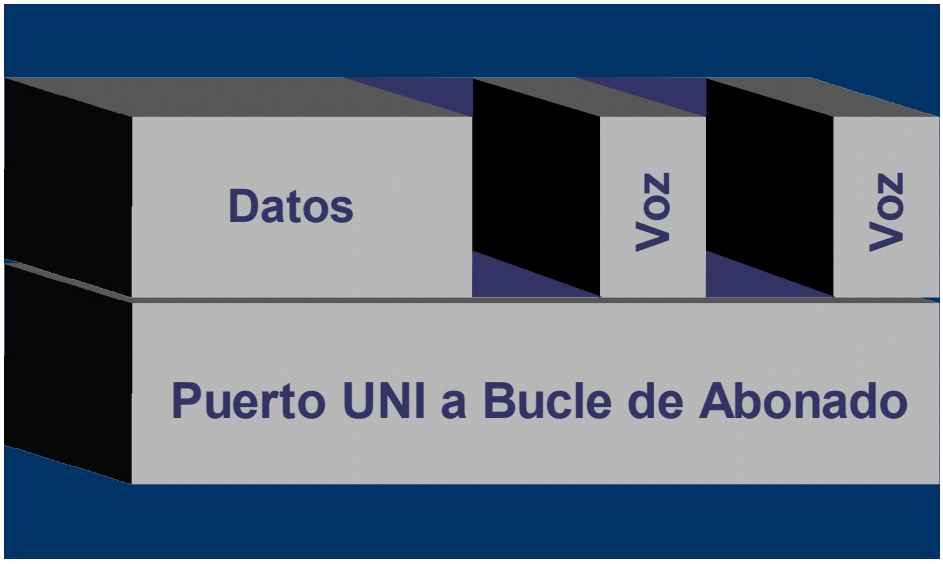



*Figura 8 Estructura del Abonado*

En un nivel inferior estaría el punto de acceso a la red, punto terminal de red, o puerto UNI. Sobre esta capa estarían todas las aplicaciones de comunicación que use el abonado. Como se muestra en la figura 8, pueden ser varias y de diferentes tipos de servicios, más de los que se representan. Pero siempre estarán limitadas por la conexión que el abonado tenga contratada con la red.

El esquema propuesto sugiere que se distingan como partes del abonado dos clases de objetos: el puerto UNI y, dicho de modo abstracto, las aplicaciones.

### 3.5.1 PUERTO UNI

En esta clase de objetos se destaca en primer lugar la semejanza con los puertos de cualquier elemento de la red: son el objeto a través del cual el conjunto recibe y envía células. A ellos está conectado un enlace.

Tan sólo añadir que, dada la simplicidad del esquema de abonado empleado, este tipo de puertos tendría que incluir una gestión de puntos de acceso al servicio – SAP (*Service Access Point*) semejantes a los del punto terminal de red – PTR en la red digital de servicios integrados – RDSI. De hecho, la UIT normaliza la red ATM en vistas a lo que ellos definen como RDSI de banda ancha o RDSI-BA. Esta RDSI-BA es a la que corresponden los objetivos de ATM citados en la introducción y tomados de recomendaciones de este organismo.

### 3.5.2 APLICACIONES

En este proyecto no se hará un modelado profundo de las aplicaciones. Se implementarán sólo dos aplicaciones muy extendidas: Voz y datos – HTTP, que en la realidad suponen más del 95% del tráfico de red. Para un modelo más detallado puede tomarse como referencia la Recomendación I.371.1 de la UIT-T[40].

Por otro lado, se introdujo el tráfico fractal como idóneo para la caracterización del tráfico en ATM. Dado que no se ha pretendido abarcar todas las aplicaciones, y las que se abarcan no se pretenden modelar con gran precisión, se ha optado por desarrollar modelos más sencillos partiendo de descripciones cualitativas menos matemáticas.

[40]UIT-T Rec. I.371.1 (11/2000) – Guaranteed frame rate ATM transfer capability, aptdo. 6 – Guaranteed Frame Rate transfer capability (GFR).

#### 3.5.2.1 Comunicaciones de voz

Las comunicaciones de voz son sin duda las más extendidas y clásicas. De acuerdo con el servicio prestado en RDSI para comunicaciones de voz normales, se establece una generación de tráfico constante CBR con una velocidad de transmisión de 64 Kbits por segundo. Dado que no se incluye en este estudio la capa de adaptación de ATM – AAL se supone que los 48 bytes de datos de la célula son empleados íntegramente para la transmisión de información. Más adelante se hará una salvedad a esta afirmación.

#### 3.5.2.2 Aplicaciones Web: tráfico HTTP

Como ejemplo de aplicación de datos se implementa también la aplicación por excelencia: navegación Web. En cuanto tráfico de datos se presta más a la conmutación de paquetes que de circuitos, aunque este punto será tratado más adelante en el enrutamiento.

Para el tráfico HTTP se establecen los siguientes descriptores:

- ✓ Tiempo promedio entre clicks, entendiendo que esto provoca una generación de tráfico por cambio de página visitada. Dado que el tiempo entre clicks no es ni mucho menos constante, se requiere un parámetro para indicar la variabilidad.

- ✓ Variabilidad del tiempo entre clicks. Este parámetro aleatorio debería no ser uniforme e incluso tampoco debería tener una función de distribución continua, con el fin de poder recoger comportamientos de usuario que pueden provocar pequeñas ráfagas de células.

- ✓ Carga promedia de un click. Con esto se quiere expresar en promedio la carga de tráfico que produce cada click.
- ✓ Variabilidad en la carga de tráfico que produce un click. Este parámetro aleatorio debería no ser uniforme e incluso tampoco debería tener una función de distribución continua, con el fin de poder recoger comportamientos de usuario que pueden provocar pequeñas ráfagas de células.

Una suposición que posible es la siguiente: dado que el usuario está limitado por la capacidad de transmisión de su bucle, una transmisión de datos HTTP ocupará cuando transmita toda la capacidad disponible. En definitiva, se trata el tráfico Web como tráfico ABR.

#### 3.5.2.3 Otras aplicaciones

Como ya se ha dicho, en este proyecto sólo se incluyen los dos tipos de comunicaciones señalados. No obstante, añadir nuevas clases de comunicación supone simplemente añadir nuevas clases de objetos.

Con esto se quiere remarcar la utilidad de la modularidad, planteada como objetivo al basarse el simulador en la programación orientada a objetos.

## 3.6 OTRAS CLASES DE OBJETOS EN UNA RED ATM

Con las clases de objetos descritas se tiene lo esencial en una red ATM. Estaría compuesta de un conjunto de nodos, entre los que es posible distinguir un conjunto de nodos NNI y un conjunto de Centrales. Habría otro conjunto de enlaces, de entre los que se podría distinguir un conjunto de enlaces NNI y otro

de bucles de abonado. Y por último estaría un conjunto de abonados, cada uno conectado a una central.

Pero aún cabe establecer operaciones internas de la red que no son propias de ninguno de los tipos de objetos enumerados. Falta, por tanto, añadir elementos que realicen esas funciones, tratándose a continuación.

### 3.6.1 CENTRO DE CONTROL DE RED

Con este nuevo objeto se pretende agrupar en un elemento de red los procedimientos de gestión de ámbito global, y que por tanto no son propios de ningún nodo concreto. En este proyecto se señalan las básicas:

- ✓ Arranque de los elementos de red. Los nodos y enlaces una vez creados hay que ponerlos en funcionamiento. Esto implica poder acceder a cada uno.
- ✓ Detención de elementos de red.
- ✓ Arranque y detención de las aplicaciones de abonado. Este tercer punto no habría que incluirlo, pero dado que los abonados son simulados, se podría proveer de un modo de controlarlos.

Sin duda se puede pensar en más opciones de gestión global de la red para incluirlos en este centro de control. Nuevamente hay que decir que este elemento es ampliable sin más que añadir funciones.

### 3.6.2 ENRUTADOR

Desde el punto de vista del enrutamiento se distinguen dos casos: conmutación de circuito, en los que la ruta queda prefijada en la admisión de la

conexión; y conmutación de paquetes, o si se prefiere de células, en los que no hay una ruta prefijada.

El cálculo de rutas en la conmutación de circuitos puede plantearse desde una perspectiva local, donde cada nodo usa su información para aportar soluciones parciales. Otra posibilidad es una perspectiva global, donde un elemento de red realiza esta función tras recopilar información de cada nodo y establece la ruta, informando luego a cada nodo que intervenga que concatenación ha de hacer.

Para las aplicaciones HTTP que se han descrito se ha elegido conmutación de células con enrutamiento local. Por el contrario, para las comunicaciones de voz se ha optado por conmutación de circuitos prefijando en la admisión de la conexión la ruta. Ésta será calculada por un elemento de red con esa única función, que recogerá información del estado de cada nodo y sus enlaces y efectuará un enrutamiento puramente global.

Por tanto, los nodos manejan dos modos de enrutamiento para las células. Se empleará uno u otro en función del VPI/VCI. Las células de datos emplean un rango de valores diferente al de las células de conexiones de voz. El siguiente paso, que se ha considerado que excede el ámbito de este proyecto, sería llevar a cabo la separación de buffers.

Más concretamente las células de tráfico HTTP llevarán un VPI y un VCI determinado. Esto está implementado así en cualquier router ADSL, como el que muestra la figura 9. Corresponde a la configuración de un router ADSL de una conexión contratada a Telefónica actualmente en uso.

Este hecho plantea que para poder identificar el destino en las células de datos no hay más remedio que implementar una capa de adaptación entre

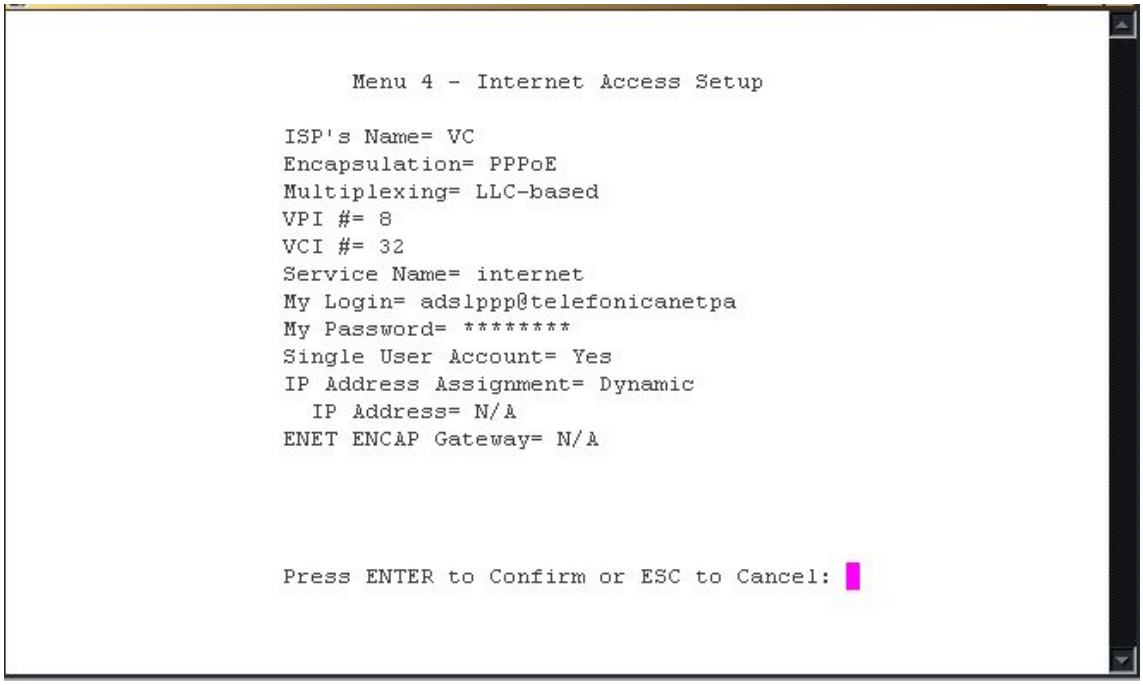



*Figura 9 Configuración de un router ADSL*

ATM y los datos HTTP. Aunque esta solución está perfectamente normalizada, en este proyecto se ha optado por una adaptación más sencilla, donde sólo se provee de direccionamiento.

Volviendo al cálculo de rutas para conmutación de circuitos, se diseña un elemento de red para calcular las rutas con base a la información que recopile del estado de los nodos y sus enlaces. Por simplicidad se implementa un algoritmo que calcula la ruta mínima.

Al enrutador se le encarga que una vez calculada la ruta haga llegar a cada nodo implicado la información que necesite.

### 3.6.3 CONSOLA

Este es un objeto sencillo. Se trata simplemente de una ventana donde monitorizar mensajes que puedan generar cualquiera de los nodos, centrales, o el centro de control o el enrutador.

# 4 DISEÑO DEL MODELO DE RED DESCRITO

## 4.1 LENGUAJE DE PROGRAMACIÓN

El lenguaje de programación elegido es Java, de Sun Microsystems Inc. Concretamente se ha empleado el kit de desarrollo J2SDK versión 1.4.1. Los motivos para esta elección son los siguientes:

- ✓ Gratuidad de la licencia, e incluso de entornos de desarrollo.
- ✓ Versatilidad en vistas a la ejecución del programa. En primer lugar no existe dependencia con ningún sistema operativo. Junto con esto, se puede compilar para su ejecución como aplicación windows y, con algunas modificaciones, para su ejecución desde la una página web.
- ✓ Extensión. Cualquier ordenador personal puede disponer de un *Java RunTime* o máquina virtual Java, y frecuentemente cuenta con él. Por otro lado, la documentación es amplia y extensa, y a veces excelente.
- ✓ Java fue concebido para ser un lenguaje orientado a objetos, aspecto imprescindible para este desarrollo.

## 4.2 PRELIMINARES

En Java, como en cualquier lenguaje de programación orientado a objetos, se utilizan las siguientes nociones básicas, introducidas con el fin de hacer más comprensibles las posteriores explicaciones.

- ✓ Clase. Este concepto se puede asemejar al de tipo. Siempre indica una clase o tipo de objetos. Es la abstracción de objetos distintos pero iguales. Todo conjunto de objetos que tengan algo en común pueden agruparse en una clase, ya que puede hacerse una descripción que sea común a todos ellos.

- ✓ Objeto. Un objeto siempre es una instancia de una clase. Dicho de un modo más coloquial, un ejemplar de una clase o tipo de objetos. Un objeto puede ser manipulado, alterado, variado, solo que no puede dejar de pertenecer a la clase -de objetos- a la que pertenece.

- ✓ Atributos. Son las variables y objetos declarados dentro de la clase.

- ✓ Métodos. Son las operaciones o funciones propias de un tipo de objetos. Se incluyen en la definición de la clase. Una clase, y por tanto, todos los objetos de esa clase, quedan definidos por la lista de objetos-componentes que poseen y por los métodos que contienen, siendo éstos las operaciones que pueden realizar.

- ✓ Constructores. El constructor es un método de la clase que se ejecuta siempre que se crea un objeto de esa clase. Se identifica del restos de los métodos por que tiene el mismo nombre que la clase en la que está incluido. Otra diferencia es que un constructor nunca devuelve un resultado.

  Una función típica de los constructores es inicializar todos sus componentes. Siempre que se crea un objeto, se hace invocando a

uno de sus constructores, ya que, como se verá, puede haber varios y diferentes.

- ✓ Herencia. Cuando una clase hereda características de otra, se habla de clase padre y clase hijo, siendo la segunda la que hereda. En este caso, la clase hijo pasa automáticamente a contener todos los componentes y todos los métodos de la clase padre, salvo los que expresamente se hayan definido no heredables en la clase padre. Estos aspectos son de especial interés, como se verá.

  Cuando se pretenda que una clase herede de otra bastará con indicarlo en la definición, con la palabra reservada apropiada y el nombre de la clase padre. En Java, la palabra en cuestión es "extends".

- ✓ Sobrecarga de métodos. Esto incluye también a los constructores. Un mismo método o función puede definirse tantas veces como se necesite. Cada definición se deberá poder distinguir del resto por el número de argumentos de cada tipo que recibe al ser invocado.

  Un ejemplo que puede ilustrar esto es el siguiente: En un objeto de una supuesta clase "lista_telefonos" se podría definir el método "entero *buscar* (cadena *apellidos*)" para que dados unos apellidos devuelva el número de teléfono. Una sobrecarga útil de esta función podría ser "cadena *buscar* (entero *numtelefono*)", con el fin de poder averiguar a qué persona pertenece un número de teléfono dado.

## 4.3 COMPOSICIÓN DE LAS CLASES DE OBJETOS.

Para cada clase de objetos se muestra en código Java los atributos, constructores y métodos de la propia clase y de las clases internas. Estos también pueden verse en los diagramas de clases del Anexo I. Para mostrar las funciones que desempeñan los métodos y el uso que se hace de los atributos se describe la parte de funcionamiento más básica, y se deja para un apartado posterior otros algoritmos de especial importancia y menos sencillos.

### 4.3.1 LA CÉLULA

Para la programación de la célula se ha intentado hacer una reproducción de la realidad lo más exacta posible. Para ello, se ha hecho en primer lugar la división cabecera-datos, y a continuación se ha especificado cada uno de estos componentes como una secuencia de bits, haciendo uso de la clase BitSet.

Celula

Simulador de Redes ATM

```
public class Celula {
  //Atributos
  protected CabeceraCl cab;
  protected BitSet datos;
  // Constructores
  public Celula()
  public Celula(int vpi, int vci, int pt, boolean clp)
  //Métodos
  private int bitanum(BitSet bits)
  public int readdata(int ini, int fin)
```

```
        public void writedata(int ini, int fin, int dato)
        // Clases internas
        public class CabeceraCl {
          //Atributos
          public BitSet datos;
          private int[][] formato;
          // Constructores
          CabeceraCl(Celula this$0, int vpi, int vci, int pt, boolean
clp)
          //Métodos
          public void set(BitSet newcab)
          public int vpi()
          public void vpi(int newvpi)
          public int vci()
          public void vci(int newvci)
          public int pt()
          public boolean clp()
          public void clp(boolean flag)
          public int hec()
          public boolean esnula()
          public void printcelcab()
        }
      }
```

Celula

El campo de datos se define como un componente BitSet dentro de la clase Célula. Su tamaño será 384 bits, que equivale a 48 bytes.

Para la cabecera se crea una clase aparte, dentro de la clase célula. Esto es lo que en documentación en inglés se denomina *inner class*. En primer

lugar habrá un componente datos de la cabecera que será un BitSet de tamaño 40.

Los métodos incorporados a la clase son los siguientes:

- ✓ Dos constructores. En uno se indican directamente de los valores de la cabecera VPI, VCI, PT, y CLP – HEC no es utilizado en el simulador, aunque el campo no se omite y su uso puede ser implementado.

  El otro constructor no tiene argumentos. Crea la célula con todos los bits de la cabecera a cero.

- ✓ "writedata" escribe un valor dado, comenzando y terminando en los bits que se le indiquen.

- ✓ "readdata" devuelve un entero con el valor convertido decimal contenido entre los dos bits que se le indiquen.

- ✓ "bitanum" es un método privado -sólo puede ser invocado desde dentro de la propia clase- y devuelve el valor decimal de una secuencia de bits. Como puede observarse en el diagrama UML del Anexo I es utilizado por "readdata" y otros métodos de la cabecera.

#### 4.3.1.1 Cabecera

La cabecera contiene una secuencia de bits -datos- y una matriz privada donde se indica el bit donde comienza el primer campo, el segundo, etc., y la longitud.

El constructor es único, y recibe como parámetros los valores de cada campo y un *handler* o referencia a una célula, que siempre será la célula (objeto concreto) a la que pertenece.

Los métodos programados son los necesarios para leer o escribir en cada campo, otro para escribir directamente toda la cabecera -método "set"-, y otros programados para testeo del código: "esnula" y "printcelcab".

### 4.3.2 NODO DE RED

El contenido de la clase nodo es el siguiente:

Nodo

Simulador de Redes ATM

```
public class Nodo {
  //Atributos
  int idnodo;
  protected String idalias;
  protected int numlinks;
  protected int TAMCola;
  private SwitchUnit swu;
  protected PUERTOSCl[] PUERTO;
  protected Conexiones[][][] ConnMatrix;
  protected NodoPanel visor;
  // Constructores
  public Nodo(int idnd, String alias, int tam, int[] idlinks,
int kbps)
  protected Nodo()
  //Métodos
  protected int ocupcola()
  protected void start()
  protected void stop()
```

```
        protected void pause()
        protected void wakeup()
        protected void chfactor(long newfactor) throws
TpoNegativoException { }
        // Clases internas
        public class PUERTOSCl {
          //Atributos
          public int idpto;
          protected int idlink;
          protected int nextnodo;
          protected boolean nxtndnni;
          //Constructores
          PUERTOSCl()
          PUERTOSCl(int i, int il)
          //Métodos
          public void RX(Celula cel)
          protected void TX(Celula cel)
          protected boolean NoEsOEM(Celula cel)
          protected void OEM(Celula cel)
        }
      }
```

## Nodo

Todo los objetos de red, tal y como se irá viendo, están identificados por un número y un alias. El número es la referencia usada en todo el código. El alias es la referencia que siempre se le muestra al usuario. Estos son los atributos “idnodo” e “idalias”, respectivamente.

A parte de esto, un nodo contiene un SwitchUnit denominado “swu”, un array de puertos denominado “PUERTO”, una matriz de tres dimensiones para almacenar información de enrutamiento -”ConnMatrix”- y un panel donde se

hace muestra información de estado -"visor". A esto hay que añadir dos parámetros de construcción: el tamaño de la cola -"TAMCola"- y el número de puertos o de enlaces conectados -"numlinks". El atributo "swu" es privado, por lo que solo el nodo al que pertenece se le permite acceder.

La clase nodo tiene dos constructores. Uno recibe todos los parámetros necesarios para la inicialización y configuración del objeto. El otro crea el objeto y punto, no inicializa ninguno de los atributos. Es empleado por otras clases en determinados momentos de la ejecución.

Los métodos son los siguientes:

- ✓ "ocupcola" responde con un promedio de ocupación de la cola tratado en la unidad de conmutación "swu".
- ✓ "start" y "stop" para arrancar y detener la unidad de conmutación "swu" nodo.
- ✓ "pause" y "wakeup" para pausar y reanudar la actividad del "swu". No están implementadas y no se usan.
- ✓ "chfactor" para cambiar el escalado de tiempo de la simulación. Si durante su ejecución se produce un desbordamiento de los parámetros de escalado, remite la excepción al método que lo invocó.

#### 4.3.2.1 Puertos

"PUERTOSCl" es una clase interna de la clase nodo. Sus atributos son un identificador -"idpto"- que sólo tiene sentido referido dentro del nodo, el identificador del enlace al que está conectado -"idlink"-, el identificador del

nodo de red con el que está conectado -"nextnodo"-, y un atributo que indica si el siguiente nodo es un nodo de red o no, es una central -"nxtndnni".

Los constructores son dos. Al igual que ocurre con la clase nodo, hay uno en el que el puerto queda configurado, y otro que simplemente lo crea.

Los métodos son transmitir y recibir -"TX" y "RX" respectivamente-, otro que resuelve si una célula es de gestión de red -"EsOEM"- y un último método -"OEM"- que trata las células de gestión que se reciben.

### 4.3.3 SWITCHUNIT

La clase "SwitchUnit" corresponde a la unidad de conmutación, tal y como se vio. Dado la complejidad de esta clase, se muestra con todas sus clases internas menos la clase del PCR, que queda a parte.

SwitchUnit

Simulador de Redes ATM

```
class SwitchUnit {
  //Atributos
  private int numlinks;
  private int TAMCola;
  private SBMCl SBM;
  private PCRCl PCR;
  protected MUXCl MuxIn;
  private DEMUXCl DEMUX;
  private Nodo nodo;
  // Constructores
  SwitchUnit(int nlks, int TMCola, int kbps, Nodo nd)
  protected SwitchUnit()
  //Métodos
  protected int ocupcola()
```

```
        protected void start()
        protected void stop()
        protected void pause()
        protected void wakeup()
        protected void chfactor(long newfactor) throws
TpoNegativoException { }
        protected void TRcel(Celula cel, int idpto)
        // Clases internas
        class DEMUXCl {
          // Constructores
          DEMUXCl(SwitchUnit this$0)
          //Métodos
          void RX(Celula dato, int out)
        }
        class MUXCl {
          //Atributos
          protected Object out;
          private int count;
          // Constructores
          MUXCl(SwitchUnit this$0)
          //Métodos
          void RX(Celula cel, int pto)
        }
        class SBMCl {
          //Atributos
          private BitSet[] cache;
          private BitSet libre;
          // Constructores
          SBMCl(SwitchUnit this$0)
          //Métodos
          void RX(BitSet dato, int posbuffer)
```

```
    BitSet get(int pos) { return null;}
  }
}//VER PCR MAS ADELANTE
```

## SwitchUnit

Al igual que el nodo, el SwitchUnit dispone de los parámetros de construcción “numlinks” y “TAMCola”, número de puertos y tamaño de cola, respectivamente.

Como ya se explicó, hay un multiplexor de entrada -”MuxIn”-, una unidad de control de prioridad y enrutamiento -”PCR”-, con un buffer de memoria para datos -”SBM”- y un multiplexor de salida -”DEMUX”.

El “MuxIn” recibe las células de los puertos, pasa la cabecera y la identificación del puerto receptor al “PCR”. Si éste -su cola- acepta la célula, almacena los datos en el “SBM”, en la posición indicada por aquél.

Cuando la cabecera es procesada en el “PCR”, éste recupera los datos del “SBM” envía la nueva célula al “DEMUX”, junto con el identificador del puerto de salida. El “DEMUX” se encarga de transmitir al puerto la célula, y éste al enlace.

Los métodos del SwitchUnit son, por un lado, los vistos en el nodo: “ocupcola”, “start”, “stop”, “pause”, “wakeup” y “chfactor”. Estos métodos son invocados por el nodo, cuando algún objeto de la red se lo pide, como por ejemplo en centro de control de red. Recuérdese que los nodos tienen un objeto SwitchUnit privado -”swu”-, y por tanto sólo el propio nodo puede manejarlo.

A estos métodos hay que añadir “TRcel”. Él es el que recibe células de gestión de red directamente del puerto por el que llegan, sin pasar por la cola. Las remite directamente al “PCR”.

A continuación son descritas las clases internas, dejando la de “PCR” para el final, por ser claramente el de mayor complejidad.

#### 4.3.3.1 Multiplexor de entrada: MUXCl

Sus únicos atributos son de testeo y no se emplean. Uno tipo Object, denominado “out”. El segundo, “count”, se trata de un contador tipo entero.

Carece de constructor, y su único método es “RX” y es invocado por los puertos cuando reciben células que no son de gestión.

#### 4.3.3.2 Demultiplexor de salida: DEMUXCl

Sin atributos ni constructor. Su único método es “RX”, y es el que emplea “PCR” para transmitir células atendidas.

#### 4.3.3.3 Buffer de Memoria Compartida: SBMCl

Sus atributos son dos:

- ✓ Una caché -denominada “cache”- consistente en un array de BitSet, cada uno con capacidad para almacenar el campo de datos de una célula. El tamaño del array es igual al tamaño de cola.
- ✓ Un array de bits -denominado ”libre”- del mismo tamaño que la caché y que se emplean como banderas para indicar las posiciones libres y ocupadas de aquélla.

Tiene un constructor que dimensiona e inicializa los arrays.

Los métodos son:

- ✓ “RX” para recibir nuevas entradas de datos.
- ✓ “get” para devolver los datos de una posición que se le indica.

#### 4.3.3.4 PCR

La clase del PCR, denominada PCRCl, es la más importante del nodo. Esto y su extensión son el motivo por el que ha sido dejada para el final. Su composición es la siguiente:

SwitchUnit.PCRCl

Simulador de Redes ATM

```
class PCRCl extends ATMThread {
  //Atributos
  private Colacirc lde;
  public int[][] TR;
  public int ocupcola;
  int ocupant;
  // Constructores
  PCRCl(SwitchUnit this$0, int kbps)
  //Métodos
  protected void TRcel(Celula cel, int idpto)
  int RX(Celula.CabeceraCl cab, int ptoin)
  public void run()
  void TX(CachePCR dato, int bufferpos)

  // Clases internas
  class Colacirc {
    //Atributos
    private CachePCR[] cola;
```

```
        private int nxtpop;
        private int lstpush;
        private int desbord;
        private int txtidas;
        private int perdidas;
        private boolean busy;
        private boolean desfase;
        // Constructores
        Colacirc(SwitchUnit.PCRCl this$1, int tam)
        //Métodos
        CachePCR pop() { return null;}
        int push(CachePCR dato)
        private int numdatos()
      }

      class CachePCR {
        //Atributos
        protected Celula.CabeceraCl cab;
        protected int idpto;
        // Constructores
        CachePCR(SwitchUnit.PCRCl this$1)
        CachePCR(SwitchUnit.PCRCl this$1, Celula.CabeceraCl c, int
pto)
      }
    }
```

SwitchUnit.PCRCl

Entre los atributos están:

- ✓ “Ide”, de tipo Colacirc, es la cola del nodo. Esta clase se verá en detalle en un subapartado algo más adelante.

- ✓ “ocupcola” de tipo entero. Es un parámetro que almacena un promedio de la ocupación de la cola, calculado en base a muestras de los últimos ciclos de ejecución del nodo. Su rango de variación está truncado por abajo, de modo que nunca indicará una ocupación inferior al 50%. Con esto se consigue que los algoritmos de enrutamiento lleguen a soluciones cuyo número de saltos no esté muy alejado del mínimo.

- ✓ “ocupant”, también de tipo entero, no se usa. Ha sido empleado en depuraciones para implementar un control lineal básico de la cola. Simplemente almacenaba el valor de ocupación de la cola “ocupcola” anterior.

- ✓ “TR”, una matriz de dos dimensiones de enteros. Empleado para almacenamiento de información de rutas usadas en conmutación de células sin establecimiento de conexión. Contiene tantas filas como centrales hay en la red, y cada fila almacena dos valores:

  - El puerto del nodo por el que llegar a la central cuyo identificador es el número de fila. Si su valor es “-1” significa que no se sabe por qué puerto ha de transmitirse, no hay ruta para esa central.

  - Un indicador de coste del camino que llega a la central. Su valor es la suma de los atributos “ocupcola” de todos los nodos por los que se pasará, incluyéndose el nodo. Una excepción es cuando la central está directamente conectada al nodo. En este caso se garantiza el camino directo, y se reduce el tiempo de ejecución de los procedimientos de enrutamiento. Por otro lado, hay que

tener presente una de las simplificaciones realizadas: una central sólo puede conectarse a un nodo de red.

El constructor único sólo recibe como parámetro la capacidad de procesamiento estimada en *kbps*. A partir de este argumento se calcula la velocidad de procesamiento que le corresponde al *Thread* -o hilo- del nodo concreto. Esto se verá en todos los objetos programados como hilos.

Los métodos del PCR son los siguientes:

- ✓ "TRCel" es empleado para recepción y procesamiento de células de gestión. Éstas pueden contener, entre otras cosas, información de las matrices "TR" de otros nodos.
- ✓ "RX" es el método invocado por el multiplexor de entrada para pasar al "PCR" una nueva cabecera. A ésta acompaña la identificación del puerto receptor.
- ✓ "TX" extrae el campo de datos que corresponda del "SBM" y lo escribe en una nueva célula. Para ello recibe la posición donde se encuentran los datos. Escribe la cabecera que recibe como argumento en la cabecera de la célula creada, y se lo pasa al demultiplexor indicando el puerto de salida.
- ✓ "run". Este método está presente en todos las clases que son hilos, aunque en cada caso se define como corresponda. Contiene la ejecución del hilo, y por tanto siempre tendrá una primera parte de operaciones de arranque, y un bucle infinito. Para mayor detalle ver más adelante el apartado de análisis de hilos.

A continuación se describe la composición de la clase interna Colacirc. La otra clase interna que hay en PCRCl, la clase cachePCR, se explica en el mismo epígrafe, al ser usada.

**Cola circular: la clase Colacirc.**

Como se indicó, la clase Colacirc es a la que pertenece el atributo “cola” del PCR. Su funcionamiento es esencial en el funcionamiento del nodo, y por tanto del simulador.

Los atributos todos son privados. Esto implica que sólo el propio objeto Colacirc puede operar con ellos. Estos atributos son:

- ✓ “cola”, de tipo CachePCR, clase interna de SwitchUnit.PCRCl. Similar al caché visto en el “SBM”. Fijándose en la clase CachePCR se observan dos atributos:
  - “cab” de la clase Celula.CabeceraCl. Es el atributo de cada posición de la caché que almacenará la cabecera.
  - “idpto”, de tipo entero, donde se almacena el identificador del puerto por el que fue recibida la célula.
- ✓ “nxtpop” es un entero que indica la siguiente posición de la cola “a sacar”. Su rango de variación es desde 0 hasta “TAMCola”-1, atributo presente en el SwitchUnit. Cada vez que alcance el valor de “TAMCola” tendrá que ser reinicializado.
- ✓ “lstpush” es un entero que indica la última posición empleada para almacenar una cabecera que llegue. Cuando una cabecera llega siempre se guarda en “lstpush”+1. Así se entiende que su rango de

variación sea desde -1 hasta “TAMCola”-2. Cuando alcanza el valor “TAMCola”-1 se le asigna nuevamente -1.

- ✓ “desbord” es un parámetro que cuenta el número de peticiones consecutivas rechazadas por estar la cola llena. Para que varíe no debe haber ninguna posición libre en la cola. Es por eso por lo que siempre vale cero tras una salida de la cola. También es empleado para la activación de medidas de saturación.
- ✓ “txtidas” y “perdidas” son variables enteras empleadas como contadores de células transmitidas y células perdidas. Su uso queda reducido a la depuración.
- ✓ “busy” es un booleano usado como semáforo. Es utilizado para que no pueda haber más de un hilo ejecutando métodos de la cola. Para mayor detalle véase más adelante el análisis de hilos.
- ✓ “desfase” es un booleano con el fin de reducir las operaciones necesarias para el cálculo del número de células en la cola. Indica si “nxtpop” y “lstpush” han sido reinicializados el mismo número de veces. En caso afirmativo “desfase” será falso. Nótese que “lstpush” -entradas- siempre “irá por delante” de “nxtpop” -salidas. Cada vez que “lstpush” se reinicialice -dicho de otro modo, de “la vuelta”-, se pondrá “desfase” a verdadero. Y cada vez que “nxtpop” se reinicialice, “desfase” se le asignará falso.

El constructor único recibe el tamaño de cola como argumento, y se encarga de inicializar todos los atributos.

Los métodos de la clase son 3:

- ✓ “push” y “pop” para meter en la cola y sacar de la cola, respectivamente. Son los únicos dos métodos que pueden ser invocados desde fuera de la clase Colacirc.

- ✓ “numdatos” es privado, solo para uso interno de la clase, y devuelve en formato entero el número de células esperando en la cola. En el análisis de hilos se verá que un aspecto crítico es el tiempo de ejecución de las operaciones de cola. Empleando “desfase”, su código queda reducido a:

```
private int numdatos() {
  if (desfase) {
    return (cola.length+lstpush-nxtpop+1);
  }
  else {
    return (lstpush-nxtpop+1);
  }
}
```

### 4.3.4 CENTRALES

Lo primero que hay que decir es que la clase Central es hija de la clase Nodo como se ve en el código que se muestra. Esto implica que las centrales heredan y poseen todos los atributos y métodos no privados de los nodos. A esto se añadirá los que haga falta.

Central

Simulador de Redes ATM

```
public class Central extends Nodo {
  //Atributos
```

```
        public PTOSUNICl[] PUERTO;
        public int numabons;
        // Constructores
        public Central(int idnd, String alias, int tam, int[] idlinks,
int NMabons)

        // Clases internas
        public class PTOSUNICl extends Nodo.PUERTOSCl {
          //Atributos
          public int idabon;
          private Central.PTOSUNICl.LeakyBucket GFC;
          // Constructores
          PTOSUNICl(Central this$0, int i, int il)
          //Métodos
          public void RX(Celula cel, boolean esuni)
          public void RX(Celula cel)
          private boolean esUNI(BitSet cab)
        }
      }
```

## Central

Gracias a que la central hereda del nodo, el código de la central queda reducido considerablemente.

Los atributos heredados serían todos menos la unidad de conmutación: “idnodo”, “idalias”, “numlinks”, “TAMCola”, “PUERTO”, “ConnMatrix” y “visor”. De estos, la central no inicializa ni utiliza “TAMCola” ni “visor”. Por otro lado, “PUERTO” es redefinido como matriz de instancias de la clase PTOSUNICl, clase interna que a su vez es hija y hereda de la clase PUERTOSCl incluida en la clase Nodo, vista antes.

Un atributo que se añade a los heredados es “numabons”, entero que indica el número de abonados de la central. Aunque este atributo no es estático, y por tanto puede contener valores diferentes en distintos objetos de la clase, en la versión del simulador presente todas las centrales tienen el mismo número de abonados.

El constructor es único, y recibe los argumentos necesarios para la inicialización de los atributos.

La clase Central tan solo cuenta con los métodos heredados de la clase Nodo. Ninguno de ellos es usado, pues como se explicó, las centrales no poseen unidad de conmutación.

#### 4.3.4.1 Puertos UNI: PTOSUNICI

Queda dicho que esta clase definida dentro de la clase central, es una hija de la clase PUERTOSCI, definida a su vez dentro de la clase Nodo.

Así pues, PTOSUNICI posee por herencia los atributos “idpto”, “idlink”, “nextnodo” y “nxtndnni”, ya explicados. De estos sólo se utilizan los dos primeros.

A los heredados se añaden los atributos siguientes:

- ✓ “idabon” es un entero usado para almacenar el identificador del abonado conectado al puerto.
- ✓ “GFC” es una instancia de la clase LeakyBucket. Ésta es una clase interna de PTOSUNICI y tiene una implementación del algoritmo conocido con el mismo nombre. Está normalizado[41] para el control de la velocidad de cresta especificada en cada conexión con el

[41]Cfr. UIT-T I.371 (03/2000), op. cit., pg. 90.

contrato de tráfico. No obstante los nodos no tienen implementado ningún procedimiento de descarte de células, por lo que el control de velocidad de cresta no surte efecto.

Tan sólo añadir que siempre el último puerto del array “PUERTO” es el conectado al nodo que le corresponde. En este caso se desconoce cuál es el nodo, y basta saber el identificador de enlace.

El único constructor de esta clase recibe como argumentos los valores que han de tener “idpto” e “idlink”.

En cuanto a los métodos, PTOSUNICl hereda “TX” y “RX”. Éste último lo redefine y sobrecarga, estableciendo una versión para recepción de células de los abonados y otra para la recepción de células provenientes del nodo de red. A estos métodos añade uno privado denominado “EsUNI” utilizado para depuración.

### 4.3.5 ENLACES NNI

La clase Enlaces se ha compuesto de forma que responde a tres cuestiones: la ausencia de buffer en los puertos, especificación de velocidades concretas, y la posibilidad de simular pérdidas en los enlaces.

Enlaces

Simulador de Redes ATM

```
public class Enlaces extends ATMThread {
  //Atributos
  protected int id;
  protected int idnodoA;
  protected int idnodoB;
```

```
        protected int idptoA;
        protected int idptoB;
        protected buffer[] inbuffer;
        protected int perdidas;
        protected int enviadas;
        protected LinkPanel visor;
        protected int rtindex;
        protected int[] history;
        protected int histidx;
        // Constructores
        public Enlaces(int idA, int idpA, int idB, int idpB, int idE,
int kbitsps) throws TpoNegativoException { }
        //Métodos
        public void RX(int idorigen, Celula cel)
        protected int perdidas()
        protected int enviadas()
        protected boolean TX(int dest, int idport, Celula cel)
        public void run()
        // Clases internas
        public class buffer {
          //Atributos
          protected Celula dato;
          protected boolean libre;
          // Constructores
          buffer(Enlaces this$0)
        }
      }
```

Enlaces

Como puede observarse los atributos del enlace son:

- ✓ “id”, numérico de formato entero que identifica al enlace en la red.

- ✓ “idnodoA”, entero para identificar el nodo o central de uno de los extremos.
- ✓ “idptoA”, entero para identificar el puerto de “idnodoA” al que está conectado.
- ✓ “idnodoB”, nodo del otro extremo. Siempre será un nodo de red, nunca una central.
- ✓ “idptoB” es el identificador del puerto de “idnodoB” al que está conectado.
- ✓ “inbuffer” es el buffer de entrada. Pertenece a la clase interna buffer. Esta clase consta de dos atributos, y carece de constructor y métodos. Los dos atributos son:
  - “dato”, de tipo célula. Es donde se escribe una célula dada por el puerto para su transmisión.
  - “libre” es un booleano utilizado para bloquear el buffer cuando está ocupado.
- ✓ “perdidas” y “enviadas” son simples contadores de células perdidas y células enviadas.
- ✓ “visor” es un panel de indicadores tipo LinkPanel, diseñado para enlaces.
- ✓ “rtindex” es una medida de la carga de del enlace. Podría decirse que almacena la última medición del flujo de células. Es un índice utilizado para el establecimiento de conexiones.

- ✓ “history” es un array numérico. Su uso es como el de un cubo bajo un grifo. Se coloca vacío y se retira tras un tiempo establecido. El nivel de llenado del cubo sirve de medida del flujo del grifo.
- ✓ “histidx” es un entero usado como índice del array “history”.

El constructor es único y recibe todos los argumentos necesarios para la inicialización del objeto.

Los métodos son:

- ✓ “RX”. Es invocado por el puerto correspondiente cuando quiere transmitir por el enlace. Recibe la célula y la escribe en el buffer si está libre.
- ✓ “perdidas” es un método que devuelve el valor de células perdidas.
- ✓ “enviadas” es un método que devuelve el valor de células enviadas.
- ✓ “TX” es un método que sólo invoca el propio enlace y se encarga de pasar la célula al nodo o central destino.
- ✓ “run” es el código que ejecuta el hilo del enlace. Al igual que en el apartado de la clase Nodo se deja su descripción para el apartado dedicado a los hilos.

### 4.3.6 BUCLE DE ABONADO: ENLACEPPP

Esta clase es simplemente una adaptación de la clase anterior y con función de bucle de abonado. Esto es el enlace que conecta un abonado con su central.

EnlacePPP

Simulador de Redes ATM

```
public class EnlacePPP extends Enlaces {
  //Atributos
  public int idabon;
  public boolean ppplibre;
  // Constructores
  EnlacePPP(ImagendeRed.imgabdos imagen) throws
TpoNegativoException { }
  //Métodos
  public void RX(int idorigen, Celula cel)
  protected boolean TX(Celula cel)
  public void run()
}
```

EnlacePPP

Como puede verse, a los atributos heredados añade dos:

- ✓ “idabon” identifica al abonado.
- ✓ “ppplibre” es un booleano utilizado para indicar si el buffer del lado del abonado está libre. La razón de ser de este indicador es que las aplicaciones web están pensadas para colmar la capacidad de transmisión del bucle de abonado.

Hay un solo constructor que recibe todos los datos necesarios para la inicialización de los atributos del objeto.

Los métodos son los mismos, pero se redefine “RX” para que incluya el caso de envío al abonado, y se añade una sobrecarga a “TX” que será la que invoque el usuario.

### 4.3.7 ABONADOS

La clase Abonados está complementada con la clase Aplicaciones. Son respectivamente, el nivel inferior y el nivel superior representados en la figura 8 del apartado 3.5. Así pues, Abonados está pensada para simular la parte de sistema abierto de los usuarios ATM. Dicho de otro modo, simula los usuarios entendidos desde el punto de vista de la red.

La composición de la clase Abonados es la siguiente:

Abonados

Simulador de Redes ATM

```
public class Abonados {
  //Atributos
  public int id;
  public int currentconn;
  public static int maxconn;
  public String alias;
  protected PUERTOCl ptoppp;
  private Aplicaciones[] apl;
  private Aplicaciones apldatos;
  protected AbnPanel visor;
  // Constructores
  public Abonados(ImagendeRed.imgabdos imagen)
  //Métodos
  protected void darelalta()
  protected void snddata(int to)
  protected int newapl(int[] vc)
  protected void start(int ap)
  protected void stop()
  protected void wakeup()
  protected void pause()
```

```
        protected void chfactor(long newfactor) throws
TpoNegativoException { }
        void tipup()
        // Clases internas
        class PUERTOCl {
          //Atributos
          protected int enlppp;
          protected int[][] ConnMatrix;
          // Constructores
          PUERTOCl(Abonados this$0, int enlace)
          //Métodos
          protected void RX(Celula cel)
          protected void TX(Celula cel)
        }
      }
```

Abonados

Los atributos que posee son:

- ✓ “id”. Identifica al abonado en la red.
- ✓ “currentconn” almacena el número de conexiones de voz abiertas en el instante actual.
- ✓ “maxconn” es el número máximo de conexiones de voz que se pueden establecer simultáneamente. Al ser estático es el mismo en todos los objetos de la clase. Si variara en uno de ellos variaría en todos.
- ✓ “alias” es una cadena usada como identificador del abonado en los mensajes y en la interfaz de usuario del simulador.

- ✓ “ptoppp” es una instancia de la clase PUERTOCl. Esta clase es interna a la clase Abonados, No hay que confundirla con la clase Nodo.PUERTOSCl.
- ✓ “apl” es un array de Aplicaciones, clase a parte que se explica tras ésta. “apl” sólo se usará para aplicaciones de voz.
- ✓ “apldatos” es otra instancia de la clase Aplicaciones, que se empleará para la aplicación web del abonado.
- ✓ “visor” es una instancia de la clase AbnPanel.

El constructor de Abonados recibe todos los datos necesarios para la inicialización de los atributos, pero no inicializa las aplicaciones, que se crean en tiempo de ejecución.

Los métodos que presenta esta clase son:

- ✓ “darelalta” es un método invocado desde la red para completar la configuración del abonado. Es utilizado por el constructor de la clase RedATM, aún no vista.
- ✓ “snddata” es un método para creación y arranque de la aplicación web.
- ✓ “newapl” crea una aplicación de voz nueva en el array “apl”.
- ✓ “start” es un método empleado para arrancar la aplicación de voz que se indique como argumento.
- ✓ “stop” es un método para detener todas las aplicaciones que tenga activas el abonado.

- ✓ “pause” y “wakeup” para pausar y reanudar las aplicaciones. No están implementadas y no se usan.
- ✓ “chfactor” sirve para cambiar el escalado de tiempo de los hilos correspondientes a las aplicaciones activas.
- ✓ “tipup” se encarga de actualizar un mensaje emergente del visor, utilizado para indicar el número de conexiones de voz activas.

#### 4.3.7.1 Puerto de Abonado

La clase interna de Abonados denominada PUERTOCl tiene dos atributos:

- ✓ “enlppp” es un entero donde se guarda el identificador del enlace bucle de abonado al que está conectado el puerto.
- ✓ “ConnMatrix” es una matriz de enteros de dos dimensiones donde se almacena información de enrutamiento de conexiones de voz.

El constructor inicializa ambos atributos, para los cual recibe como argumento el identificador de enlace al que está conectado el puerto.

Los métodos son “RX” para recepción de células del bucle, y “TX” para transmisión de células por el bucle.

### 4.3.8 APLICACIONES

Esta clase corresponde a la parte del usuario no visible desde la red. Puede ser considerada como una capa que se apoya sobre la clase Abonados. Esta capa puede contener una aplicación de datos o de voz.

En el simulador, el abonado posee varias instancias de la clase Aplicaciones, tal y como se vio. Y cada instancia de la clase Aplicaciones está limitada a poseer una instancia de la clase correspondiente a aplicaciones de voz o la de datos.

Gráficamente sería como se muestra en la figura 10.

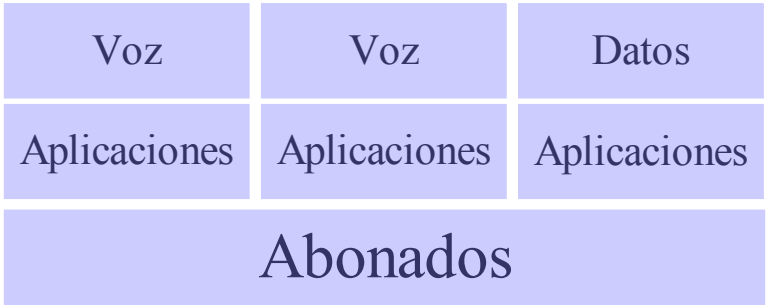



*Figura 10 Estuctura del usuario en el simulador*

Realmente podría suprimirse la clase aplicaciones, y que el abonado poseyera directamente aplicaciones de voz o de datos. Se ha mantenido esta clase por que, aunque no está implementada, sería donde se tendría que buscar la capa AAL de las redes ATM. Además, la capa AAL será diferente con distintos tipos de aplicaciones.

Como se verá, las clases correspondientes a aplicaciones concretas de voz o datos son clases internas de la clase Aplicaciones.

Situado el papel que representa esta clase, se pasa a su composición.

| Aplicaciones |
| --- |
| Simulador de Redes ATM |

```
class Aplicaciones {
  //Atributos
  protected boolean running;
  private Abonados abn;
```

```
    protected static int tammedio;
    protected static int tamvar;
    protected static int ncmedio;
    protected ATMThread aplic;
    // Constructores
    Aplicaciones()
    Aplicaciones(Abonados abon)
    //Métodos
    protected void setaplvoz(int[] vc)
    protected void setapldatos(int abndest)
    // Clases internas: VER MAS ADELANTE
}
```

Aplicaciones

Los atributos que posee son:

- ✓ “running” es un booleano que indica si hay alguna aplicación de voz o datos funcionando.
- ✓ “abn” se usa como puntero al objeto abonado que ha creado la instancia de Aplicaciones.
- ✓ “tammedio”, “tamvar”, y “ncmedio” son enteros empleados en la depuración y no tienen uso en funcionamiento normal.
- ✓ “aplic” es una instancia de la clase ATMThread, aún no vista. Este atributo se inicializa como una aplicación de datos o de voz.

La clase Aplicaciones consta de dos constructores. Uno sin argumentos se utiliza para crear el objeto sin aplicación concreta. El otro se invoca cuando se quiere crear directamente una aplicación concreta.

Los métodos son dos:

- ✓ “setaplvoz” inicializa el atributo “aplic” como aplicación de voz.
- ✓ “setapldatos” inicializa el atributo “aplic” como aplicación de datos.

A continuación se describe la composición de las clases internas, que son las únicas que ejecutan hilos, ya que no lo hacen ni la clase Abonados ni la clase Aplicaciones.

#### 4.3.8.1 Voz: aplvoz

La composición de la clase aplvoz es la siguiente:

Aplicaciones.aplvoz

Simulador de Redes ATM

```
class aplvoz extends ATMThread {
  //Atributos
  double cps;
  double rnd;
  double delay;
  int vpi;
  int vci;
  int pt;
  // Constructores
  aplvoz(Aplicaciones this$0, int pi, int ci, int t)
  //Métodos
  public void run()
  void TX(Celula cel)
 }
}
```

Aplicaciones.aplvoz

Como se ve, hay seis atributos:

- ✓ “cps” indica la velocidad de transmisión en células por segundo.
- ✓ “rnd” y “delay” son atributos cuyo uso se ha limitado a depuración.
- ✓ “vpi”, “vci” y “pt” son los valores vpi, vci y pt que llevarán las células que se transmitan.

El constructor inicializa todos los atributos necesarios.

Los métodos son “TX” para transmitir, “run” que define la ejecución del hilo.

#### 4.3.8.2 Datos: apldatos

Su composición es la siguiente:

Aplicaciones

Simulador de Redes ATM

```
class apldatos extends ATMThread {
  //Atributos
  long tpowebms;
  double cargaclick;
  long rnd;
  long delay;
  int vpi;
  int vci;
  int pt;
  int destino;
  // Constructores
  apldatos(Aplicaciones this$0, int pi, int ci, int t, int
abndest)
  //Métodos
  public void run()
  private void webclick(double carga)
```

```
    void TX(Celula cel)
}
```

Aplicaciones.aplvoz

Los atributos que contiene son:

- ✓ “tpowebms” contiene el valor promedio del tiempo entre clicks, entendiendo que un click es el evento que provoca una nueva carga de transmisión.
- ✓ “cargaclick” es el valor promedio de carga en Kbytes que hay que transmitir a causa de un click. El rango de valores está truncado por arriba, para simular las descargas web. Así pues, se deja un 3% de posibilidad de que el click suponga una descarga.
- ✓ “rnd”, “delay”, “vpi”, “vci” y “pt” tienen el mismo sentido que en aplvoz.
- ✓ “destino” es una atributo que identifica el otro extremo de la comunicación.

  Esto requiere aclarar dos simplificaciones realizadas:

  - Se considera que todo el tráfico de datos es entre dos extremos cualesquiera. Concretamente, dos abonados de la red elegidos en tiempo de ejecución.
  - Todo el tráfico de datos se considera de subida, al contrario de los que realmente ocurre, que principalmente es de bajada. Sin embargo, no supone una restricción seria, ya que el efecto en

los nodos es independiente del sentido del tráfico, y el efecto de la transmisión de datos se consigue.

El constructor recibe todos los datos para la inicialización de los atributos utilizados.

Por último, los métodos de la clase apldatos son:

- ✓ “run”. Como en otras clases con hilo, define la ejecución de éste.
- ✓ “webclick”. Se encarga de realizar la transmisión de todos los datos que requiere un click.
- ✓ “TX”. Invocado por “webclick”, transmite la célula que éste le pasa.

### 4.3.9 CENTRO DE CONTROL DE RED

En esta clase se concentran algunas funciones necesarias en la red la red. Como puede verse, carece de atributos y constructores.

CCRed

Simulador de Redes ATM

```
public class CCRed {
  //Métodos
  public static void start()
  public static void stop() throws SecurityException { }
  public static void pause()
  public static void wakeup()
  public static boolean chfactor()
  public static boolean chfactor(long newfactor) throws
TpoNegativoException { return false;}
  public static void runabn(int abn, int ap)
}
```

CCRed

Los métodos “start”, “stop”, “pause”, “wakeup” y “chfactor” tienen las mismas funciones que sus homónimos en las clases con hilo. La única diferencia es que aquí se emplean para ejecutar el método igual nombrado en cada uno de los objetos con hilo de toda la red. Por ejemplo, “stop” provoca la detención de todos los hilos de la red.

La sobrecarga de "chfactor" que no recibe argumento alguno no se utiliza. Su uso ha quedado limitado a la depuración.

Por último, el método “runabn” se utiliza en los establecimientos de conexiones de voz para indicarle al extremo llamado que comience la aplicación.

### 4.3.10 EL ENRUTADOR: ROUTER

Esta clase es una de las más interesantes, en cuanto que en ella se ensayan los algoritmos de enrutamiento para el establecimiento de comunicaciones orientadas a conexión.

Router

Simulador de Redes ATM

```
public class Router {
  //Atributos
  private int maxsaltos;
  private int dim;
  BitSet[] nnimatrix;
  int[][] nnimatrixFB;
  private boolean minima;
  private int niteraciones;
```

```
        private int abn1;
        private int abn2;
        // Constructores
        public Router(int nummaxsaltos)
        //Métodos
        public int[] conecta(int abon1, int abon2) { return null;}
        private int[] conectand(BitSet[] nni, int nodo0, int nodo1,
int nodo2, int marca, int saltos) { return null;}
        private BitSet[] clonmatrix(BitSet[] matrix) { return null;}
        private int[] setcircuito(int[] circuito) { return null;}
        private int[] setconn(int[] vc) { return null;}
        private int[] setconn(int nodo1, int nodo2, int nodo3, int[]
vc) { return null;}
        private void setconn(int[] vc, int central, int[] nnivc)
      }
```

## Router

Esta clase presenta los siguientes atributos:

- ✓ “maxsaltos”, es un parámetro de construcción que limita la rutas en función de un número de nodos de red máximo.
- ✓ “dim” se usa como variable auxiliar. Su valor a de ser el número de nodos más el número de centrales que hay en la red.
- ✓ “nnimatrix” es una matriz de bits con dimensión 2 que se usan como booleanos para indicar la existencia o no de enlaces.
- ✓ “nnimatrixFB” es una matriz de enteros con dimensión dos que se inicializa con las mediciones de carga que realizadas en el constructor. Así pues, el entero nnimatrixFB[ i ][ j ] valdrá la suma de la medición de carga del enlace entre los nodos “i” y “j” más la

carga medida del nodo “j”. Vendría a significar el coste de, estando en “j”, dar el paso a “i”.

- ✓ “minima”. Booleano que se pasa a verdadero si la mejor ruta hallada es mínima en número de saltos.
- ✓ “niteraciones”. Dado que la solución se realiza mediante ejecución recursiva, “niteraciones” es un contador que indica el número de iteraciones realizadas.
- ✓ “abn1” y “abn2” son los identificadores de los abonados a conectar.

El constructor inicializa las matrices y realiza las mediciones. Estos objetos son de “usar y tirar”. Se crean cuando se quiere calcular una ruta. Tras crear un objeto Router se invoca el método “conecta”, pasando como argumentos los dos abonados a conectar. Este es el único método no privado. Todos los demás son invocados internamente en el objeto Router. Sus funciones son:

- ✓ “conectand”. Es el núcleo de la clase. Su función es hallar la ruta de menor coste entre dos nodos, y funciona recursivamente.
- ✓ “clonmatrix”. Es utilizado por “conectand”. Devuelve una réplica de la matriz de bits que se le pasa como argumento.
- ✓ “setcircuito” es el método que emplea “conecta” una vez que la ruta a sido encontrada. Se encarga de escribir la información de conmutación que cada nodo necesita en sus respectivas ConnMatrix. Para ello decide que identificadores VPI/VCI utiliza en

cada enlace. Lo mismo hace en las centrales y abonados implicados.

- ✓ "setconn" es utilizado por el método anterior. Es el que hace las funciones de dicho método para un par de elementos -nodo, central o abonado- contiguos en la ruta. Está sobrecargado para cubrir los siguientes casos: bucle de abonado llamante, enlace de red, y bucle de abonado llamado.

El funcionamiento se deja para su exposición en otro apartado más adelante.

### 4.3.11 OBJETO RED ATM

El simulador está pensado para que sólo haya un objeto RedATM, ya que todos sus atributos y métodos son estáticos. Si creara varios objetos RedATM, todos serían el mismo, pues los elementos estáticos son comunes a todos los objetos de una clase. Ni siquiera es necesario crear un objeto: se puede invocar sus atributos, constructores y métodos directamente con el nombre de la clase.

La composición de la clase RedATM es la siguiente:

| RedATM |
| --- |
| Simulador de Redes ATM |

```
public class RedATM {
  //Atributos
  public static quest ConsolaIO;
  protected static Nodo[] mgn;
  protected static Central[] mgc;
  protected static Enlaces[] mge;
```

```
    protected static Abonados[] mga;
    protected static EnlacePPP[] mgb;
    protected static int numcentrales;
    protected static int numlinksnni;
    protected static int numnodosnni;
    protected static int numabonados;
    protected static int maxvpi;
    protected static int maxvci;
    protected static VNodos nmonitor;
    protected static VEnlaces emonitor;
    protected static VEnlaces epppmonitor;
    protected static VAbonados amonitor;
    protected static CCRed Ctrl;
    protected static Consola ConsolaMsgs;
    // Constructores
    public RedATM(ImagendeRed imag)
    //Métodos
    public static void dataconn(int abn1, int abn2)
    public static void conecta(int abn1, int abn2)
}
```

RedATM

Como siempre, en primer lugar se explican los atributos. Para mayor orden, en esta ocasión se agrupan por categorías.

- ✓ Matrices generales de objetos.
  - “mgn” es una matriz de objetos tipo nodo. También abarca las centrales, lo cual es posible por que la clase Central es hija de la clase Nodo. Dicho de otra forma, son nodos modificados. Desde “mgn” solo es posible acceder a la parte de las centrales

heredadas de los nodos. Dicho de otra forma, solo pueden ser tratadas como nodos.

- "mgc" es una matriz de objetos de la clase Central, a través de la cual son accesibles todas las centrales en cuanto centrales. Es decir, a diferencia de "mgn", con "mgc" también se puede acceder a lo que las centrales no tienen en común con los nodos.

- "mge" es una matriz de objetos de la clase enlaces. De modo similar a lo que ocurre con la matriz de nodos, también abarca los bucles de abonado, lo cual es posible por que la clase EnlacePPP es hija de la clase Enlaces. Desde "mge" solo es posible acceder a la parte de los bucles de abonado heredadas de los enlaces.

- "mgb" es una matriz de objetos de la clase EnlacePPP. A través de ella hay un acceso completo a todos los bucles de abonado de la red.

- "mga" es una matriz de objetos de la clase Abonados. Por medio de ella se puede acceder a todos los que existen en la red.

  Todas estas matrices tienen en común que el identificador de un objeto dado es la posición que ocupan en la matriz.

✓ Parámetros de construcción.

- "numcentrales" almacena el número de centrales existentes en la red.

- “numlinksnni” almacena el número de enlaces de red y, por tanto, excluyendo a los bucles de abonado.

- “numnodosnni” almacena el número de nodos de red y, por tanto, excluyendo a las centrales.

- “numabonados” almacena el número de abonados que existen en la red.

- “maxvpi” y “maxvci” son los máximos valores utilizables en la red. Esto, que en un principio puede sorprender, tiene el siguiente motivo:

  Cada nodo y central posee una matriz ConnMatrix con información de conmutación para células de conexiones de voz, como queda explicado. Esta matriz nos dice, una célula que llegó por un puerto con unos VPI/VCI, por qué puerto debe salir y con que VPI/VCI. Por tanto es una matriz de dimensión tres, salvo en el caso de los abonados, que sólo tienen un puerto. El número de enteros almacenados en el conjunto de todas las matrices, incluyendo también las de los abonados es:

  *2 x 3 x [maxvpi] x [maxvci] x [número de enlaces de red+numero de bucles de abonados]*

  De esto se deduce que con “maxvpi” y “maxvci” altos -y mucho más si se usan los valores reales- el simulador puede requerir una cantidad de memoria en el ordenador en que se ejecute totalmente desorbitada. Por ello se recomienda ajustar estos topes lo más razonablemente posible.

- ✓ Interfaz gráfica.
  - “nmonitor” es la ventana donde se encuentran todos los paneles de todos los nodos. Las centrales, al no tener hilos, no disponen de panel. En esta ventana también va el menú del simulador.
  - “emonitor” es la ventana donde se encuentran todos los paneles de todos los enlaces de red.
  - “epppmonitor” es la ventana donde se encuentran todos los paneles de todos los bucles de abonado.
  - “amonitor” es la ventana donde se encuentran todos los paneles de todos los abonados que existen en la red. Como se verá, desde esta ventana se establecen las conexiones en tiempo real.
  - “ConsolaIO” es la ventana empleada en el simulador para hacer preguntas al usuario. Desaparece en cuanto el usuario da una respuesta válida.
  - “ConsolaMsgs” es la ventana donde se imprimen todos los mensajes del simulador ante eventos como pueden ser “ruta encontrada”, o “petición de rutas”, etc.
- ✓ A los explicados hay que añadir “Ctrl”. Pertenece a la clase CCRed, y por medio de él se manipulan todos los hilos de objetos de red.

En cuanto al constructor de la clase RedATM, probablemente lleve a la confusión mostrar el código, por lo que sin llegar a ese punto se explica brevemente su funcionamiento.

Recibe como argumento una imagen de red, clase aún no explicada. Siguiendo esta imagen, ha de crear los nodos, centrales enlaces, bucles y abonados que aparezcan en ella.

En primer lugar, crea los nodos. Luego las centrales. Antes de crear cada nodo o central, hace recuento del número de conexiones físicas que tendrá, para crearlo con tantos puertos.

Una vez creadas las matrices “mgn” y “mgc” crea la matriz “mga” -Abonados- y “mgb” -EnlacesPPP, o sea, bucles de abonado.

Para terminar crea “mge”. Primero incluye en ella los objetos de “mgb”. Después busca pares nodo-nodo o central-nodo que, según la imagen, haya que conectar. Encontrado un par, busca sendos puertos libres, y los configura para que queden conectados a un enlace de red que crea para ello.

Por último decir que hay dos métodos en la clase RedATM. Uno se utiliza para crear una comunicación de voz entre dos abonados -”conecta”-, y otra para hace lo propio con comunicaciones de datos -”dataconn”.

### 4.3.12 OTRAS CLASES ÚTILES

En este apartado se agrupan otras clases que merecen ser mencionadas.

#### 4.3.12.1Clase ImagendeRed

ImagendeRed pretende plasmar la idea que el usuario tiene de la red que quiere simular. Puede tomarse de un archivo atm, o de un cuestionario

que el simulador hace al usuario. En este segundo caso siempre se guarda la imagen en un fichero para futuras simulaciones.

Solo cuando se tiene una imagen de red completa se invoca al constructor de RedATM pasándole esta imagen como argumento.

#### 4.3.12.2Archivo ATM: atmfile

La clase atmfile está pensada para guardar y recuperar imágenes de red, y de este modo puedan ser reutilizadas. Al arrancar el simulador se muestran que archivos atm hay disponibles y se pregunta que identificador de red -nombre de archivo- cargar. En caso de responder uno que no exista, el simulador lo advierte y pregunta si se quiere crear una nueva red. En caso afirmativo, tras crear la imagen a partir del cuestionario se almacena con el nombre que se solicitó sin que existiera.

Un ejemplo de un pequeño archivo “atm” es el siguiente:

```
SimATM-OO
Simulador ATM Orientado a Objetos
Archivo Imagen de Red ATM
*****************************************
:DATAINI
1:2:2:2

:0:0:1000
:0:0:1000
:1000:1000:0

sevilla:0:1:10
madrid:1:1:10
```

```
sevilla:0:10:1
madrid:1:10:1
nodo1:2:10:1000

sevilla&0:0:0:0:0:1000
madrid&0:1:1:1:0:1000
```

Esta imagen corresponde a una red con dos centrales y un nodo. Cada central consta de un abonado.

### 4.3.13 INTERFAZ GRÁFICA

La interfaz se estructura en paneles y ventanas – monitores, más una ventana de mensajes.

Los paneles se usan para agrupar los indicadores y en el caso de abonados mandos de un determinado tipo de objetos. Hay tres tipos de paneles: Para nodos, enlaces y bucles, y abonados.

Los paneles de objetos de la misma clase se agrupan en monitores. Hay cuatro monitores:

- ✓ Monitor de Nodos. Agrupa los paneles de todos los nodos de red.
- ✓ Monitor de Enlaces: Agrupa los paneles de todos los enlaces de red.
- ✓ Monitor de Bucles de abonado: Agrupa los paneles de todos los objetos de la clase EnlacePPP.
- ✓ Monitor de Abonados: Agrupa los paneles de todos los Abonados.

Cada monitor es una ventana. A estas cuatro ventanas hay que añadir otra más: “Mensajes del Simulador”. Esta ventana es un atributo del objeto ConsolaMsgs, atributo a su vez de RedATM.

Lo dicho pude verse en la figura 11.

Todos los paneles tienen una etiqueta en la esquina superior izquierda con en alias del objeto al que pertenece y monitoriza. A parte, los indicadores que pueden observarse en cada panel son:

- ✓ Paneles de nodo
  - Nivel de la cola. Muestra numéricamente y con una barra progresiva el número de células esperando en la cola.
  - Células perdidas. Muestra el número total de células perdidas y el porcentaje respecto al número de peticiones.
  - Carga, o “% de uso”. Muestra el porcentaje de ciclos que sí procesan células respecto al total de ciclos.
  - Piloto de ocupado “Busy”. En rojo cuando la cola a sido bloqueada por entrada o salida de una cabecera.
  - Próximo en salir: Indica la próxima posición de la cola que será procesada.
  - Última entrada: Indica la posición de la cola donde se guardó la última cabecera que entró.


*Figura 11 Interfaz gráfica del simulador*

- ✓ Paneles de Enlaces y bucles

  Como puede observarse los indicadores están duplicados, para monitorizar cada sentido por separado. Así, para cada uno se puede observar:

  - Células perdidas. Muestra el número total de células perdidas y el porcentaje respecto al número de peticiones.
  - Carga. Muestra el porcentaje de ciclos que sí transmite células respecto al total de ciclos.
  - Piloto de ocupado “Busy”. En rojo cuando hay una célula en el buffer que está siendo transmitida.

- ✓ Paneles de abonado.

  - Células enviadas. Es el número de células enviadas por el abonado.
  - Células recibidas. Las que el abonado ha recibido en total.

La etiqueta de los abonados puede ser pulsada para establecer nuevas conexiones o transmisiones de datos. El modo de hacerlo es:

- ✓ Elegido el abonado origen o que realiza la llamada pulsar una vez si se desea conexión de voz. Dos si se desea que transmita datos. Una tercera vez para cancelar. En cada pulsación la etiqueta cambia su color de fondo y se hunde. A la tercera vuelve a su aspecto normal.

- ✓ Tras preparar el abonado origen pulsar una vez en la etiqueta de abonado destino. Los mensajes pertinentes aparecerán en la consola.

## 4.4 ALGORITMOS DE ESPECIAL IMPORTANCIA

### 4.4.1 ENRUTAMIENTO PARA CONEXIONES DE VOZ

Como ya se dijo, de esta función se encarga la clase Router.

En primer lugar, el simulador crea una instancia de esta clase, y al ejecutarse el constructor se toman las medidas de carga de todos los nodos y enlaces de red.

Una vez creado el objeto, se invoca su método “conecta” pasándole como argumento los identificadores de los abonados a conectar. Este método localiza la central de cada abonado. Si la central es la misma para los dos abonados establece la conmutación en ella y determina qué par VPI/VCI ha de usar cada abonado. Si son distintas centrales averigua el nodo de red conectado a cada central. Si el nodo es común a ambas, establece la conexión a través del nodo. Si no, invoca al método “conectand” pasándole los siguientes argumentos:

- ✓ Los identificadores de ambos nodos.
- ✓ Las matrices “nnimatrix” y “nnimatrixFB”, que contienen, respectivamente, los saltos de un nodo a otro posibles y el coste de cada salto[42].

[42]Cfr. apartado 3.6.2.

- ✓ Una marca cuyo valor no puede superar el coste de la solución. Esta vez, al ser la primera, será un valor suficientemente elevado como para asegurar que cualquier solución es válida.

- ✓ Un número de saltos máximo. Este argumento no se volverá a nombrar, dado que la implementación no lo usa como restricción. Se entiende que la ruta más corta puede no ser la mejor, y que el incrementar demasiado el numero de saltos incrementará bastante el coste. Tal y como se calcula este coste, su mínimo implica, con cierta precisión, retraso mínimo.

El método “conecta” espera a que “conectand” le devuelva una solución. Cuando esto ocurre, entra en un bucle donde se pide a “conectand” que encuentre de nuevo una ruta pero con un coste menor que la que devolvió. El bucle finaliza cuando “conectand” devuelve el equivalente a “ruta imposible”, lo cual quiere decir que la mejor es la anterior. Hallada la solución, se establece la conexión y se devuelve el par VPI/VCI que ha de usar cada abonado.

Si ndA y ndB son los nodos a conectar, lo primero que hace “conectand” es una lista con los saltos posibles desde ndA. Si uno de ellos es ndB y el coste del salto es menor que el permitido devuelve esa ruta con el coste que tiene.

Si ndB no es alcanzable directamente, se dan los siguientes pasos:

- ✓ Se elige el salto que tenga el coste más bajo. Sea ese salto al nodo nd1.

- ✓ Se incrementa el coste -que parte de cero- con el coste del salto.

- ✓ Se hace una copia de la matriz de saltos permitidos -"nnimatrix", recibida como argumento- y se deshabilitan en la copia todos los saltos de cualquier nodo a ndA.

- ✓ "conectand" se invoca a sí misma para hallar una ruta de nd1 a ndB dada la matriz "nnimatrixFB" con los coste, la copia modificada de "nnimatrix" con los saltos posibles, y estableciendo como coste máximo la marca que se recibió como argumento menos el coste del salto entre ndA y nd1.

Llegados a este punto pueden suceder dos cosas:

- ✓ Se devuelve una ruta entre nd1 y ndB con un coste válido. En este caso se compone la ruta entre ndA y ndB con el coste total y se devuelve como solución.

- ✓ Se devuelve el equivalente a "ruta imposible". En este caso se realizan los siguientes pasos:

  - Se modifica la matriz de saltos permitidos que se recibió como argumento para deshabilitar los saltos de cualquier nodo al nodo nd1.

  - Se comprueba que tras esto el nodo ndA no queda aislado. Si quedara aislado se termina la ejecución devolviendo "ruta imposible".

  - Si no queda aislado, "conectand" devuelve el resultado de invocarse a sí mismo para conectar ndA con ndB con el coste máximo que se que se recibió en un principio, con la matriz de

costes “nnimatrixFB” y la “nnimatrix” que se recibió como argumento pero modificada, para no permitir salto a nd1.

### 4.4.2 ENRUTAMIENTO DE DATOS

Las rutas para conexiones de voz se establecen antes de que comiencen a transmitir, y permanecen fijadas hasta que acaba la conexión.

Por el contrario, el enrutamiento de datos no se realiza antes de la admisión de la conexión, si no en el transcurso de ésta.

Lo primero es recordar que todas las células de datos que viajan por la red lo hacen con un VPI/VCI preestablecido. Los cuatro primeros bytes del campo de datos se usan para indicar la central destino y el abonado destino. Cuando el abonado origen envía células de datos, sólo especifica el abonado destino. Será la central, quien averigüe y escriba en la célula cuál es la central destino.

En los nodos hay dos matrices con información de rutas. “ConnMatrix” tiene la información de conmutación de células de voz, es decir, de células cuyos VPI/VCI no son los empleados para datos. La otra matriz es “TR”, y tiene la información de enrutamiento para de células de datos, que no disponen de circuito establecido.

Al crearse un nodo, automáticamente guarda en TR las rutas a las centrales que tiene conectadas directamente, asignándoles coste 1. El coste cuando no se conoce ninguna ruta hacia una central, es un número suficientemente elevado. De esta forma, cualquier ruta posible tendrá un coste menor. Un nodo determina que no tiene ruta a una central cuando en “TR” no hay especificado ningún puerto de salida para esa central. Y en ese caso,

cualquier ruta de la que le informen tendrá un coste menor, por lo que la aceptará como nueva ruta.

Cuando el “PCR” de un nodo detecta una célula de datos lee en ella cuál es la central destino. En su matriz “TR” encontrará por qué puerto transmitir las células de datos que vayan a esa central, y qué coste tiene el camino hasta esa central. Pudiera ocurrir que no tuviera esa información por no haberla necesitado antes. En ese caso envía la célula por un puerto aleatorio que esté conectado a otro nodo e inicia un proceso de petición de rutas para esa central. Este proceso consiste en pedir a cada nodo de red conectado directamente con él cuál es su coste hasta la central en cuestión.

Cuando un nodo recibe de otro nodo vecino el coste de ruta que éste posee hasta una central cualquiera, comprueba si es mejor que el almacenado en su propia matriz “TR”. Si es así, escribe en el campo “TR” para esa central el coste recibido y el puerto por el que llegó. Acto seguido, envía por todos los puertos que dan a nodos, menos por el que llego, la información actualizada añadiendo al coste la ocupación media de su cola.

Cuando no se tiene una ruta, el coste registrado tiene un valor suficientemente alto. En ese caso, el modo de preguntar por una ruta es simplemente anunciar el coste que se tiene registrado. El vecino que sí tenga registrada una ruta comprobará que su coste es menor, y responderá anunciándolo.

Este procedimiento es similar al que se encuentra en el protocolo RIP en redes TCP/IP. Si por temas se seguridad no se especifica lo contrario, los enrutadores se comunican entre ellos anunciándose rutas con su cantidad de saltos a modo de distancia o coste.

#### 4.4.2.1 Protección ante saturación de cola

Si la cola rechaza un número concreto de peticiones consecutivas, se inicia un proceso de emergencia por saturación. Se supone que esta situación es drástica.

En primer lugar se vacía la cola, desechando todas las células que haya en ellas.

Tras esto se escribe en TR los costes iniciales con los que se creó el nodo: coste 1 para las centrales directamente conectadas, y un número suficientemente elevado para el resto. No se borran las rutas. Después se envía una señal de emergencia a todos los nodos de red directamente conectados. Cuando un nodo recibe esa señal por un puerto las rutas de “TR” que vayan por ese puerto.

### 4.4.3 FUNCIONAMIENTO DE LA COLA DE LOS NODOS

A continuación se hace una introspección en el funcionamiento de la cola, por ser un aspecto determinante en el funcionamiento de todo el simulador.

Como ya se dijo la cola presenta de cara al exterior dos métodos: “push” para nueva entrada de datos, y “pop” salida. Estos dos métodos son los que se describen a continuación. Antes de empezar se recuerda los atributos de los que depende el funcionamiento, que se vieron en la descripción de la clase y son: “desbord”, “lstpush”, “nxtpop”, y por supuesto, “cache”.

#### 4.4.3.1 Admisión: push

La admisión conlleva los siguientes pasos:

- ✓ Se comprueba que “desbord” es cero. Si no lo es:
  - La célula no ingresará en la cola. Se incrementa en uno “desbord”.
  - Si “desbord == 3” se inicia el procedimiento de protección ante saturación de cola, descrito más arriba.
  - Se termina, devolviendo el equivalente a “célula rechazado”
- ✓ Si “desbord” es cero se comprueba que “lstpush” no apunta al final de la cola. Si es así se hace “lstpush = -1”, pues “-1” significa que la posición anterior a la primera es la cero.
- ✓ Se ingresa la cabecera recibida en la posición siguiente a la última, a la vez que se incrementa “lstpush” una unidad. Con la cabecera se guarda el identificador de puerto por el que se recibió.
- ✓ Se comprueba el número de datos en la célula. Si es igual al tamaño de cola se establece “desbord = 1”.
- ✓ Se finaliza devolviendo el valor “lstpush” para que los datos correspondientes a la cabecera recibida se guarden en esa posición del “SBM”.

#### 4.4.3.2 Salida: pop

Cuando “pop” es invocado siempre devuelve una entrada de la cola. Si la cola estaba vacía devolverá una entrada de cola vacía, pero sin variar el índice de salida “nxtpop”. Los pasos que se dan son los siguientes:

- ✓ Se comprueba si “nxtpop” es igual al tamaño de cola. Dado que la primera posición es cero, la última es “[tamaño de cola] – 1”. Si “nxtpop” es igual al tamaño de cola, se le asigna el valor cero para que apunte al principio de la cola.
- ✓ Se comprueba si la cola tiene al menos un 7% de posiciones libres. Si es así se establece “desbord” a cero. El porcentaje es ajustable en el código.
- ✓ Se lee la posición “nxtpop” de la cola. Si no está en blanco se borra esa posición y se incrementa en 1 “nxtpop”.
- ✓ Se devuelve la lectura que se hizo de la cola.

#### 4.4.3.3 Bloqueo de la cola

Como ya se explicó en la composición de la clase Nodo, es fundamental que las operaciones de cola sean atómicas, que no empiece una si que haya acabado la anterior. Esto se consigue con el semáforo booleano “busy”. Lo primero que hacen los métodos de entrada y salida es esperar a que “busy” sea falso. En cuanto esta comprobación se verifica lo siguiente es establecer el valor de “busy” a su negado. Acto seguido se comprueba que “busy” es verdadero.

Junto con esto, lo último que hacen ambos métodos es establecer “busy” a falso, justo antes de finalizar.

Aunque pueda parecer exhaustivo, el método descrito no basta. De hecho, se han hecho múltiples comprobaciones y se han encontrado casos en los que algún hilo de ejecución conseguía entrar a la vez que otro. Esto queda

reducido a un error despreciable con la implementación final, mostrada a continuación.

Las primeras líneas del método de salida “pop” son:

```
while (busy) {
  try {
    Thread.sleep(tpoms/20,(int)(1000*Math.random()));
  }
  catch (InterruptedException ex) {
  }
}
if (!busy) busy=true;
else return new CachePCR();
```

La tercera línea que se ejecuta dentro del bucle provoca una detención en la ejecución. El tiempo que dura esa pausa viene dado por la suma de dos componentes:

- ✓ “tpoms/20”. Es la vigésima fracción de los milisegundos que hay en un ciclo de nodo. Consigue que el hilo que espera a que la cola se desbloquee no dispare la actividad del microprocesador al 100%. Se puede comprobar midiendo la carga del microprocesador que al ejecutarse un bucle del tipo “mientras (verdadero) { }” aquella sube al 100%. Esto supondría que la simulación carecería de valor.
- ✓ “1000*Math.random()”. Es un tiempo aleatorio entre cero y mil nanosegundos. Consigue que probabilísticamente no todos los hilos que esperan el desbloqueo hagan la próxima comprobación a la vez.

En el método “push” la implementación es la misma.

Este procedimiento es una adaptación del algoritmo de escucha implementado en el protocolo de contienda para acceso al medio CSMA, presente en cualquier red informática Ethernet. Este algoritmo tiene entre sus objetivos que, cuando el medio físico queda libre para transmitir, no se colisionen las transmisiones de los terminales que estaban esperando para enviar datos.

## 4.5 ANÁLISIS DE HILOS

Para completar el análisis se detalla la ejecución de los hilos correspondientes a las diferentes clases de objetos programadas. Es decir: aplicaciones (de voz o datos), bucles de abonado, enlaces de red, y nodos.

### 4.5.1 ANÁLISIS DE CARGA DEL SIMULADOR EN EL MICROPROCESADOR.

Es necesario escalar el tiempo. Es imposible que la simulación corra en tiempo real, ya que se simulan multitud de elementos con velocidades de procesamiento quizá mayores que la del propio equipo en que se ejecuta el simulador.

A esto hay que añadir algo que ya se adelantó: Una simulación carece absolutamente de valor si en la actividad del microprocesador con el que se ejecuta se aprecian actividades del 100%. Esto es así por que el simulador trabaja en tiempo real, pero escalado.

Supóngase que se elige como factor de escala 200. El simulador estaría obligado a completar en 200 segundos la simulación de un segundo real, con todas las operaciones que en la realidad hacen todos los componentes de la

red a lo largo de un segundo. Si se viera que la CPU alcanza el 100% de actividad, no es posible garantizar que en 200 segundos de ejecución haya simulado todas las operaciones de ese segundo real.

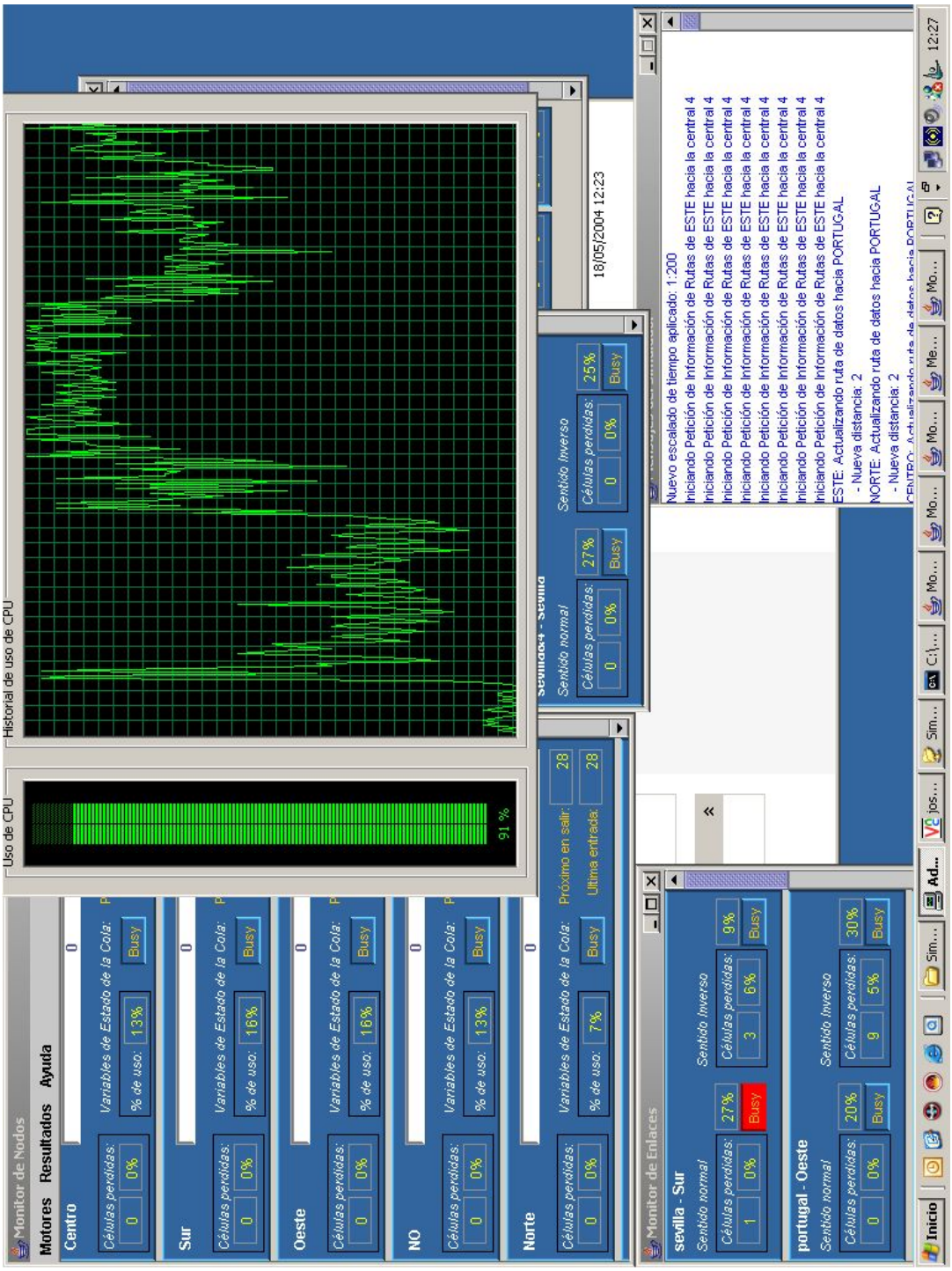



*Figura 12 Actividad de la CPU en una simulación*

En la figura 12 se puede observar una elección de factor de escala un poco bajo, que provoca saturaciones en la actividad del microprocesador. Aumentándolo un poco se conseguiría asegurar la calidad de la simulación.

### 4.5.2 LA CLASE ATMTHREAD

ATMThread es la clase padre de todas las clases con hilos. A su vez, esta clase es hija de la clase Thread, disponible en el kit de desarrollo de Java. Dado que en sí no es un tipo de elemento de red, se ha dejado su descripción hasta llegar al análisis de hilos.

La composición de ATMThread es la siguiente:

ATMThread

Simulador de Redes ATM

```
public abstract class ATMThread extends Thread {
  //Atributos
  protected long tpoespera;
  protected long tpoms;
  protected int tpons;
  public static long factor;
  public long kbps;
  // Constructores
  ATMThread() throws TpoNegativoException { }
  ATMThread(int klbps) throws TpoNegativoException { }
  //Métodos
  public void chfactor(long newfactor) throws
TpoNegativoException { }
  public void pause()
  public void wakeup()
}
```

## ATMThread

Los atributos de la clase son los siguientes:

- ✓ “tpoespera” es el tiempo de ciclo. Cada vez que el hilo realiza sus operaciones espera ese tiempo hasta realizarlas de nuevo. Se estima en función de la velocidad de trabajo asignada al objeto.
- ✓ “tpoms” es la cantidad de milisegundos enteros de “tpoespera”.
- ✓ “tpons” es la cantidad de nanosegundos de “tpoespera” no contabilizados en “tpoms”. Esta división de “tpoespera” se hace porque la función que pausa la ejecución del hilo recibe como argumentos una cantidad de milisegundos y otra de nanosegundos.
- ✓ “kbps” es la velocidad de trabajo asignada al objeto en Kbps.
- ✓ “factor” es el valor por el que se multiplica el periodo real del objeto, hallado a partir del “kbps”. El resultado se le asigna a “tpoespera”.

  El atributo “factor” es estático. Esto implica que todos los atributos “factor” de todos los objetos de clases con hilo son en realidad un mismo y único atributo “factor”. Las clases con hilo, como se dijo, son hijas de ATMThread, y por tanto heredan los atributos y métodos no privados de ATMThread.

Los constructores son dos por necesidad del código. El que recibe una velocidad de procesamiento como argumento es llamado siempre por los constructores de las clases hijas, e inicializa todos los atributos.

De los métodos, solo está implementado y en uso “chfactor”. Este método es utilizado para cambiar “factor”, y aplicar el cambio a los demás

atributos, de modo que surta efecto en la ejecución. Si al calcular los atributos de tiempo se detectara un desbordamiento, el hilo volvería a su estado previo y "chfactor" provocaría una excepción programada para el caso, de la clase TpoNegativoException. Entonces, el método que realiza los cambios de factor en cada hilo capturará la excepción, restaurará todos los hilos con el factor que se tenía, y comunicará lo ocurrido con un mensaje en la consola de mensajes de red.

### 4.5.3 ALCANCE DE LA EJECUCIÓN DE LOS HILOS

El caso ideal es que cumpliendo todas las funciones estrictamente propias del objeto simulado, el hilo realizara el menor número posible de operaciones. Un problema que surge entonces es que el número de hilos necesarios para ello podría ser extremadamente alto, y en consecuencia se fraccionaría en exceso el tiempo de actividad del microprocesador.

Para entenderlo mejor, téngase en cuenta la siguiente diferencia: En la realidad, cada objeto de la red funciona en tiempo continuo. En el simulador, cada objeto de la red funciona durante fracciones de tiempo en las que dispone del microprocesador, de forma que cuando un objeto está ejecutándose, los demás están parados.

En la implementación presentada se ha reducido la asignación de hilos a las clases aplicaciones de voz y datos, enlaces y bucles, y nodos. En algún caso puede parecer que un hilo realiza muchas más operaciones de las que le son propias debido a que muchas clases no tienen hilo. Esto conlleva una gravedad relativa, tal y como se explica a continuación.

Ciertamente, en el tiempo de espera de los hilos, no se descuenta el tiempo empleado en las operaciones del ciclo. Esto implica, que el periodo realmente es “tpoespera” más lo que se tarde en ejecutar las operaciones del ciclo. Si la actividad del microprocesador es baja, el efecto es despreciable, pues implica que “tpoespera” es considerablemente mayor que los que se tarda en ejecutar las operaciones del ciclo. Sería suponer que el microprocesador tiene una velocidad infinita.

A continuación se muestran las operaciones habituales que se realizan en cada ciclo. Estas operaciones se encuentran dentro del bucle infinito de los métodos “run”.

#### 4.5.3.1 Clases Aplicaciones.aplvoz y Aplicaciones.apldatos

Salvando algunas diferencias entre ambas clases, las operaciones son cualitativamente:

- ✓ Se crea una célula con la cabecera pertinente y los datos.
- ✓ Se invoca al propio método “TX”
- ✓ Se actualizan los parámetros de contadores y variables aleatorias.

Este análisis estaría incompleto si no se incluyera el alcance de “TX”. Se entiende por alcance el conjunto de invocaciones a otros métodos de otros objetos que la ejecución de “TX” provoca. Son los siguientes (se especifica en forma de escalera indicando la clase a la que pertenece el método):

- `Abonados.PUERTOCl.TX`
  - `AbnPanel.otraTx`
  - `EnlacePPP.RX`
    - `Enlaces.RX.`
      `Este método comprueba si el buffer está libre y guarda la célula en él.`

- LinkPanel.unomas
- LinkPanel.clost
  (Si el buffer no estaba libre)

#### 4.5.3.2 Enlaces

Las operaciones en un ciclo son:

- ✓ Actualiza el contador de ciclos de su panel.
- ✓ Comprueba si el buffer asignado a un sentido de la comunicación está ocupado. Si es así le pasa la célula al método “TX” y actualiza el contador de transmisiones del panel.
- ✓ Repite lo mismo para el buffer asignado al otro sentido de transmisión.

Por tanto, el método “TX” puede ser invocado una vez, dos, o ninguna. El alcance de este método puede variar según los casos. En el más común sería:

- Nodo.PUERTOCl.RX
  - SwitchUnit.MUXCl.RX
    - SwitchUnit.PCRCl.RX
      - SwitchUnit.PCRCl.Colacirc.push
    - SwitchUnit.SBM.RX
      (Si la célula no la rechaza la cola.)

#### 4.5.3.3 Bucles de Abonado: EnlacePPP

Las operaciones son los mismos 3 pasos que había en la clase Enlaces. El alcance de “TX” es donde está la diferencia.

En primer lugar, hay que distinguir según el sentido de la transmisión. Si el destinatario es el Abonado, sólo invoca el método “RX” del puerto de éste.

Mayor es el alcance cuando la célula va destinada a la central. En este caso sería:

- `Central.PTOSUNICl.RX.`
  En primer lugar hace la conversión de formato de cabecera. Luego comprueba si es de datos. En caso afirmativo, averigua y registra en el campo datos la central del abonado destino.
  - `Central.PTOSUNICl.TX`
    - `Enlaces.RX.`
      Este método quedó descrito en el apartado anterior.

#### 4.5.3.4 Nodos: SwitchUnit.PCRCl

Como se indicó, en los nodos es el objeto “PCR” contenido en SwitchUnit el que posee hilo de ejecución. Su ciclo consta de las siguientes operaciones:

- ✓ Actualiza el contador de ciclos del panel.
- ✓ Lee una entrada de la cola: Colacirc.pop.
- ✓ Comprueba si la lectura no corresponde a una entrada de cola vacía. Si es así la transmite invocando “TX”.
- ✓ Actualiza las mediciones de carga del nodo.

En esta clase, el alcance de “TX” es el que más variaciones puede tener. Se muestran el caso en que la célula a transmitir es de voz. El alcance del método “TX” de la clase PCRCl es:

- `Celula.Celula`
- `SMBCl.get`
- `Celula.CabeceraCl.vpi`
  - `BitSet.set`
- `Celula.CabeceraCl.vci`
  - `BitSet.set`
- `DEMUXCl.RX`
  - `Nodo.PUERTOSCl.TX`

- Enlaces.RX

Cuando la célula es de datos hay casos: Que exista registrada en “TR” la ruta para la central destino, o que no exista. El caso más habitual es que sí exista. Existiendo ruta, prácticamente no hay diferencias con el alcance dado para células de voz.

Cuando la célula es de datos y no se conoce la ruta antes de pasarle la célula al demultiplexor se ejecuta el siguiente algoritmo para petición de rutas:

```
for (int h = 0; h < nodo.PUERTO.length; h++) {
  if (nodo.PUERTO[h].nxtndnni) {
    while (lde.numdatos()>TAMCola-2){
      try {
        Thread.sleep(tpoms/38,(int)(1000*Math.random()));
      }
      catch (InterruptedException ex2) {
      }
    }
    Celula celtr = new Celula(0, 0, 0, false);
    celtr.writedata(0, 16, ctrdest);
    celtr.writedata(16, 32, TR[ctrdest][1]);
    MuxIn.RX(celtr,h);
  }
```

Dado que el tráfico de gestión de rutas es prioritario, el algoritmo funciona a una velocidad 38 veces superior que la cola. Le basta que la cola no esté llena (al menos dos posiciones libres) para transmitir a cada nodo de red vecino su distancia o coste hacia la central, que al no existir será un valor suficientemente alto, como ya se explicó. La respuesta llegará de cualquier nodo que comprueba que su ruta tiene un coste menor.

# 5 RESULTADOS OBTENIDOS CON EL SIMULADOR

En este apartado se muestran los resultados de la simulación de una red ATM compuesta de 6 nodos, 6 centrales y 30 abonados. La estructura de la red es la mostrada en la figura 13.

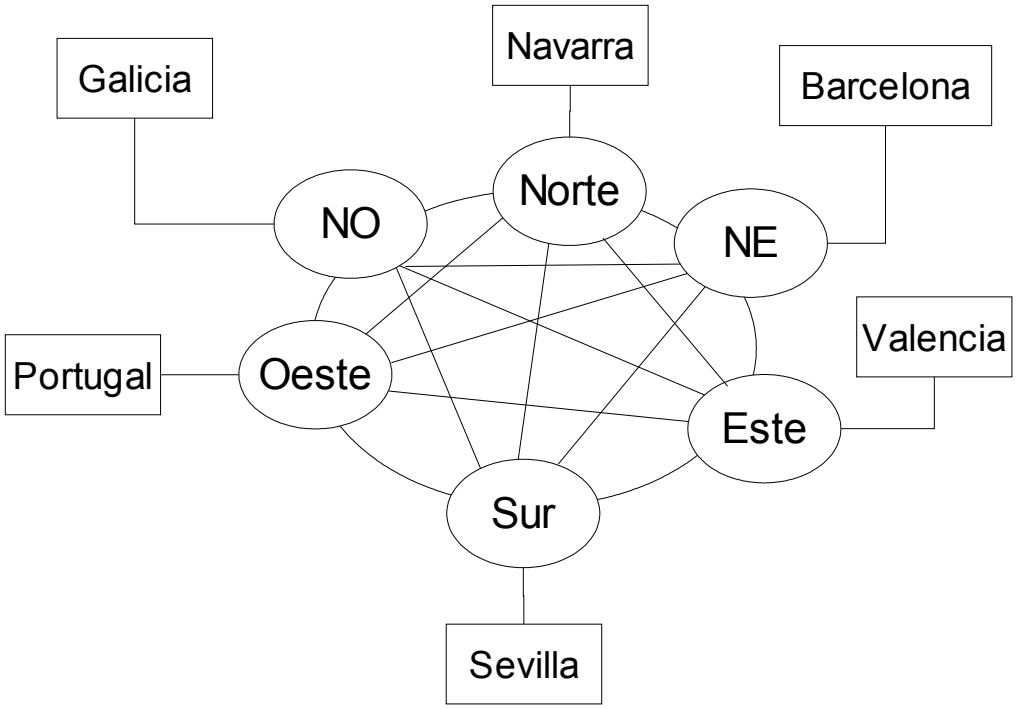



*Figura 13 Red Simulada*

A los enlaces de red representados se le asigna una velocidad de 1000 Kbps. Ciertamente está lejos del caso real, pero dado que el número de abonados ha de ser bajo, se a intentando dimensionar las velocidades en la red de acuerdo con esta limitación. A los nodos se les asigna la misma velocidad que a los enlaces de red. Por último, cada central cuenta con 5 abonados, cada uno con una acceso de 256 kbps y la posibilidad de mantener hasta 4 comunicaciones de voz simultáneas junto con una de datos.

En primer lugar se simula sólo tráfico de voz. En una segunda simulación se empieza generando sólo tráfico de datos, añadiendo al final comunicaciones de voz con el fin de mezclar ambos tipos de tráfico.

Para poder extraer resultados se han incorporado funciones con las que registrar recepciones de células por parte de los abonados. En éstas, se ha empleado el campo de datos para indicar el momento en que la célula es generada, y así poder medir el retraso cuando llega a su destino. Junto con las gráficas de estas mediciones, se incluyen otras que muestran la actividad de los nodos y de los enlaces de red.

## 5.1 TRÁFICO DE VOZ

Se han simulado 286 milisegundos y se han registrado 1925 células que alcanzaron su destino. Los abonados han generado el tráfico correspondiente a las 4 comunicaciones simultáneas que se les permitía. La tasa de bits para cada conexión de voz es de 64 kbps.

La distribución de carga en la red hecha por el enrutamiento puede verse en la gráfica de la figura 14. En ella se muestran en primer lugar el número de ciclos que han ejecutado los nodos, el número de células recibidas y las pérdidas por rechazo de cola, que son nulas.

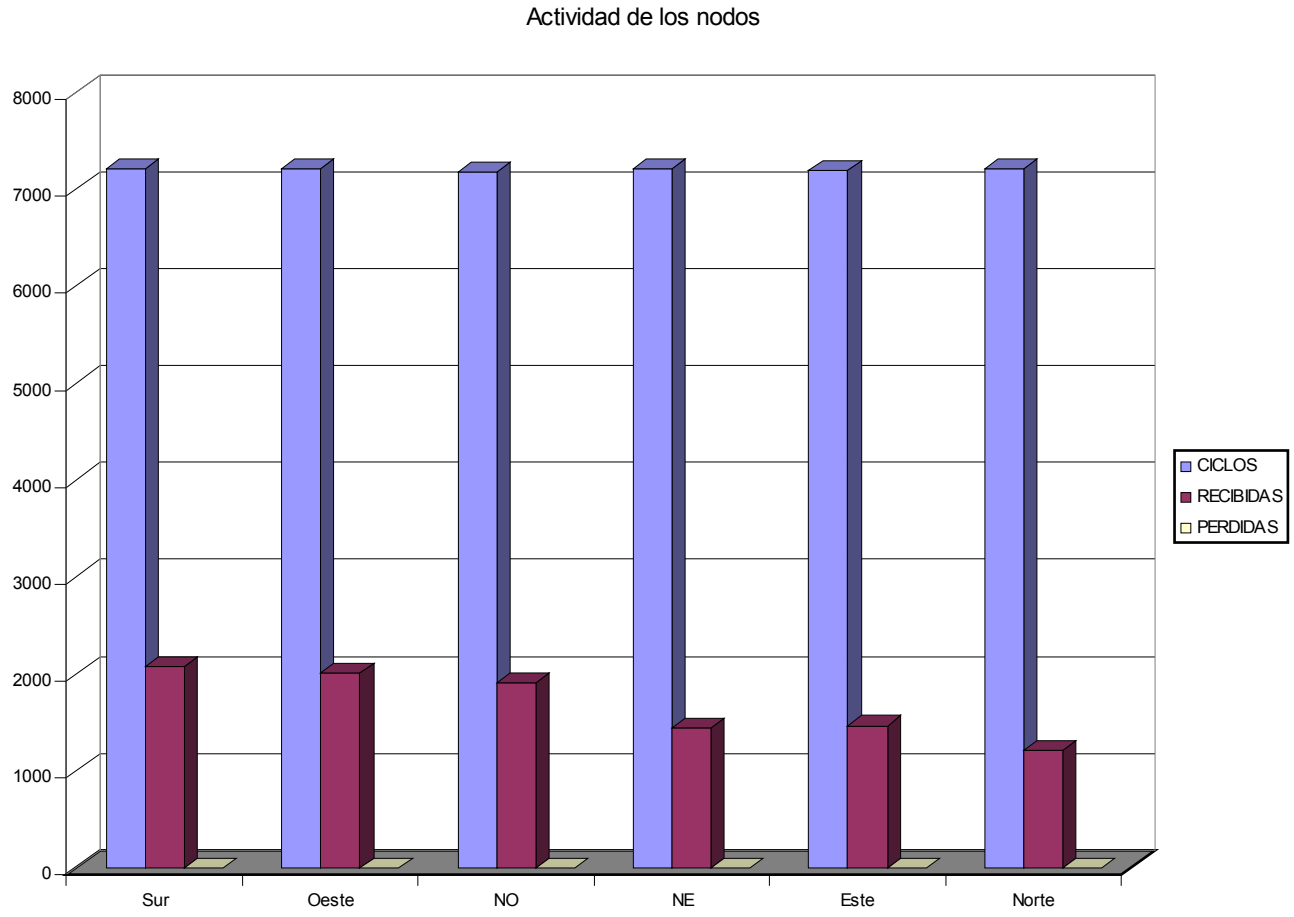



*Figura 14 Actividad de los nodos por el tráfico de voz*

En los enlaces de red se encuentra la mayor pérdida de células, especialmente en los enlaces entre central y nodo. Esto puede verse en la figura 15.

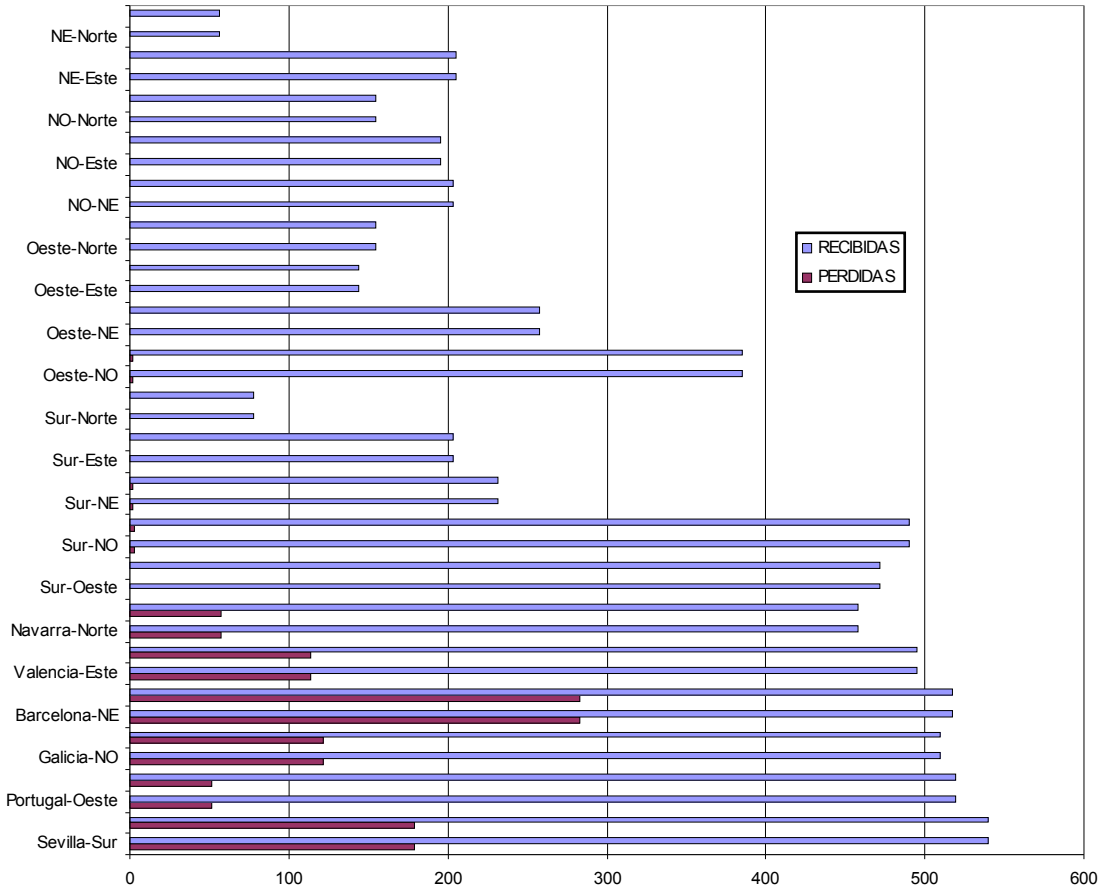



*Figura 15 Enlaces de red*

Por último se muestra en la figura 16 una gráfica de puntos dispersos donde se aprecia la densidad de células transportadas con éxito por la red, así como la variación del retardo.

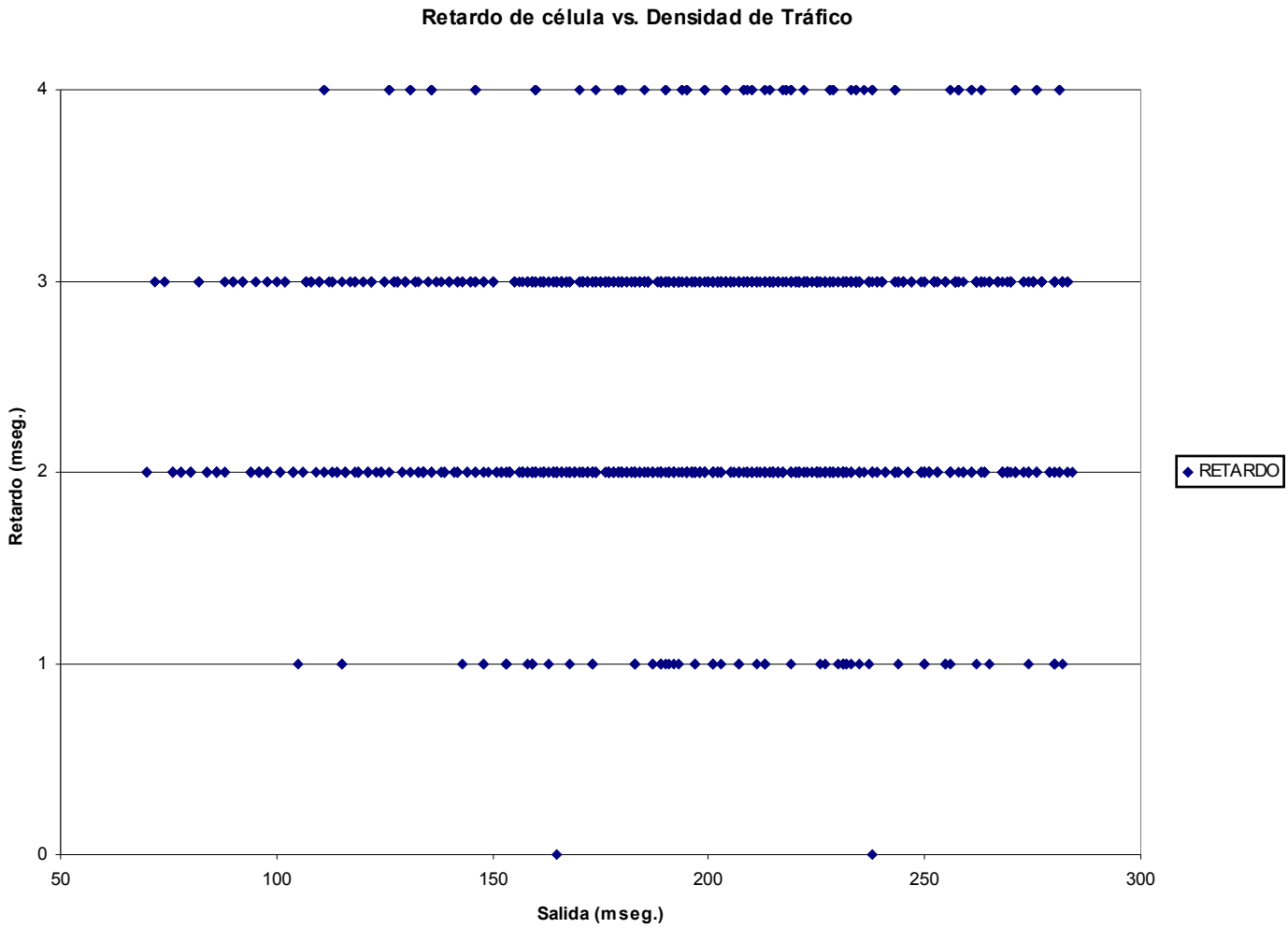



*Figura 16 Retardos medidos en la recepción de células*

Cada punto de la gráfica corresponde a una célula recibida por un abonado. Hay que tener en cuenta que se ha medido con una precisión de 1 milisegundo. El tiempo en la gráfica no está escalado, es tiempo real.

## 5.2 TRÁFICO DE DATOS Y MEZCLA DE TRÁFICOS

Se han simulado 400 milisegundos en los que se han registrado 1365 células transportadas con éxito por la red. Transcurrido ese tiempo, se ha empezado a generar tráfico de voz, y durante 108 milisegundos más han coexistido células de ambos tipos de tráfico. Durante este segundo periodo, se

han alcanzado su destino 869 células de voz y 685 células de datos. El resultado total de carga y células perdidas en cada nodo es bastante parecido al caso anterior, como muestra la figura 17.

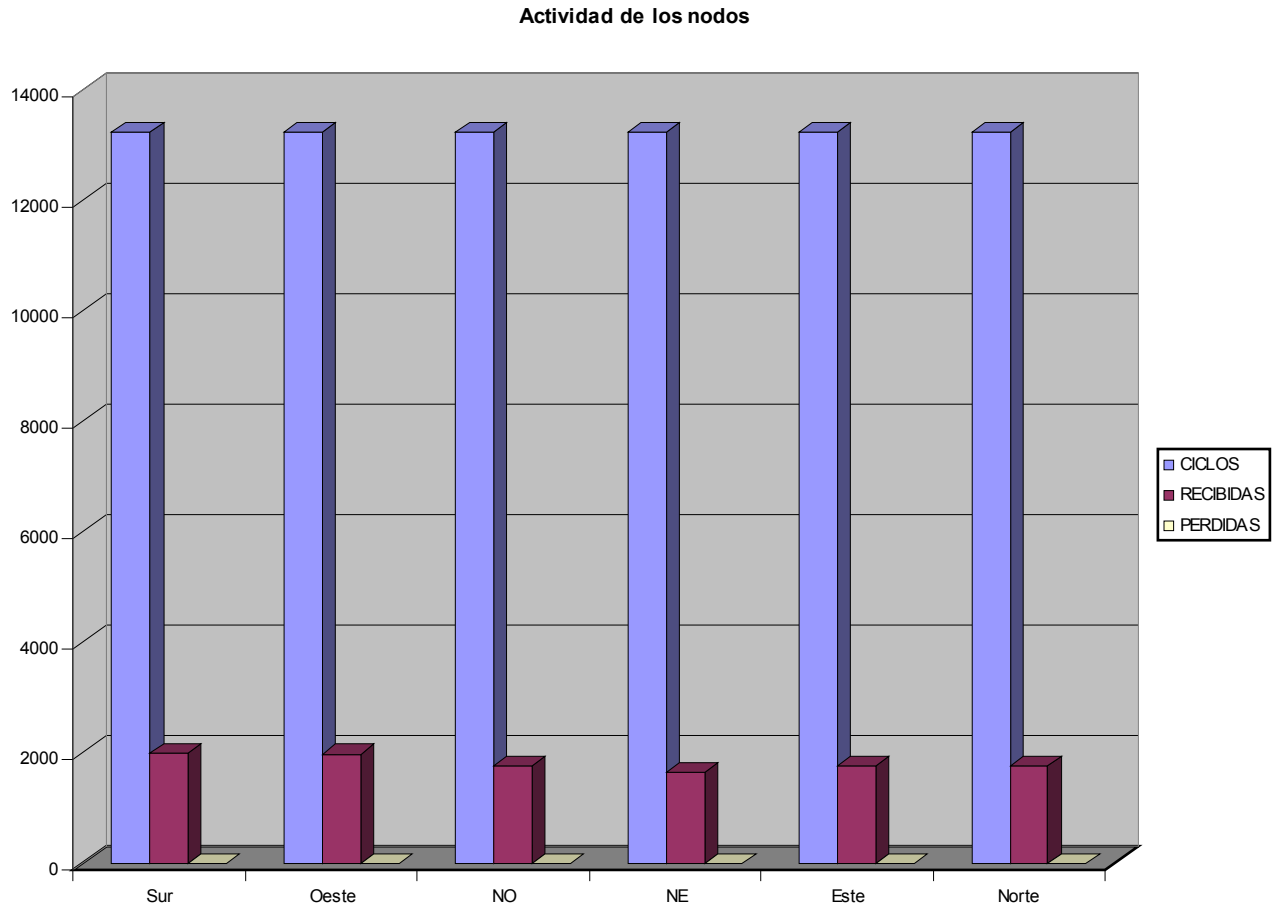



*Figura 17 Actividad de los nodos*

Por otro lado, en los enlaces también aparece un resultado similar. Sólo hay pérdidas en algunos enlaces nodo-central. Esto se muestra en la figura 18.

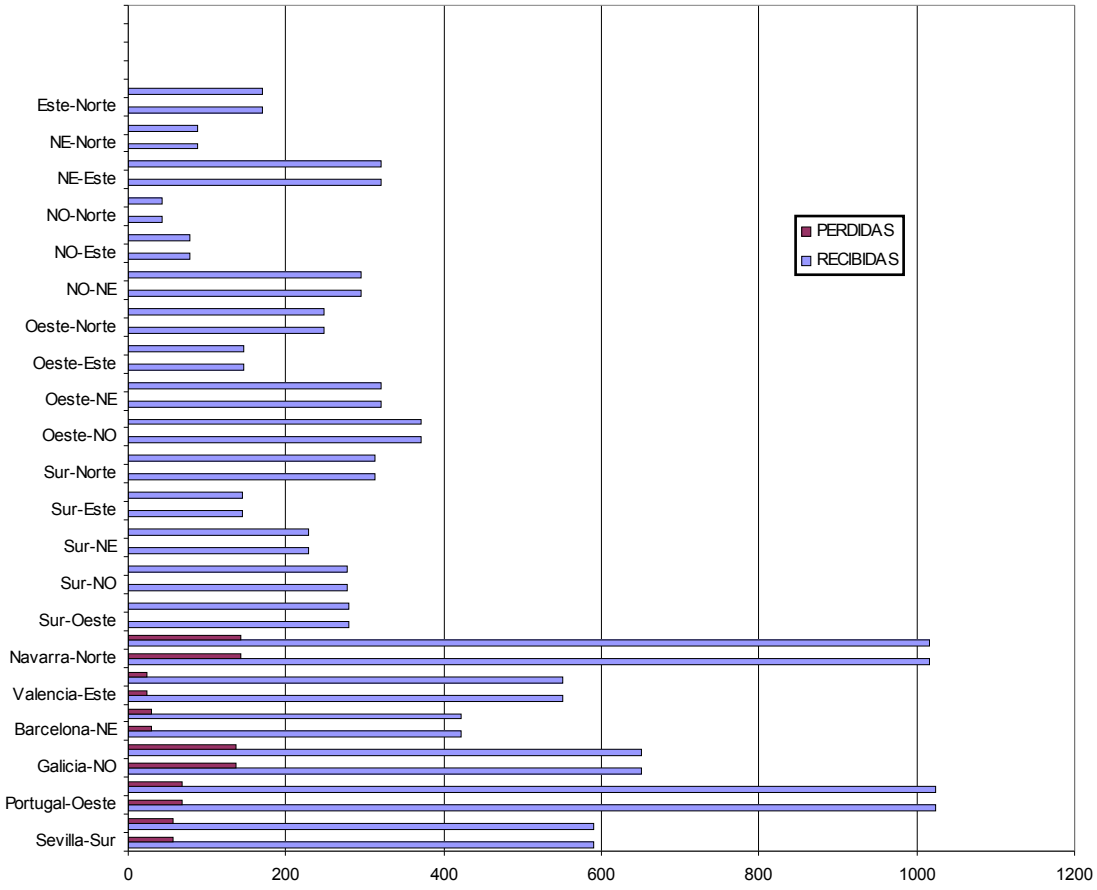



*Figura 18 Células recibidas y perdida en cada enlace de red*

Por último se muestran dos gráficas con densidad de células muestreadas y el retardo de cada una. La primera gráfica corresponde al periodo de simulación completo y puede verse en la figura 19.

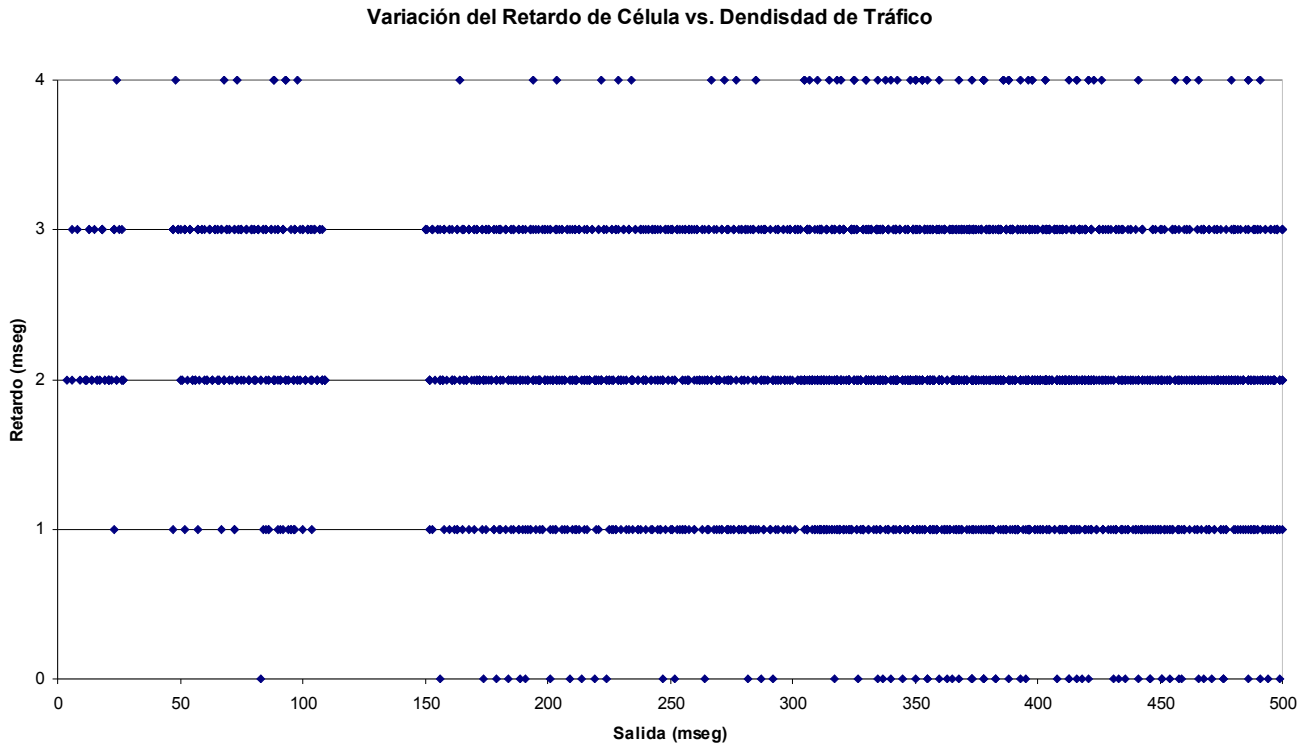



*Figura 19 Retardo de células*

La segunda gráfica muestra la parte de la figura 19 que corresponde a las 869 células de voz transportadas los 108 últimos milisegundos de la simulación. Puede verse en la figura 20.

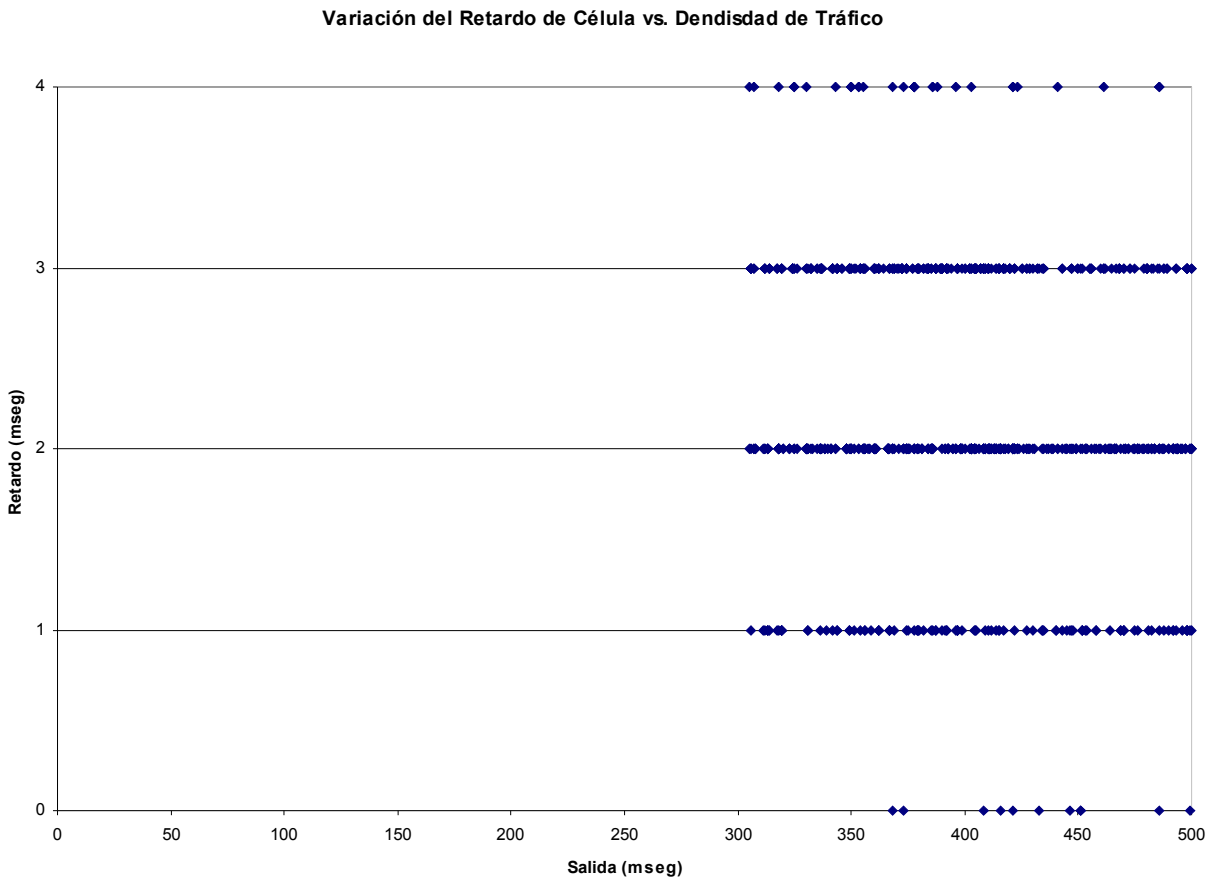



*Figura 20 Células de voz en la simulación con tráfico mezclado*

Haciendo una comparación entre las figuras 16, 19 y 20 se observa que tanto el retraso de células como la variación de éste es parecida, y no cabe hacer distinciones entre células de datos y células de voz. Esto es bastante coherente con la implementación realizada. Al no haber separación de buffers para cada tipo de tráfico a todas las células le afectan por igual los retrasos provocados la conmutación, y concretamente por las colas.

Una diferencia entre los dos tipos de tráfico que sí se ha podido ver durante las simulaciones es que el tráfico de voz por sí solo no es capaz de provocar que las colas de los nodos se llenen, antes bien mantenían niveles

bajos e incluso se vaciaban. Por el contrario, durante la simulación con tráfico de datos era normal ver niveles de cola notablemente más altos.

# 6 CONCLUSIONES Y LÍNEAS DE AMPLIACIÓN

En este trabajo se ha propuesto una metodología para simular comportamientos de una realidad. Esta metodología consiste en identificar y describir tipos de objetos, y diseñar las clases correspondientes mediante un lenguaje de programación orientada a objetos, dotando de autonomía a los que lo requieran.

Se ha aplicado esta metodología de trabajo a las redes ATM, creando modelos de nodos, abonados, enlaces, centrales, etc., e implementándolos en un lenguaje de programación orientado a objetos. Con ello se ha conseguido obtener comportamientos de red. De este modo se ha desarrollado una herramienta con la que poder contrastar algoritmos de optimización de la red y comparar sus resultados. También sería posible probar modelos matemáticos de tráfico de red.

Por otro lado, el problema de optimización en redes ATM es un problema abierto en el que, debido a su complejidad, se hace necesario desarrollar algoritmos metaheurísticos basados en inteligencia artificial.

Si una realidad se puede descomponer en objetos, y éstos se pueden describir y modelar, será posible ensayarlos e incluso estudiar su comportamiento.

Las técnicas avanzadas de inteligencia artificial están orientadas a diseñar objetos dotados de autonomía que ejecuten un algoritmo para búsqueda de soluciones. Del mismo modo que con redes ATM, el método de trabajo puede servir para conseguir comportamientos de esos objetos diseñados para encontrar soluciones.

Así por ejemplo, se pueden diseñar agentes que compongan un sistema multi-agente. Mediante programación en Java u otro lenguaje orientado a objetos, se podrían implementar cada clase de agente, crear cuantas instancias se quisieran de cada tipo, estando dotados de autonomía. De este modo, se conseguirían comportamientos de ese sistema multi-agente. Y en definitiva el método de trabajo que se propone en este proyecto podría utilizarse para el diseño de un sistema multi-agente que incorporara un escenario de simulación y optimización.

En esta misma linea de ampliación se podría trabajar en dotar a los objetos, no sólo de autonomía, sino también de conocimiento y aprendizaje. Así, cabría aplicar bases de datos orientadas a objetos como PostgreSQL, junto con un lenguaje como Java. El método propuesto se utiliza para diseñar objetos a los que dotar de autonomía. Sin embargo sería necesario buscar un método complementario con el que diseñar estructuras de conocimiento en bases de datos, y así poder dotar a los objetos de capacidad de aprendizaje.

# 7 ANEXO I: DIAGRAMAS UML DEL SIMULADOR

En este anexo se ha querido reunir por motivos de orden todos los diagramas UML (*Unified Modeling Language*) de las clases programadas.

Se incluye en primer lugar un índice de diagramas para facilitar la localización de los diagramas. Como puede observarse se ha optado por ordenarlos alfabéticamente según los nombres de las clases. Las clases contenidas en otras se nombran usando el nombre de la clase contenedora como prefijo, tal y como lo hace el compilador del lenguaje java.

Se ha intentado ajustar el tamaño y la orientación de los diagramas de modo que sea lo más legible posible el texto que contienen, y a la vez se pueda ver el diagrama completo. Dado que esto no es posible con algunos de ellos, se decide en cada caso si recortar o dividir la imagen, según el interés que pueda tener cada uno. En buena parte de ellas se ha seccionado el diagrama y se ha recolocado una de las partes, para que cupiera. En estos casos se indica la sección realizada con unas líneas.

La interpretación de los diagramas UML es sencilla. En cada uno se pueden identificar las siguientes partes:

- ✓ Enmarcado en el centro con el nombre de la clase se encuentran atributos, constructores y métodos.
- ✓ Encima de la clase, indicado por una flecha, se indica la clase padre. Por defecto, si no se ha dispuesto otra cosa, la clase padre será Object del paquete “java.lang” del kit de desarrollo.

- ✓ A la izquierda del recuadro de la clase, y agrupadas por paquetes, se encuentran las clases usadas para declarar atributos, así como las que poseen atributos de la clase representada por el diagrama. Unas y otras se distinguen por el sentido de las flechas.

- ✓ A la derecha del recuadro de la clase, y agrupadas por paquetes, se encuentran las clases en las cuales algún método acceden a atributos o métodos de la clase representada en el diagrama. Así mismo, se encuentran aquellas clases con algún atributo o método que accedido desde la clase representada.
  Unas y otras se diferencian por el sentido de las flechas.

A modo de ejemplo se interpreta el diagrama de una clase no vista. Corresponde a la imagen de un nodo dentro de la imagen de red, que sí se explicó.

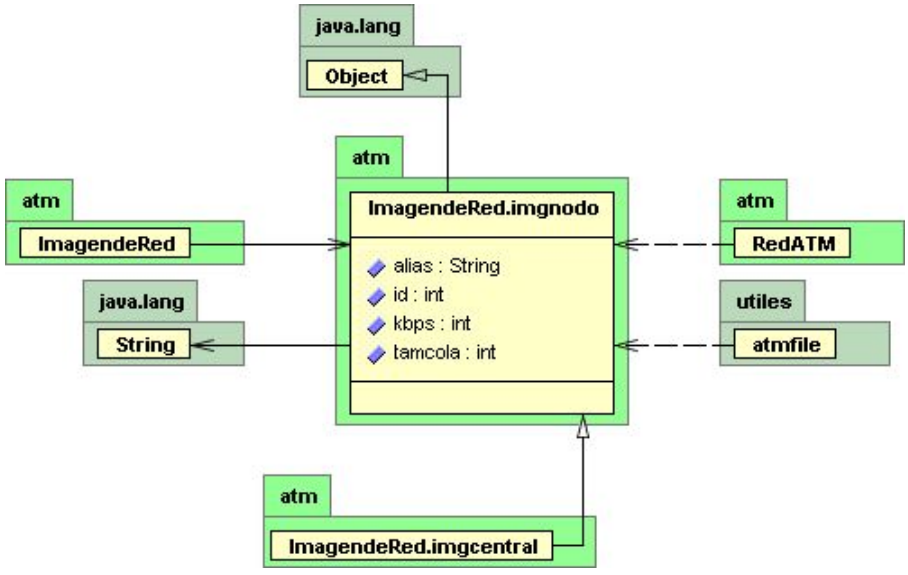



*Figura 21 Ejemplo de interpretación de diagramas UML*

Esta clase llamada “imgnodo” no posee constructor ni métodos, solo cuatro atributos que son “alias”, “id”, “kbps” y “tamcola”. El papel que

desempeñan se deduce inmediatamente recordando lo explicado de la clase Nodo.

La clase “imgnodo” es una clase interna de “ImagendeRed”, que pertenece al paquete “atm” al igual que la mayoría de las clases programadas y descritas. No se especifica ninguna clase padre, luego esta es directamente “Object”.

Por otro lado también se indica que “imgnodo” tiene una clase hija llamada “imgcentral”, definida igualmente dentro de la clase “ImagendeRed”.

En la parte izquierda del diagrama se observa que esta clase emplea la clase “String” del kit de desarrollo de Java. A su vez es empleada por la clase “ImagendeRed”, que concretamente tendrá tantas imágenes de nodo como nodos vaya a haber en la red.

Por último, en la clase derecha se puede ver que “imgnodo” no accede a ningún atributo o método de ninguna otra clase. Por el contrario, hay dos clases que sí acceden a sus atributos (no puede ser a sus métodos por que no tiene). Estas clases son “RedATM” y “atmfiles”. Esta segunda clase es de las pocas clases programadas que no están en el paquete “atm”, sino en otro también creado que se llama “utiles”.

# Índice de Diagramas UML

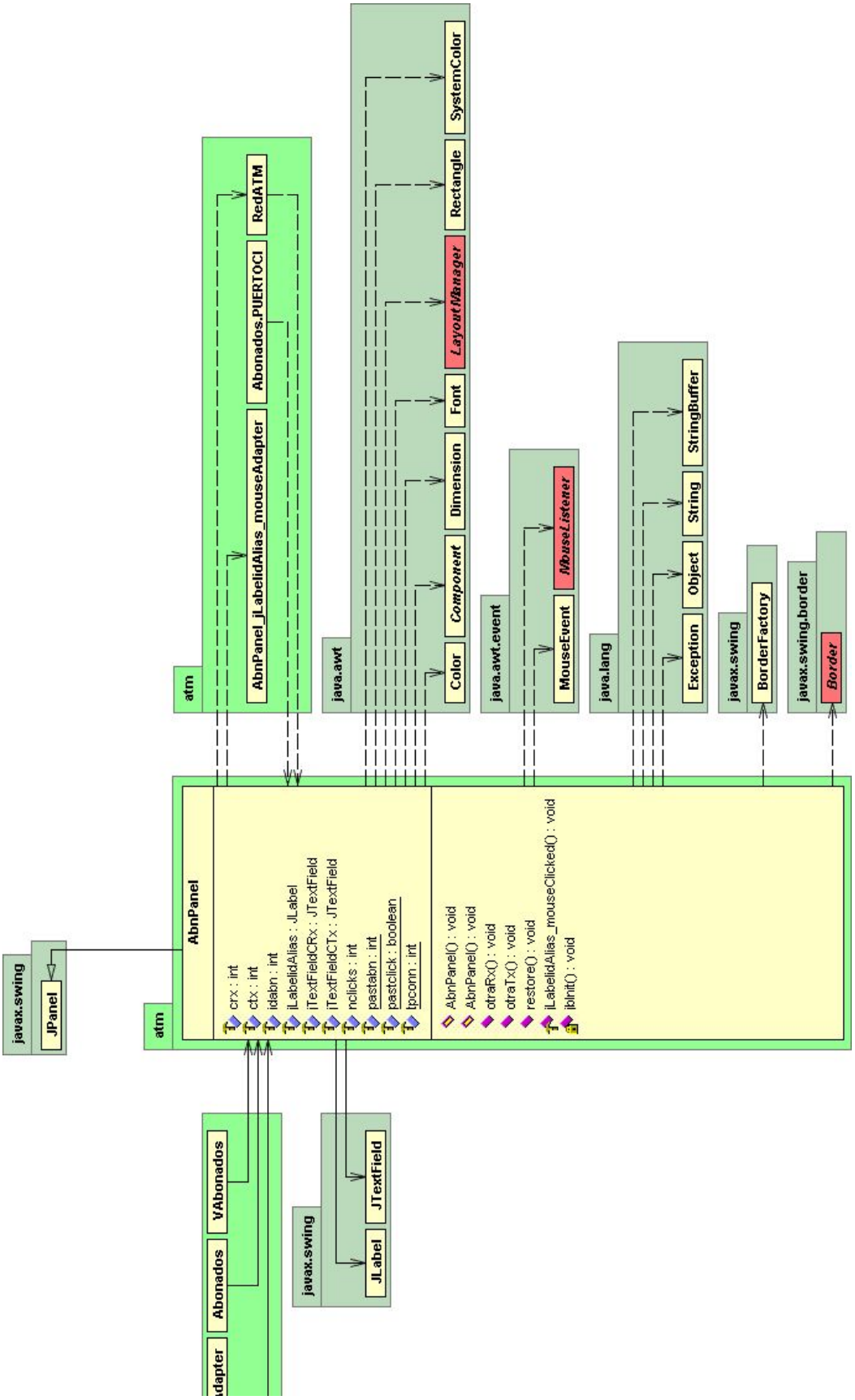



*Figura 22 AbnPanel*

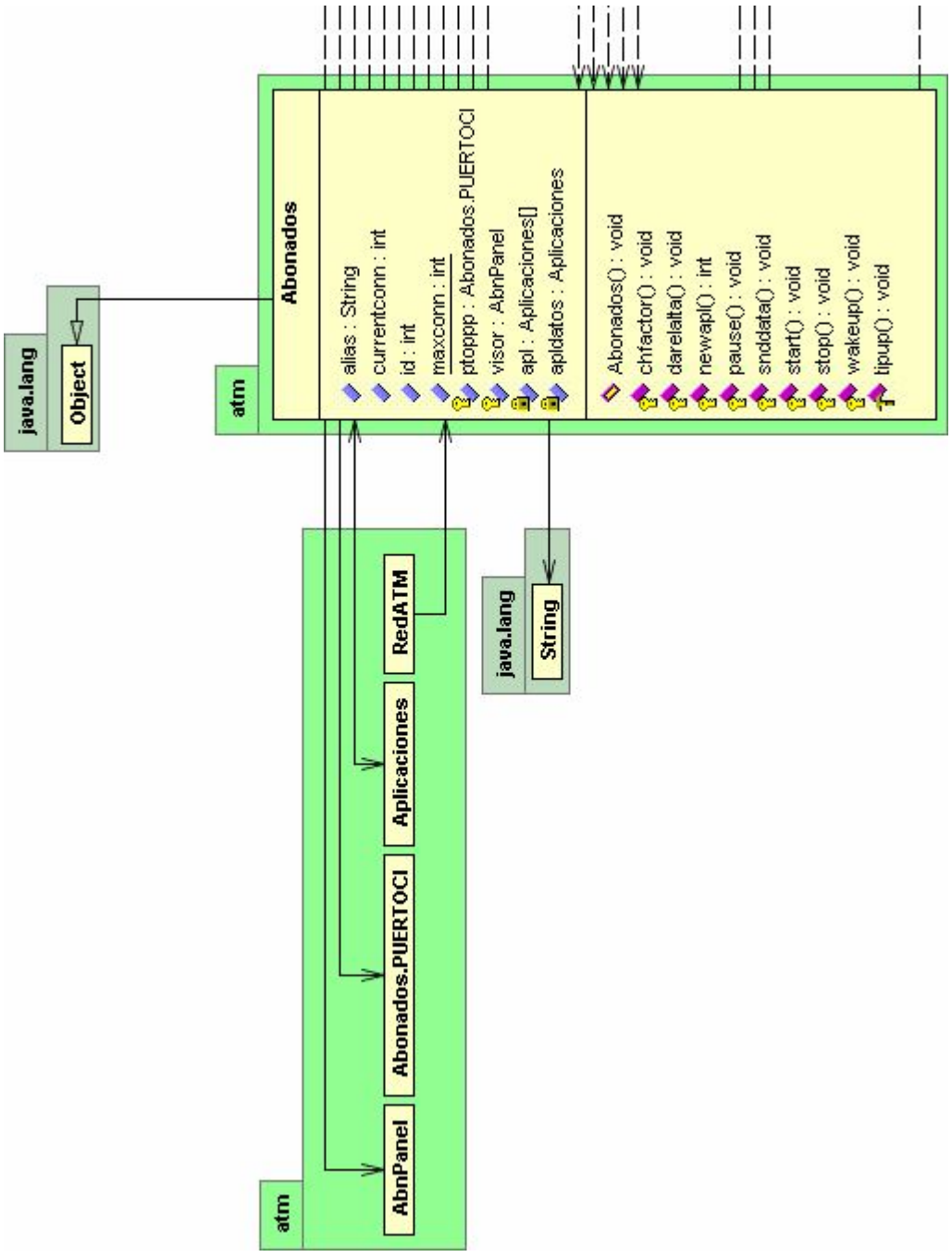



*Figura 23 Abonados (I)*

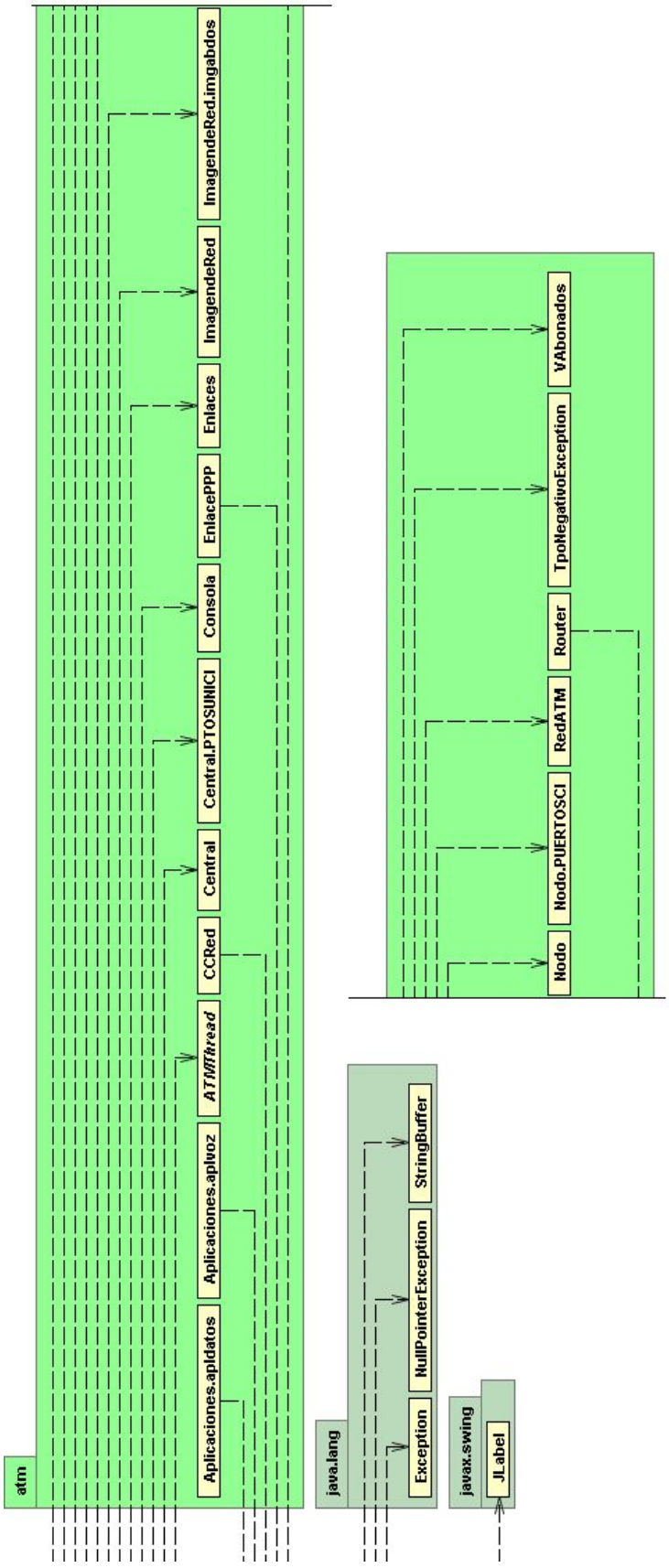



*Figura 24 Abonados (II)*

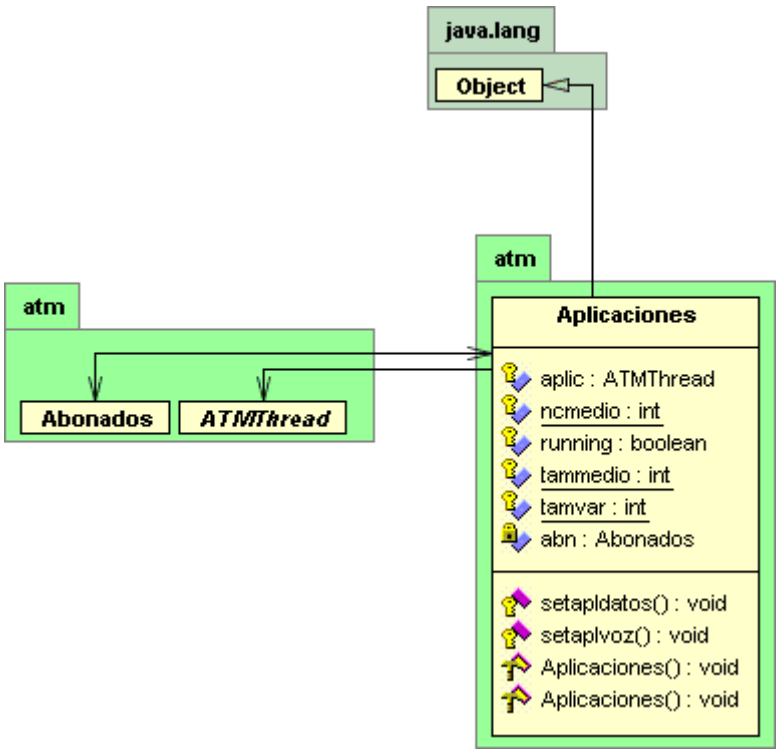



*Figura 25 Aplicaciones*

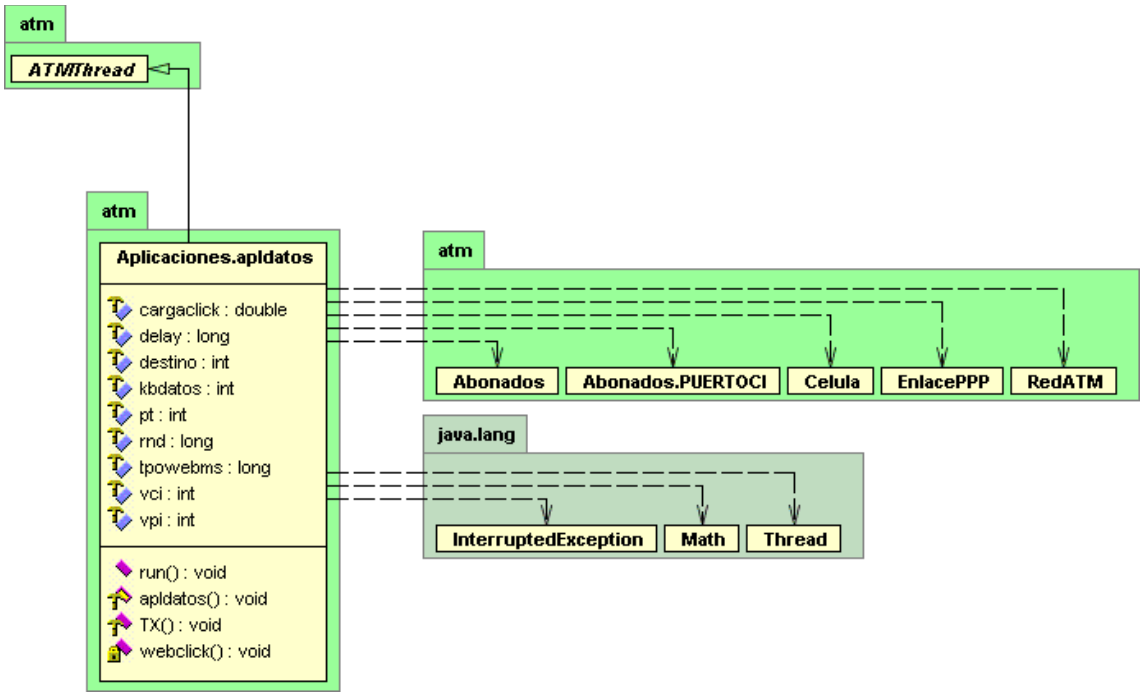



*Figura 26 Aplicaciones.apldatos*

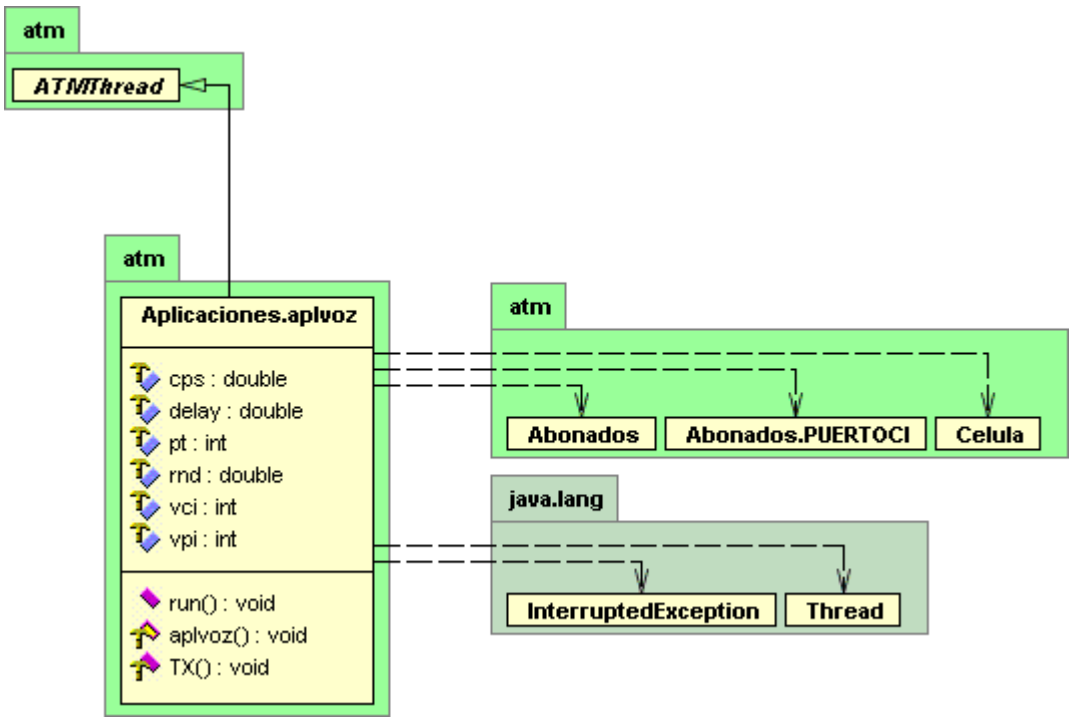



*Figura 27 Aplicaciones.aplvoz*

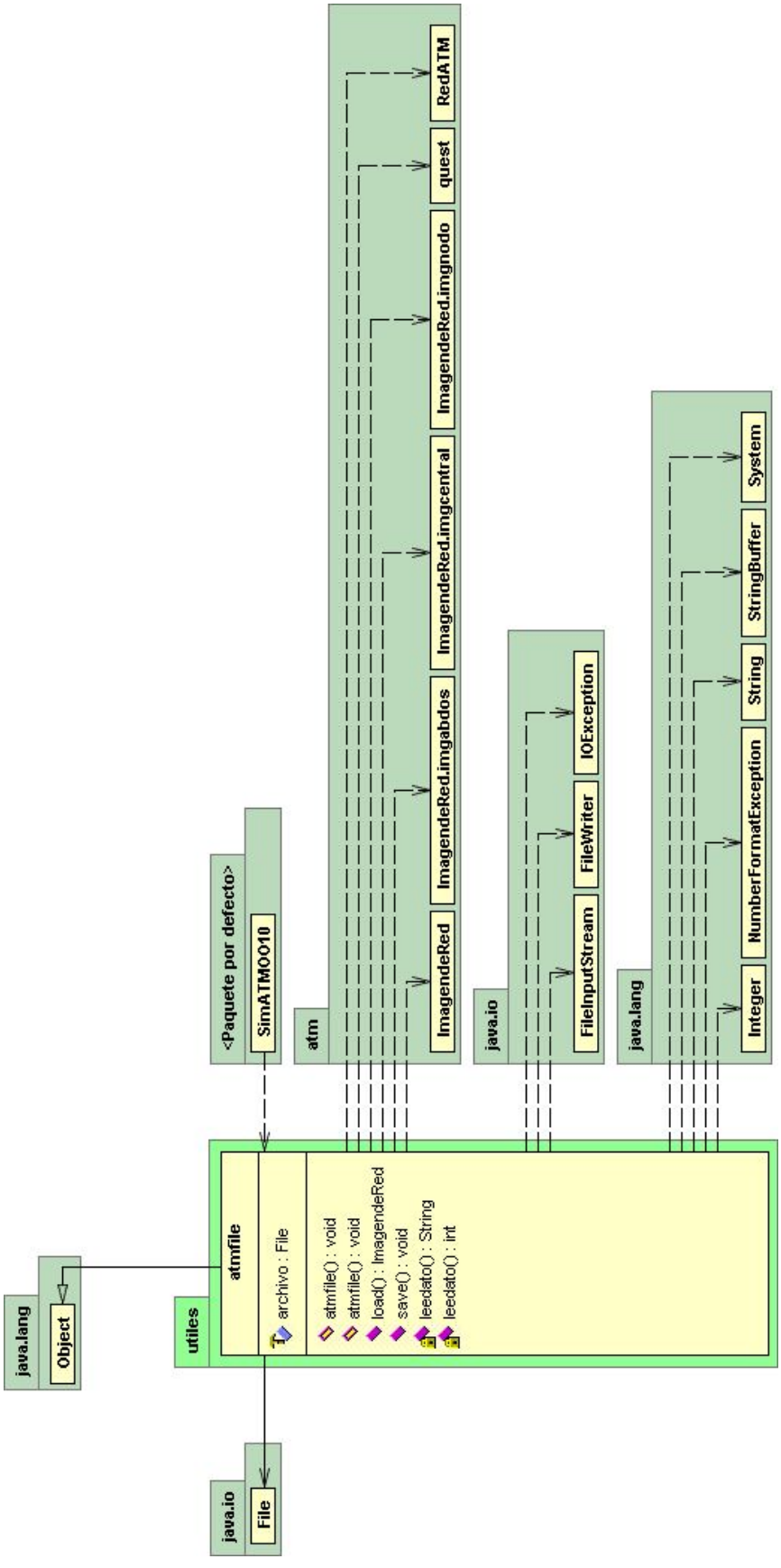



*Figura 28 atmfile*

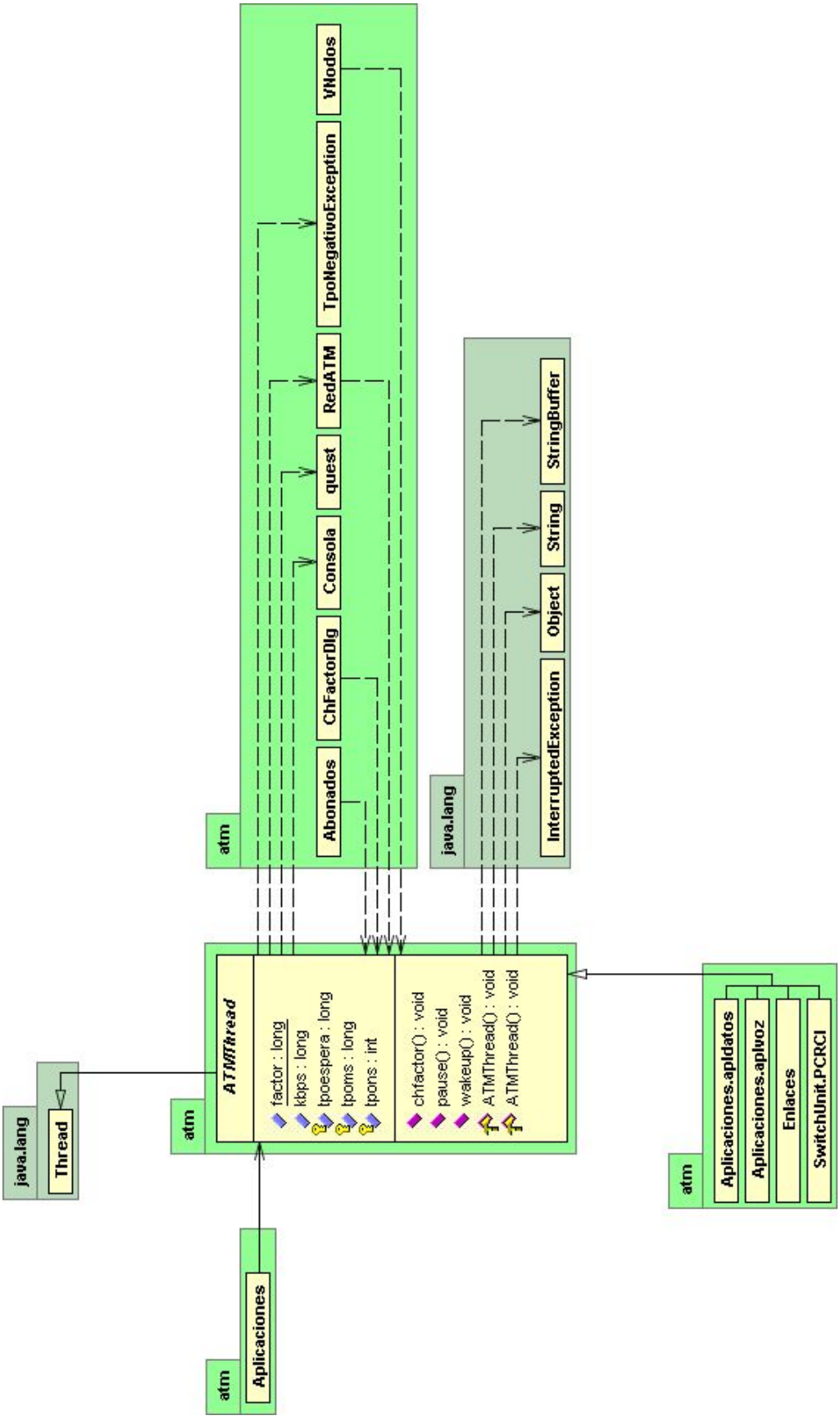



*Figura 29 ATMThread*

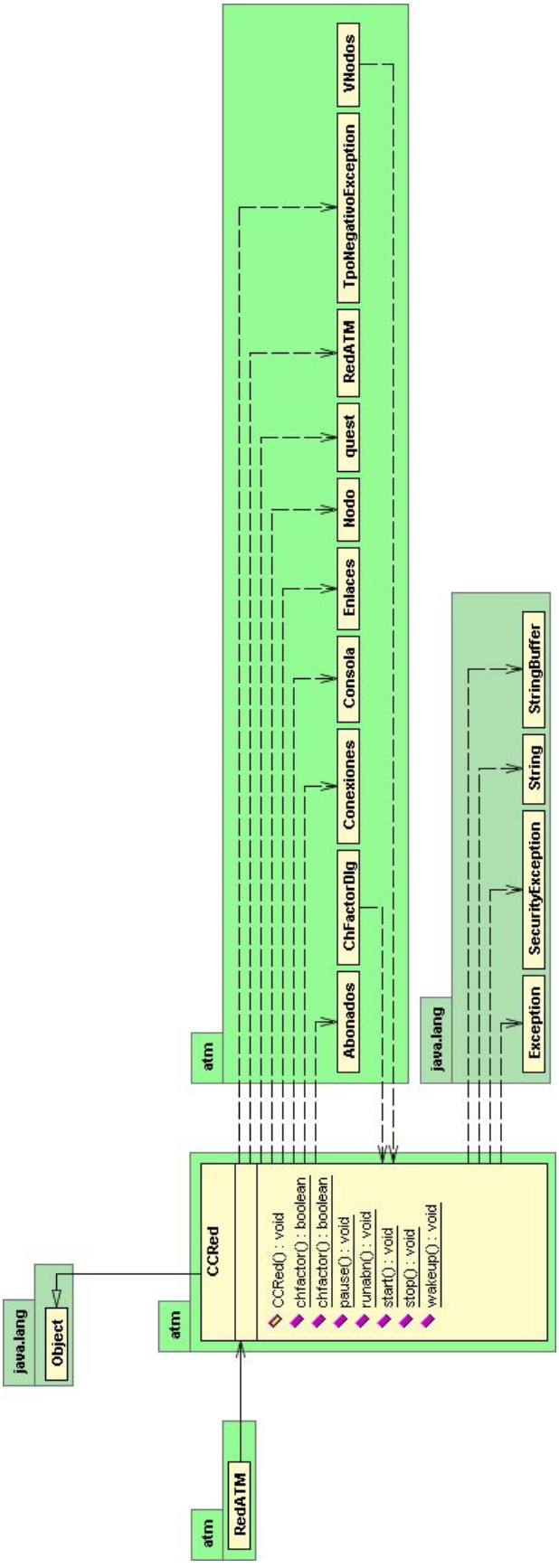



*Figura 30 CCRed*

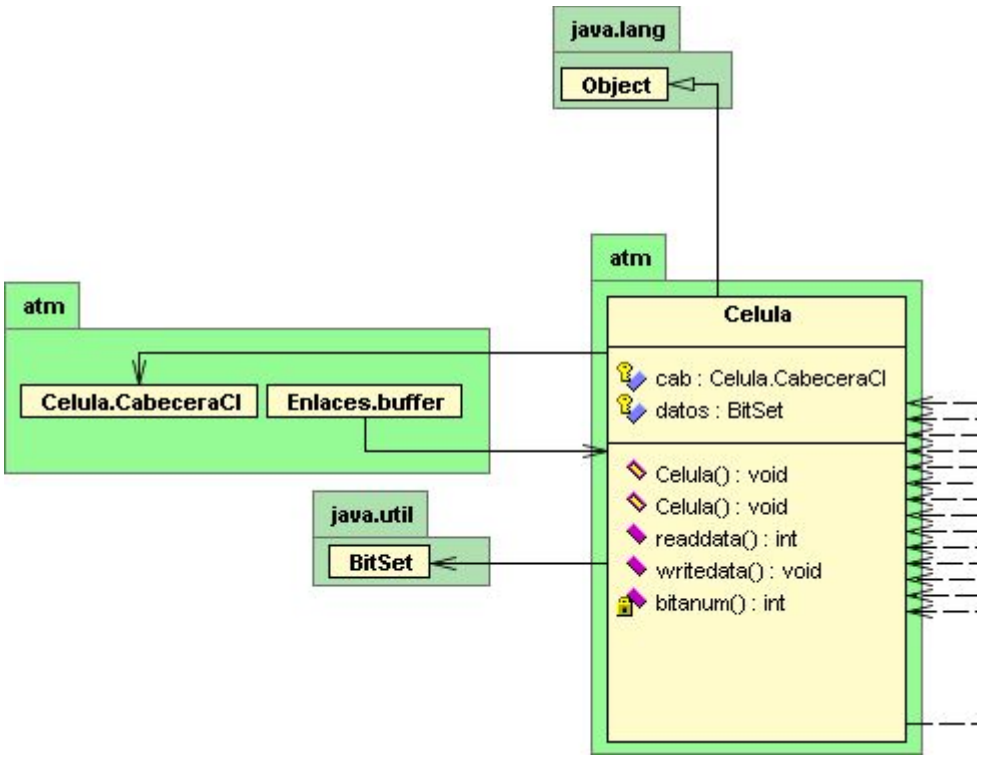



*Figura 31 Celula (I)*

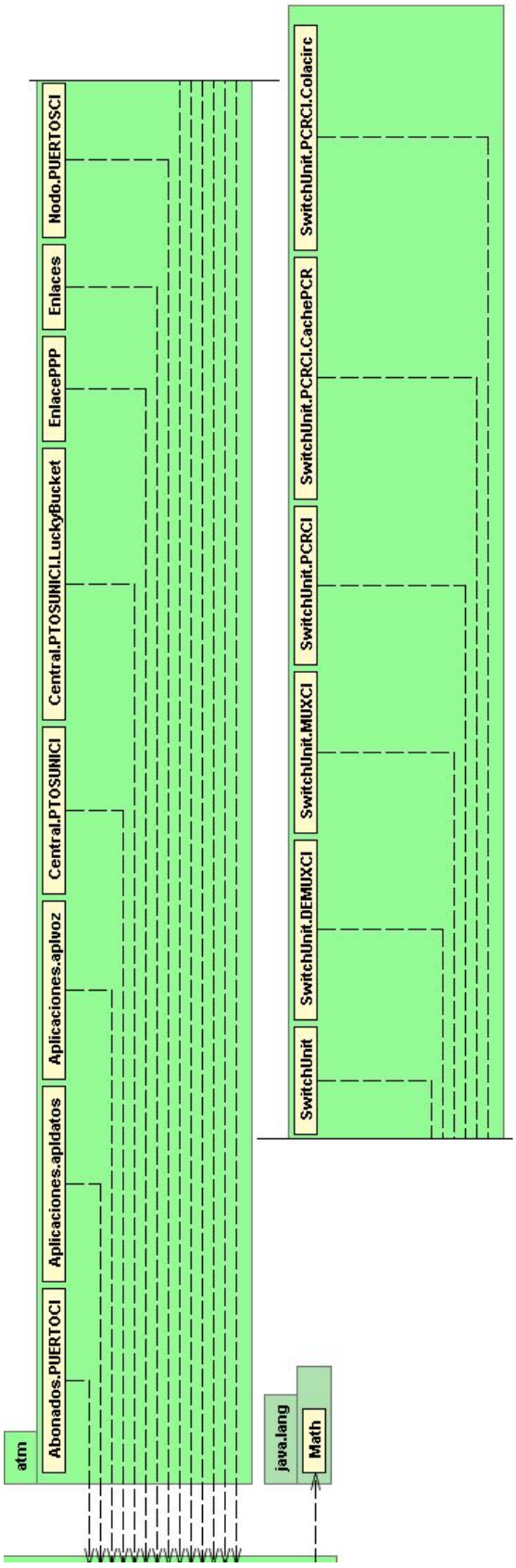



*Figura 32 Celula (II)*

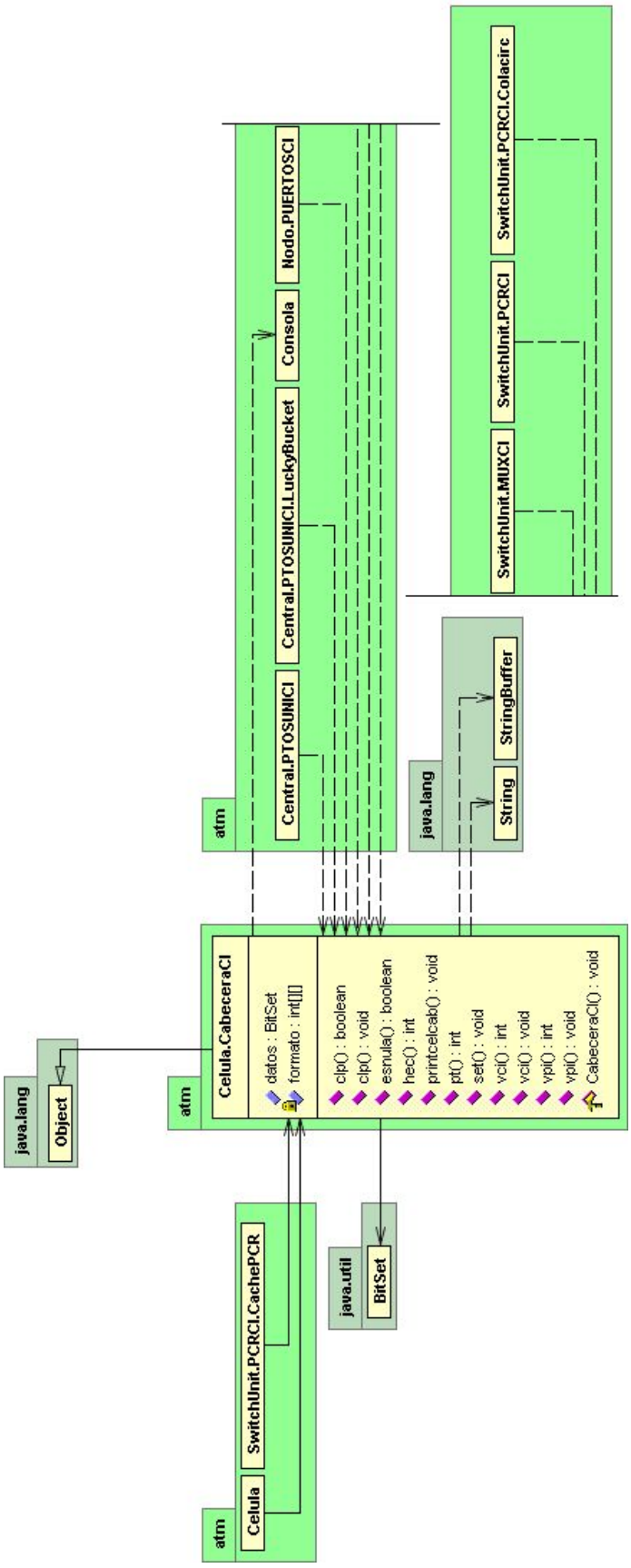



*Figura 33 Celula.CabeceraCI*

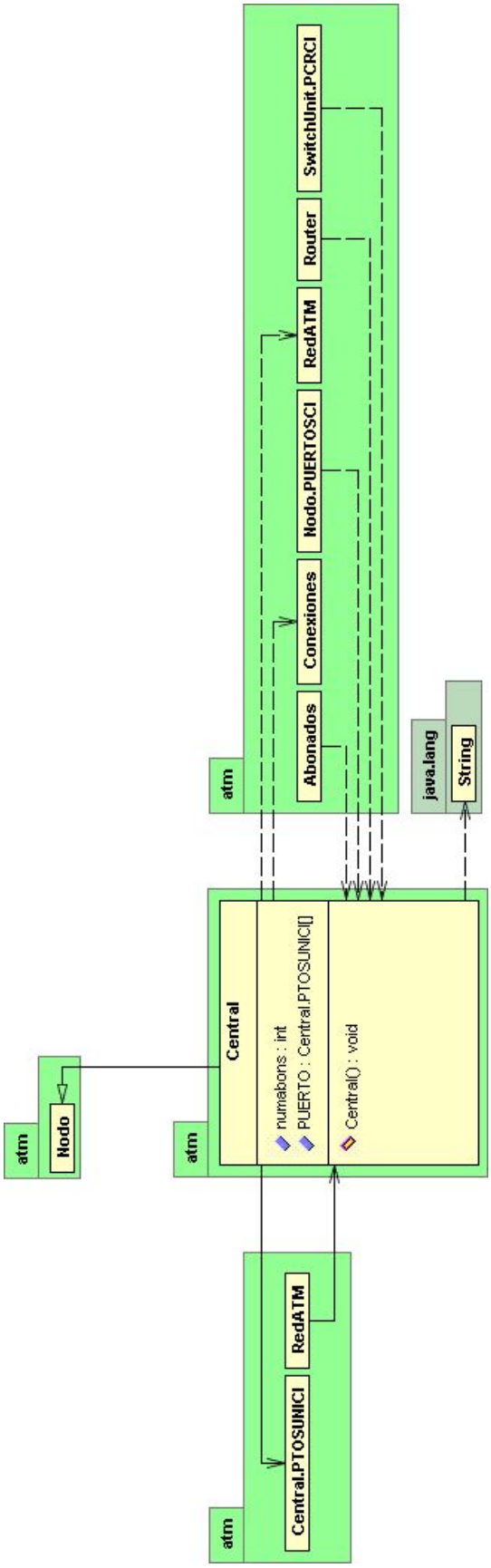



*Figura 34 Central*

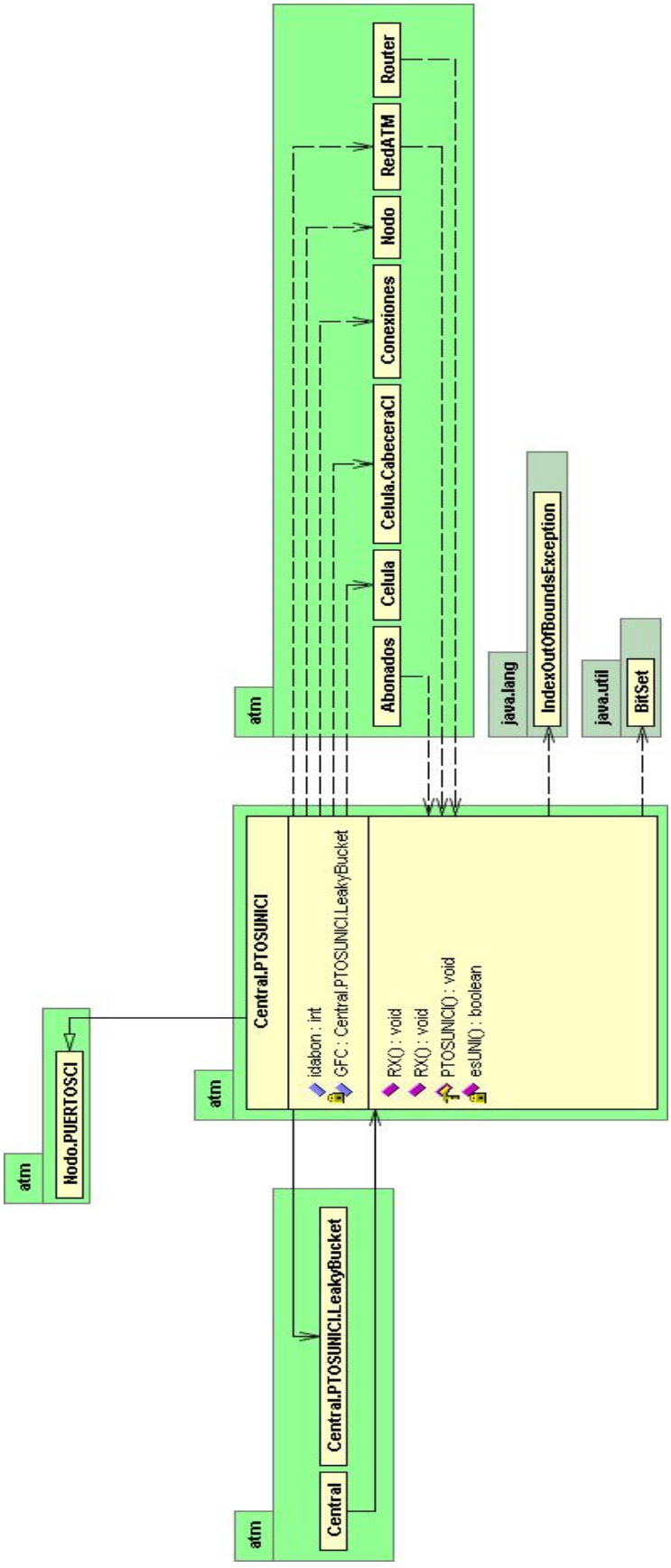



*Figura 35 Central.PTOSUNICI*

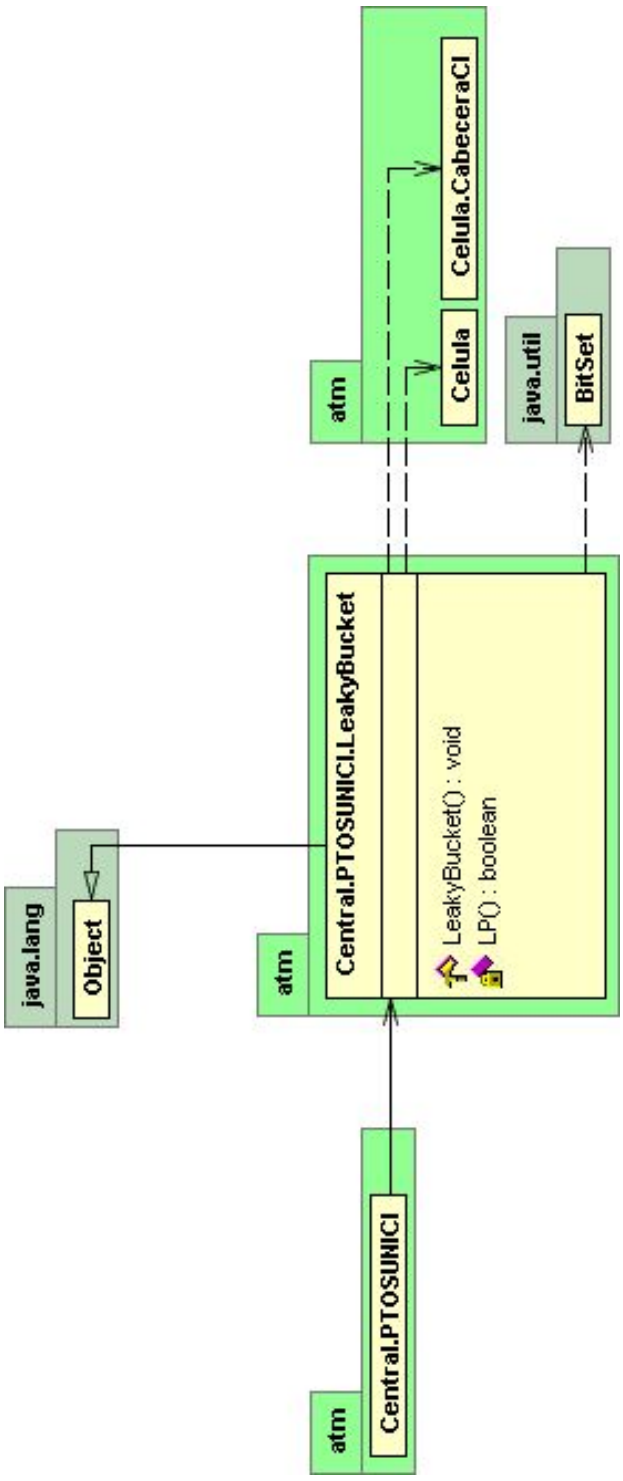



*Figura 36 Central.PTOSUNICI.LeakyBucket*

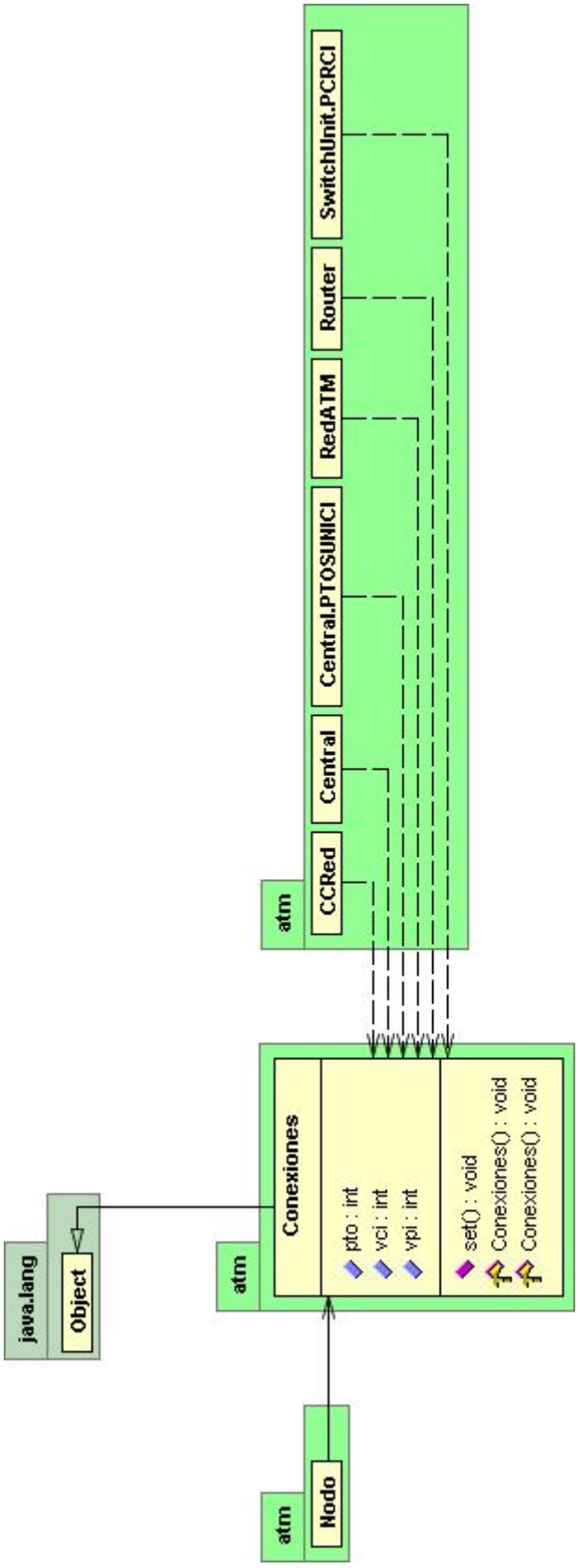



*Figura 37 Conexiones*

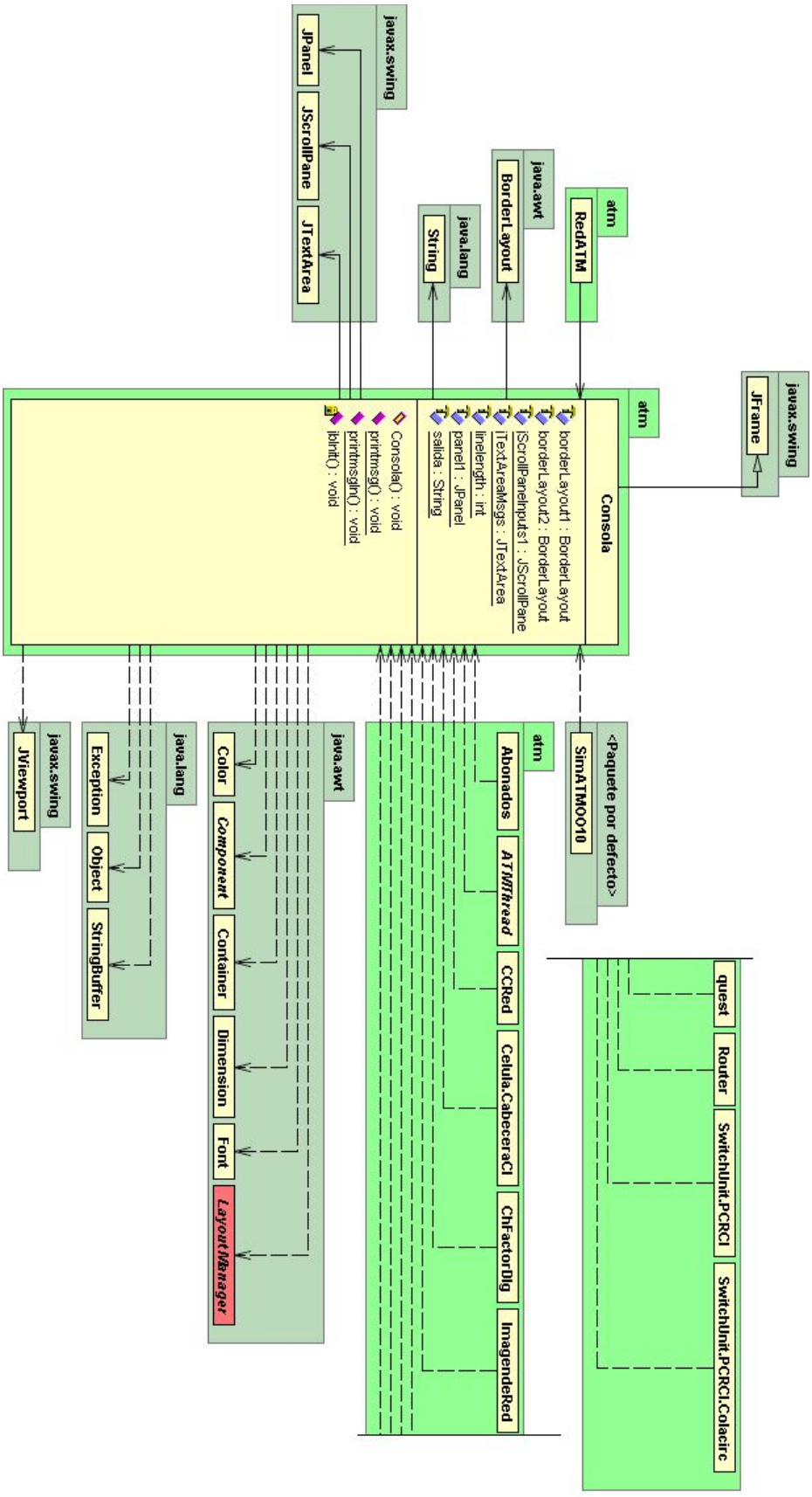



*Figura 38 Consola*

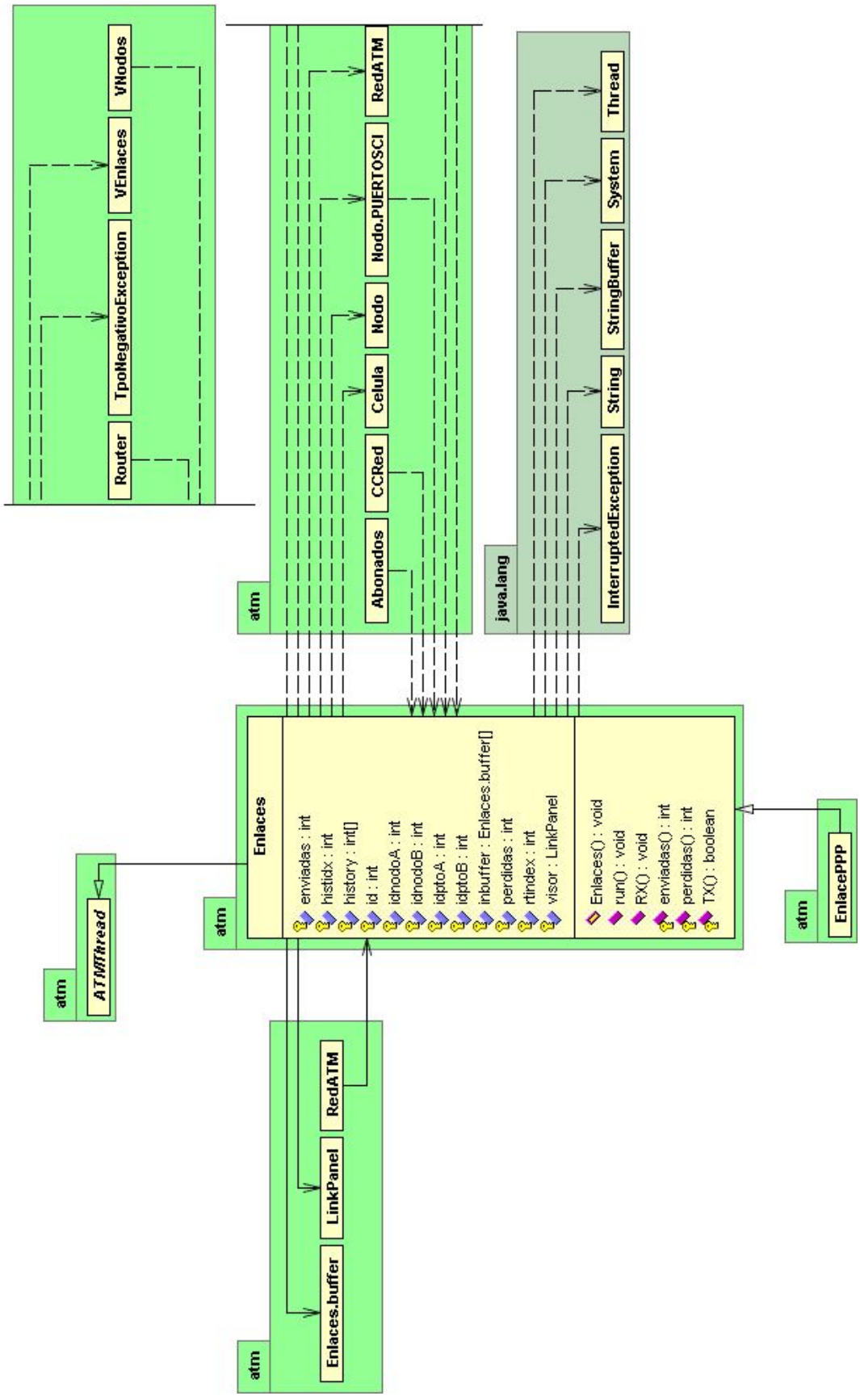



*Figura 39 Enlaces*

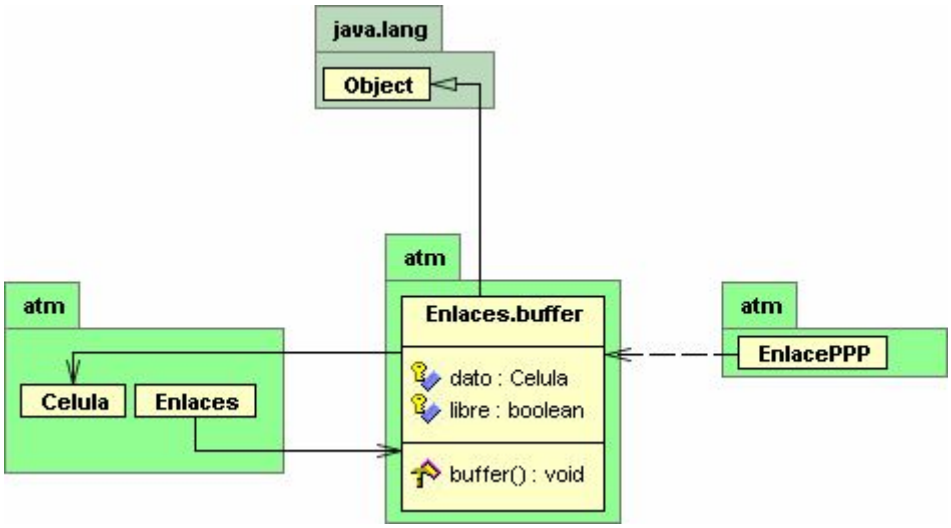



*Figura 40 Enlaces.buffer*

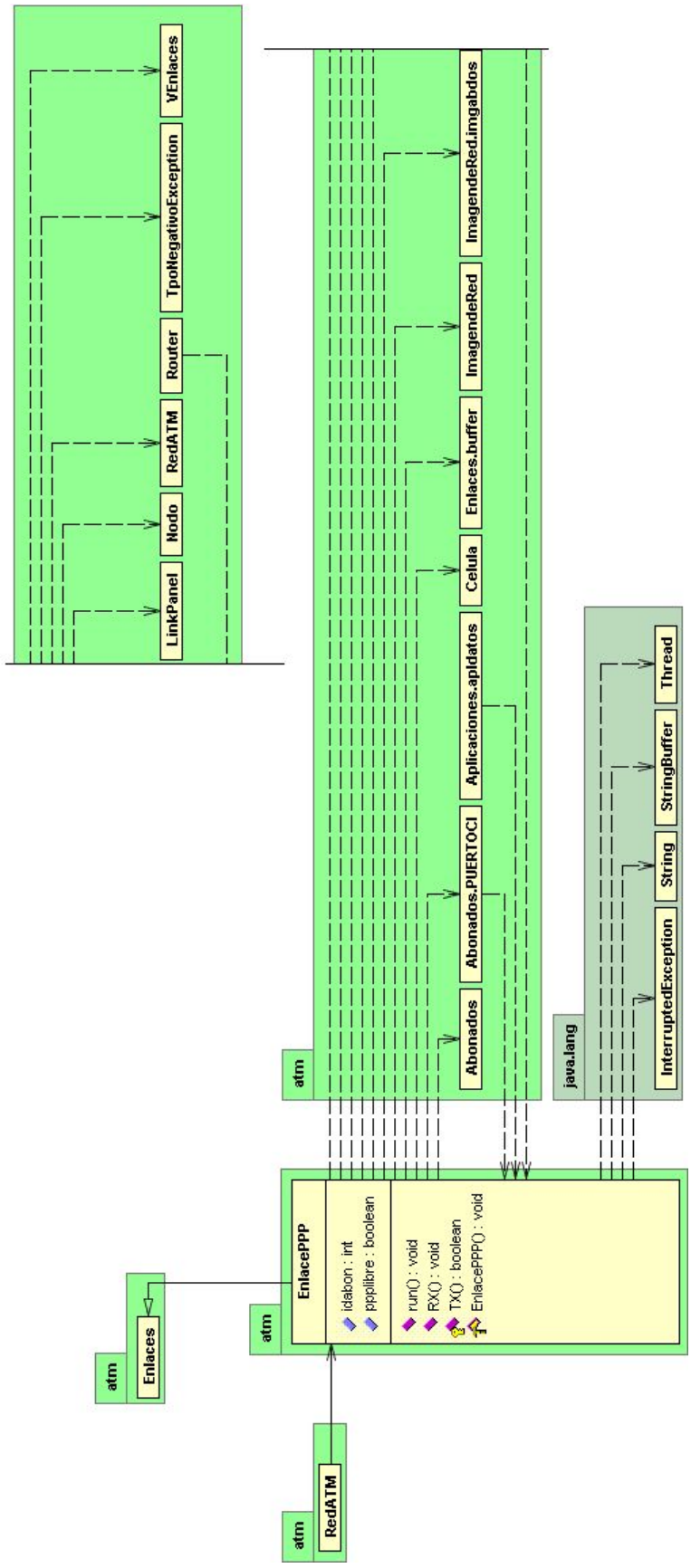



*Figura 41 EnlacePPP*

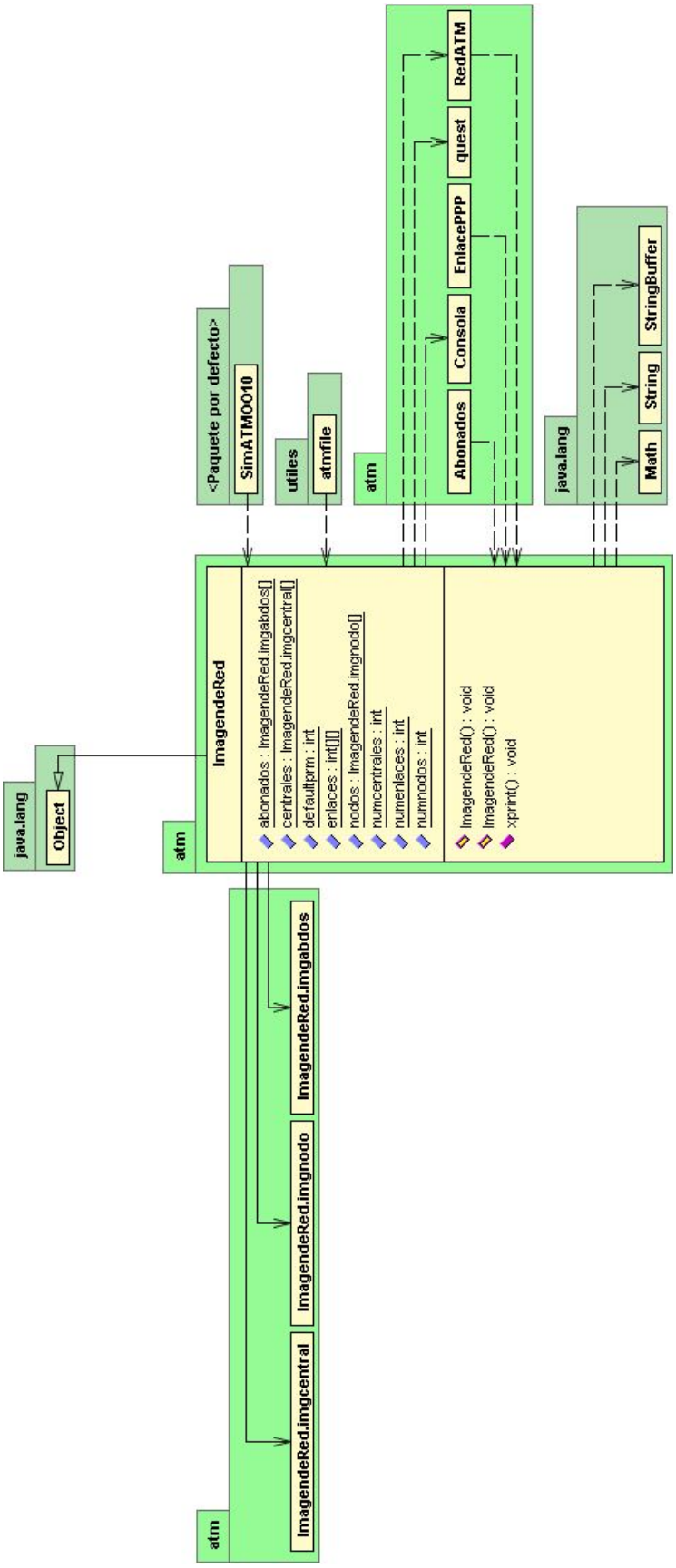



*Figura 42 ImagendeRed*

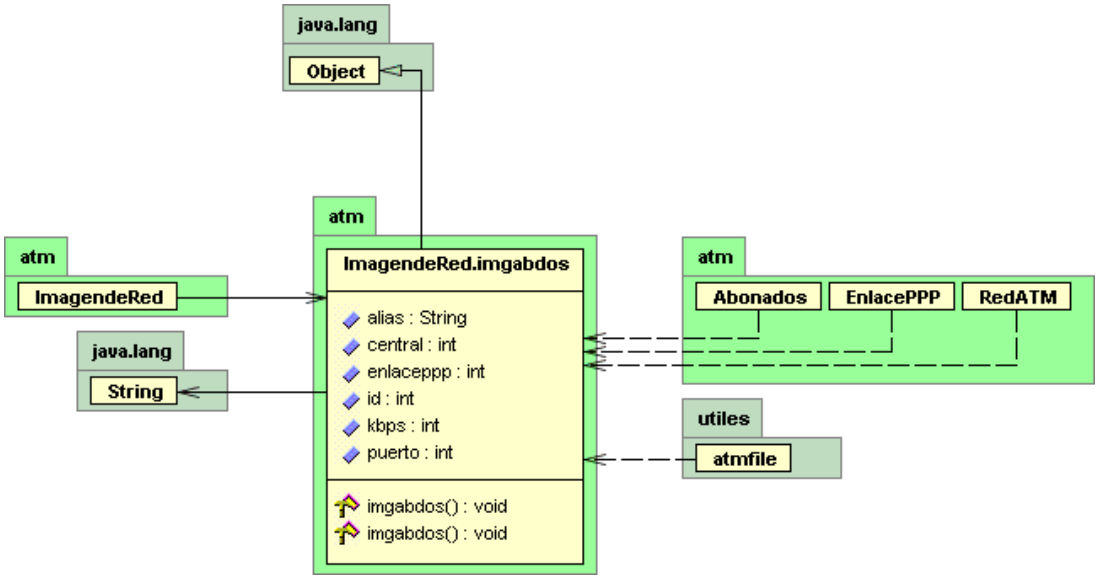



*Figura 43 ImagendeRed.imgabdos*

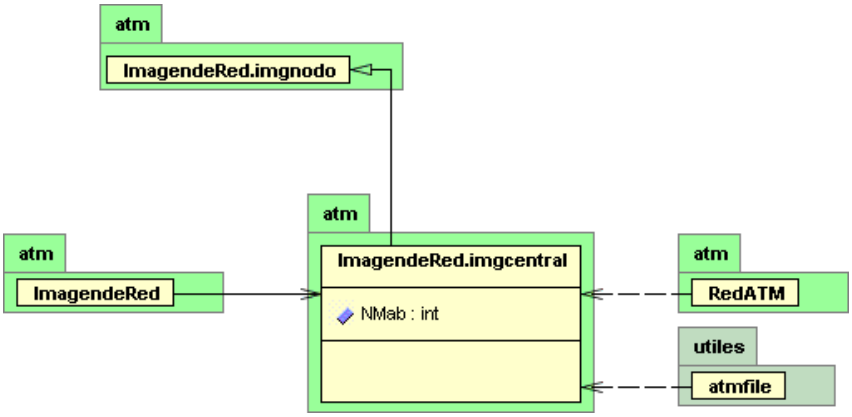



*Figura 44 ImagendeRed.imgcentral*

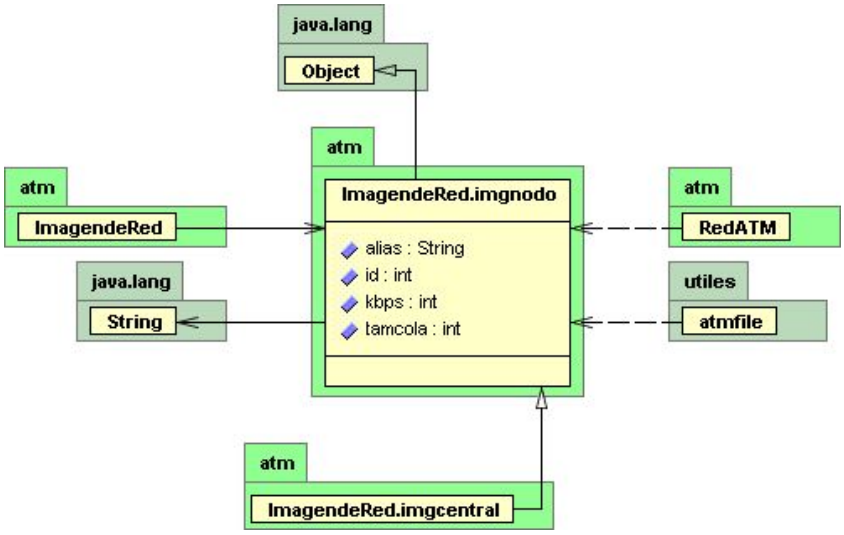



*Figura 45 ImagendeRed.imgnodo*

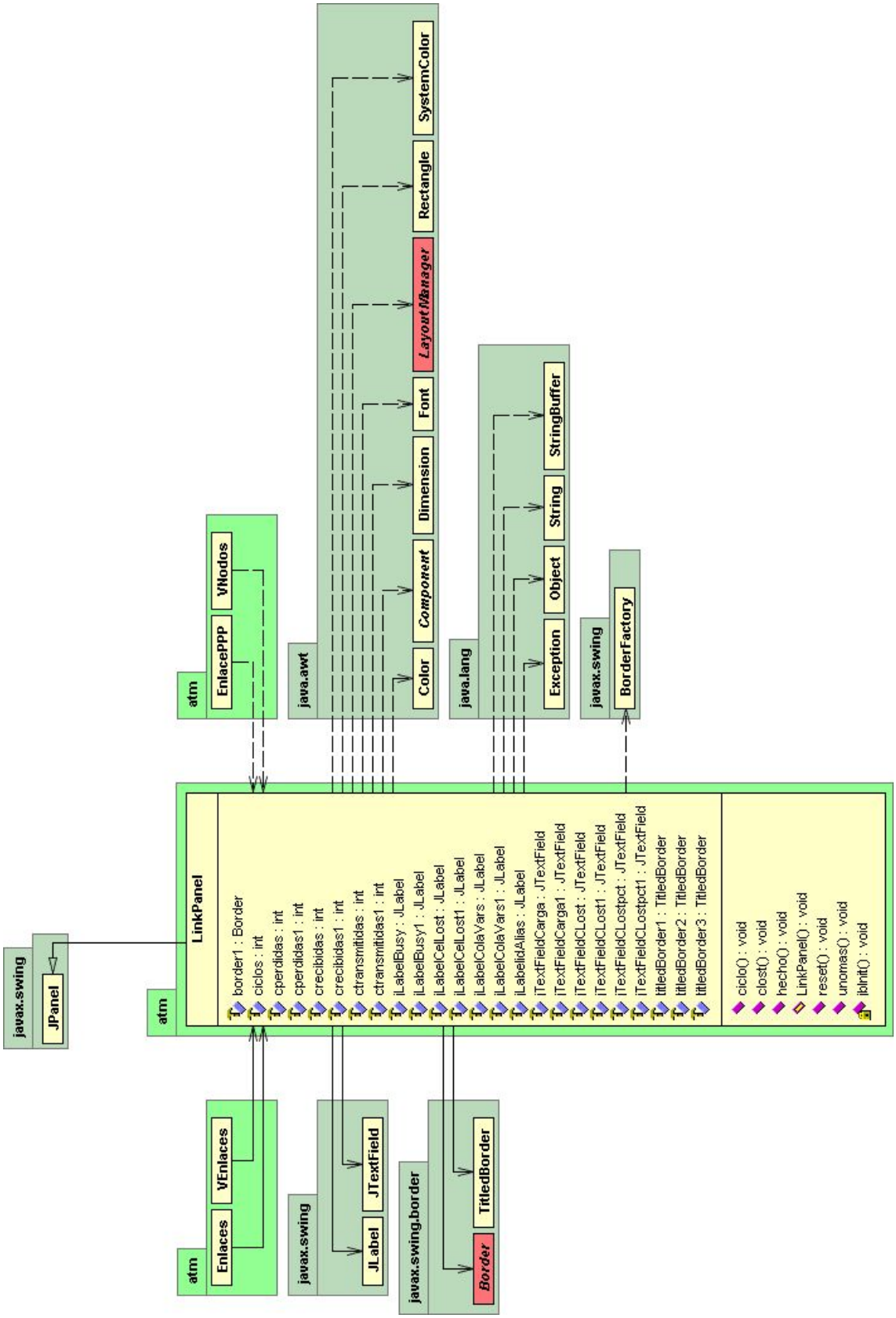



*Figura 46 LinkPanel*

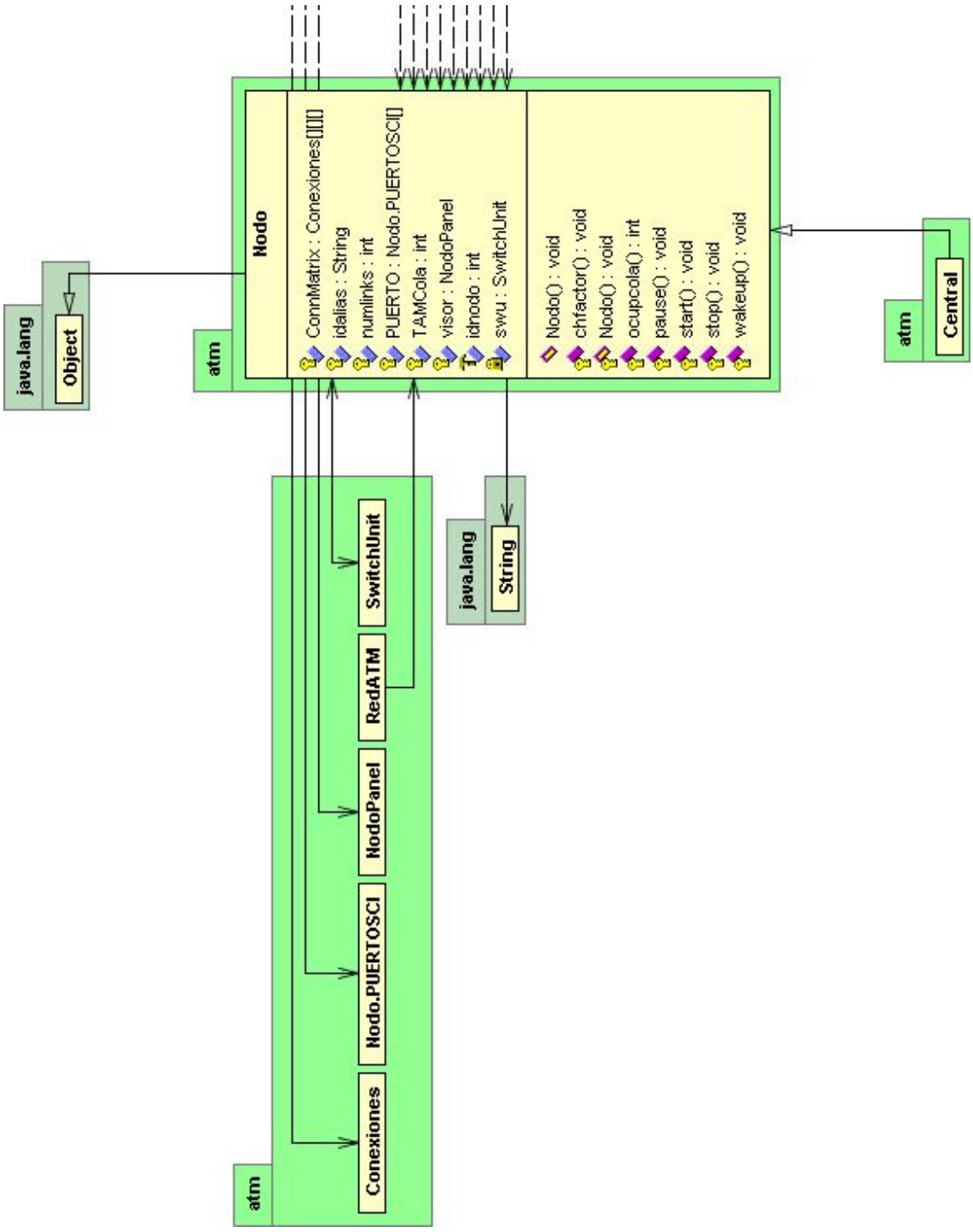



*Figura 47 Nodo (I)*

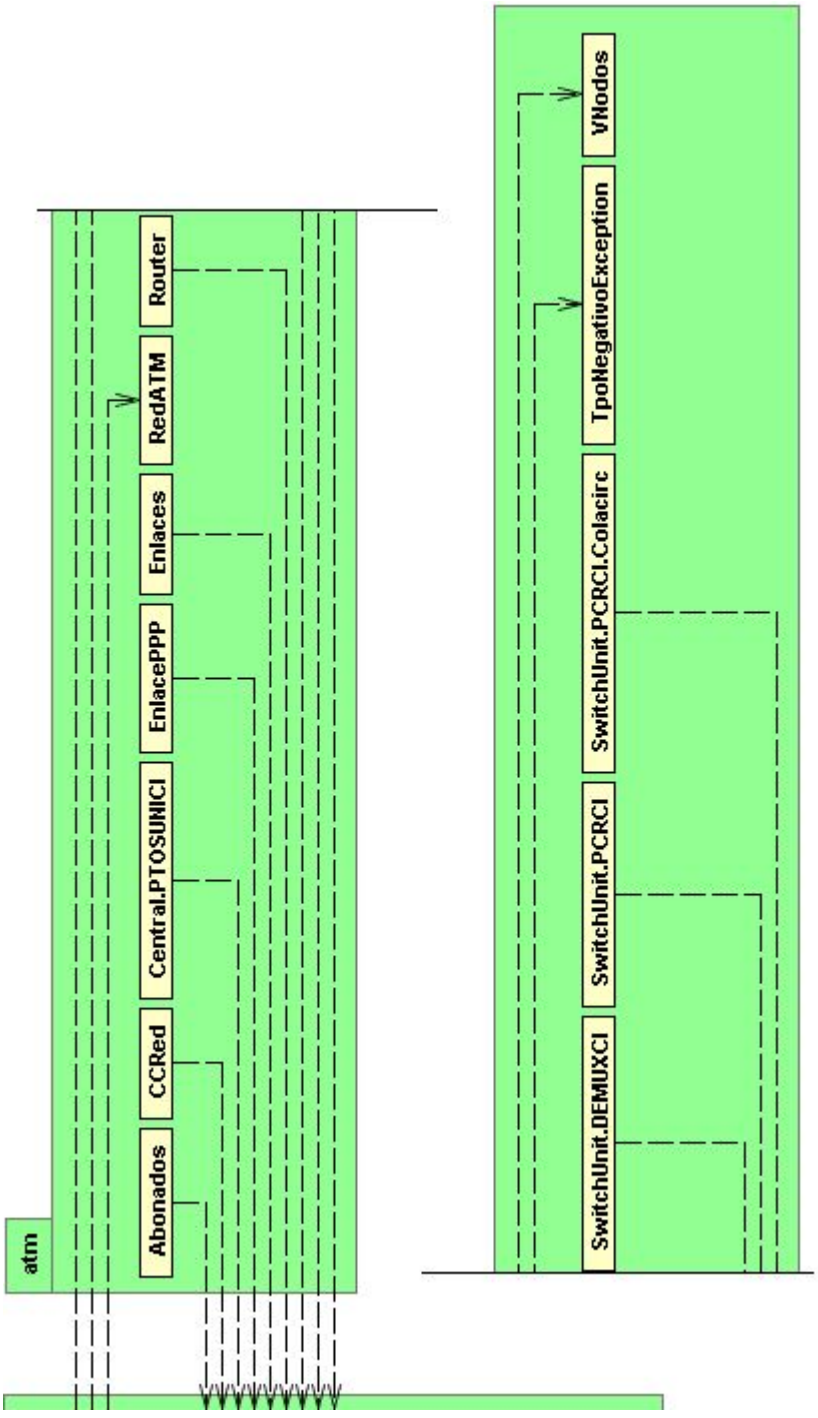



*Figura 48 Nodo (II)*

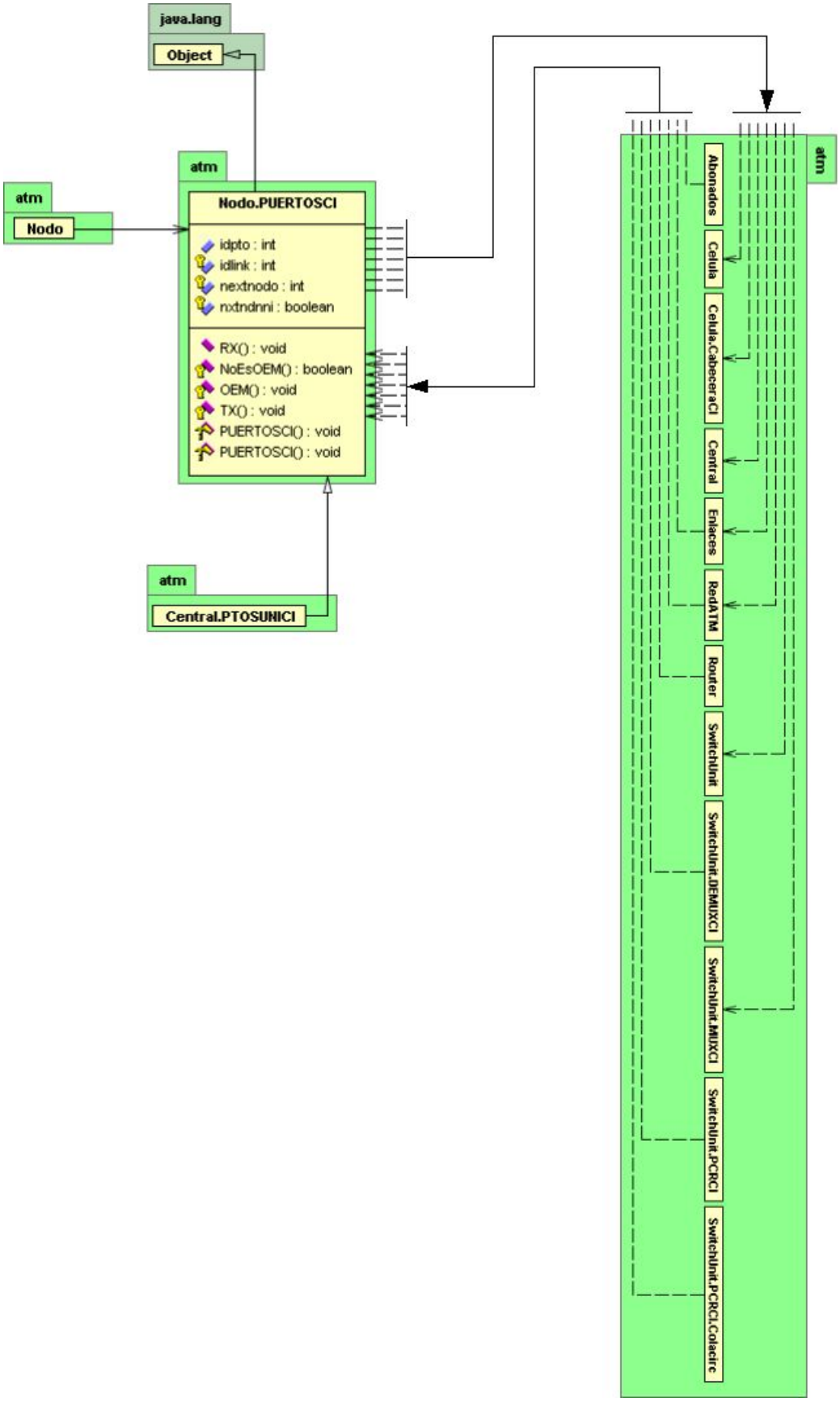



*Figura 49 Nodo.PUERTOSCI*

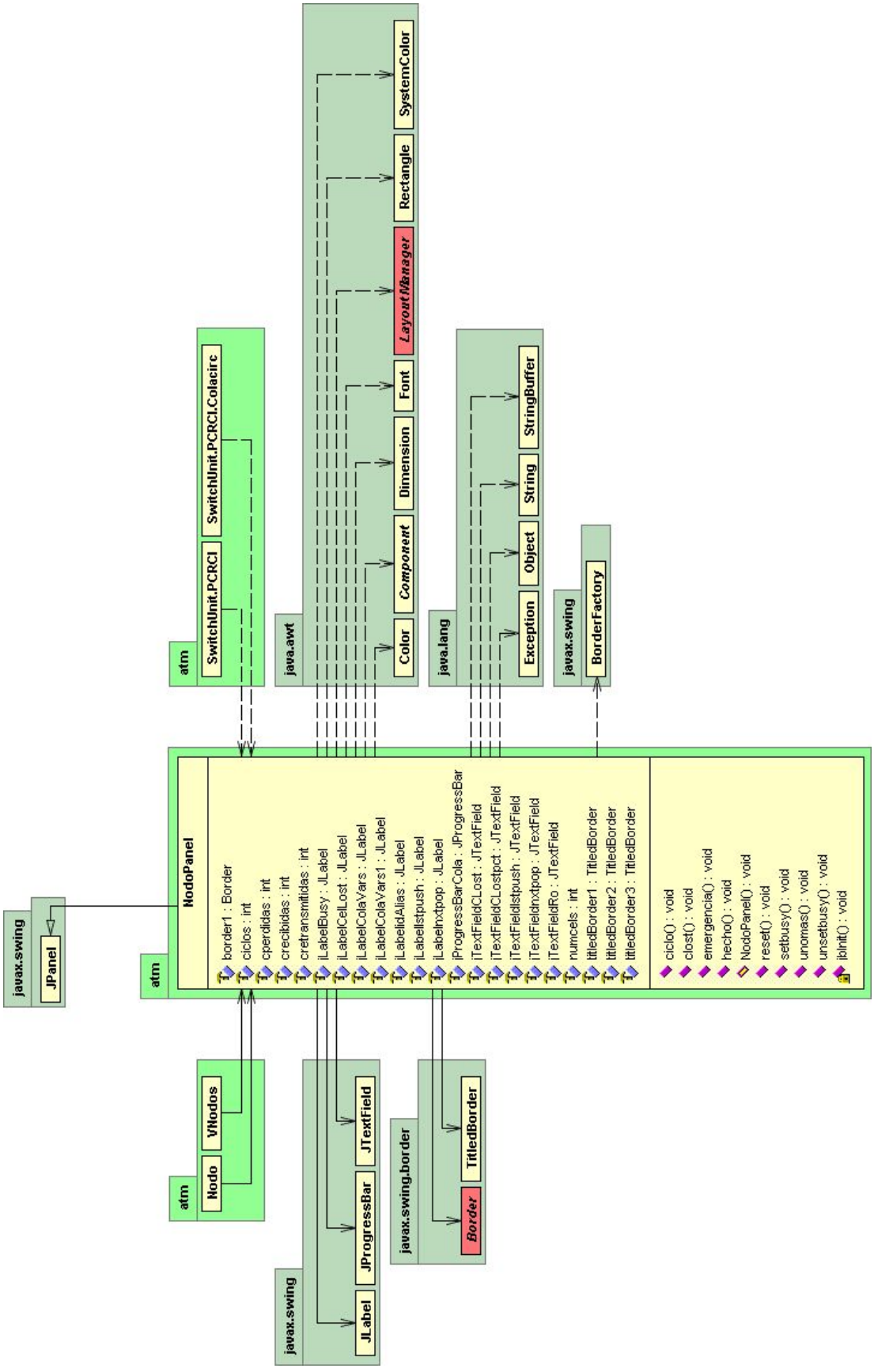



*Figura 50 NodoPanel*

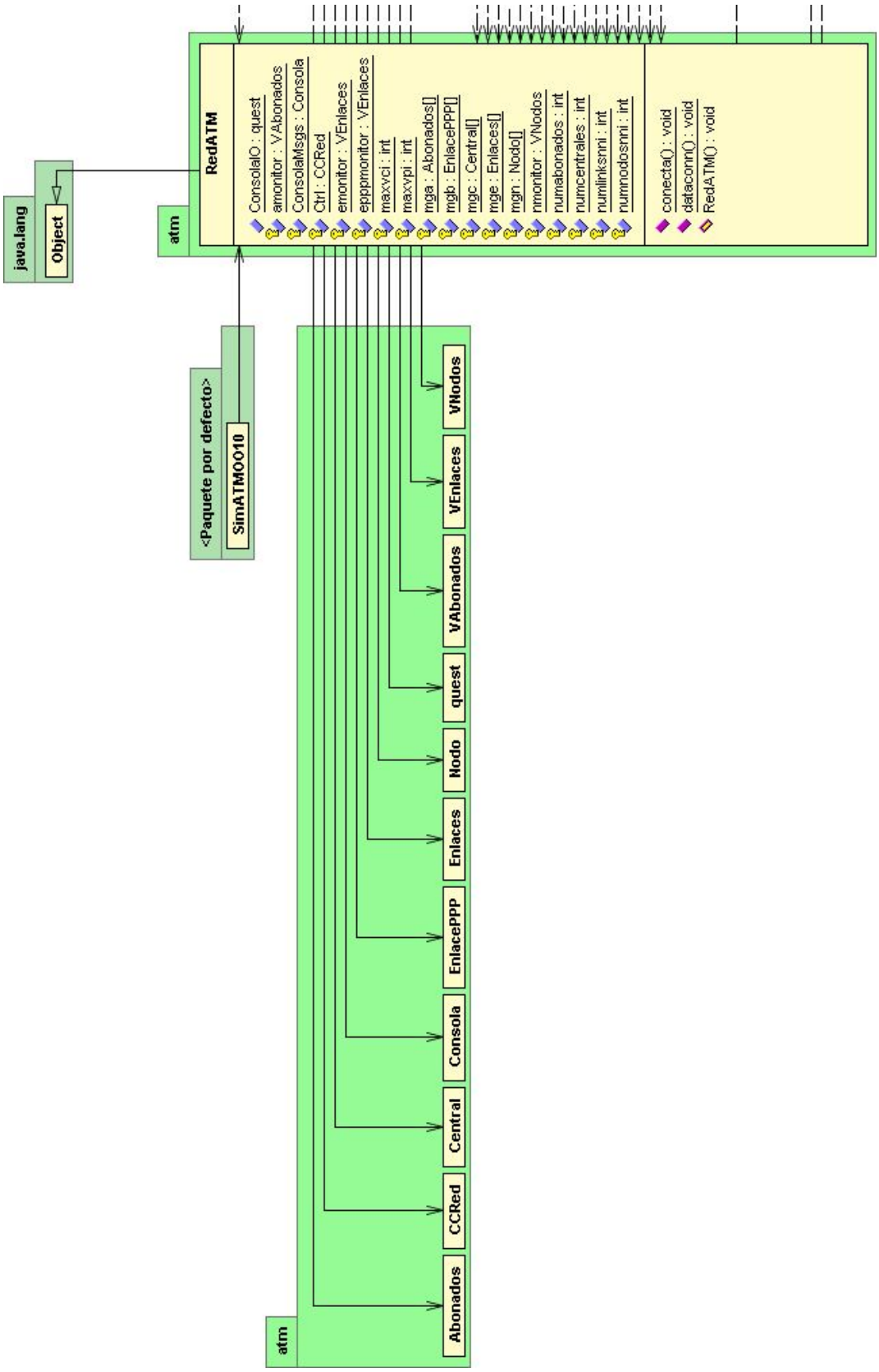



*Figura 51 RedATM (I)*

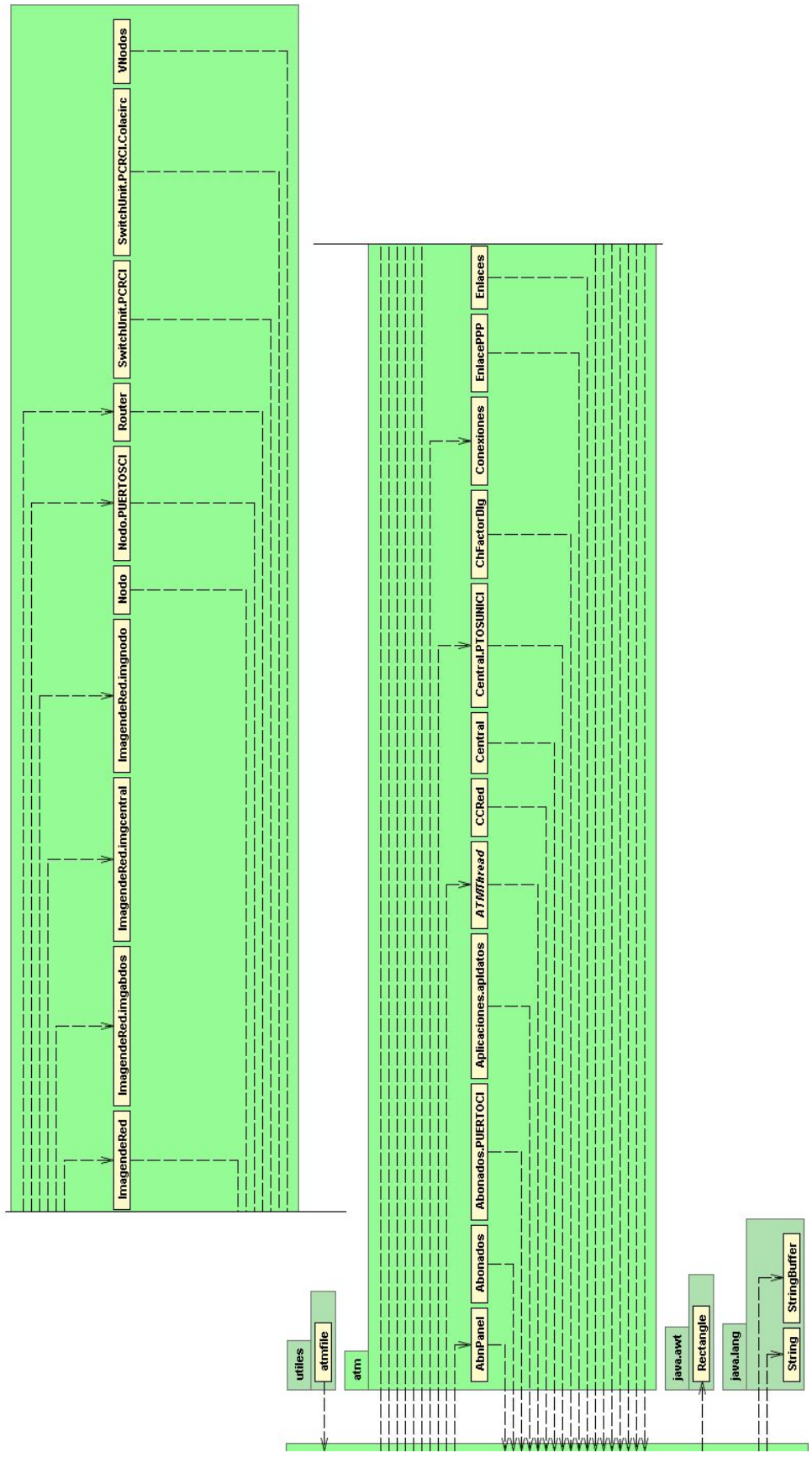



*Figura 52 RedATM (II)*

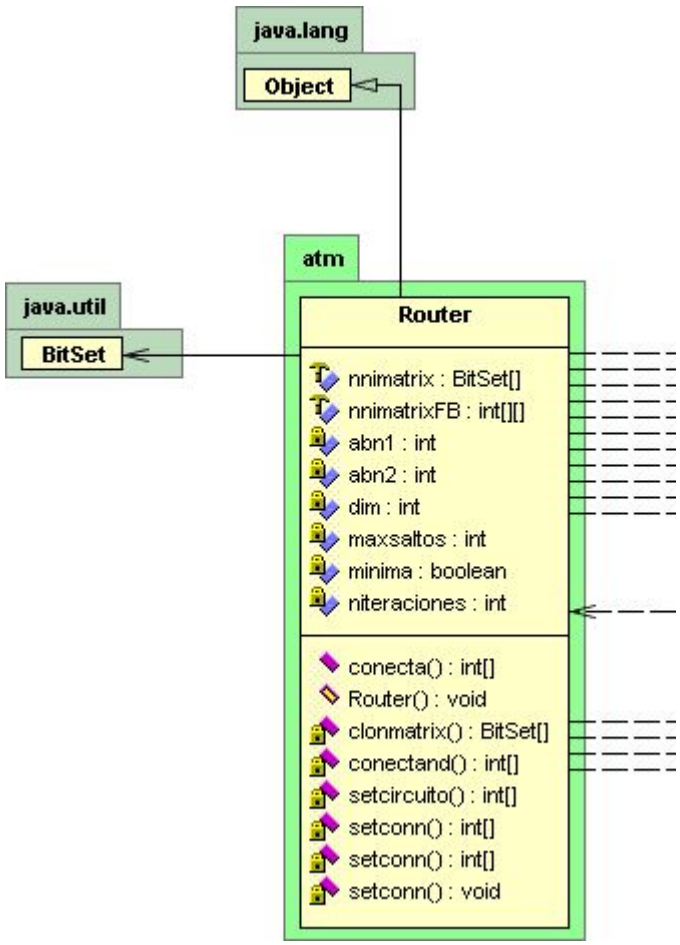



*Figura 53 Router (I)*

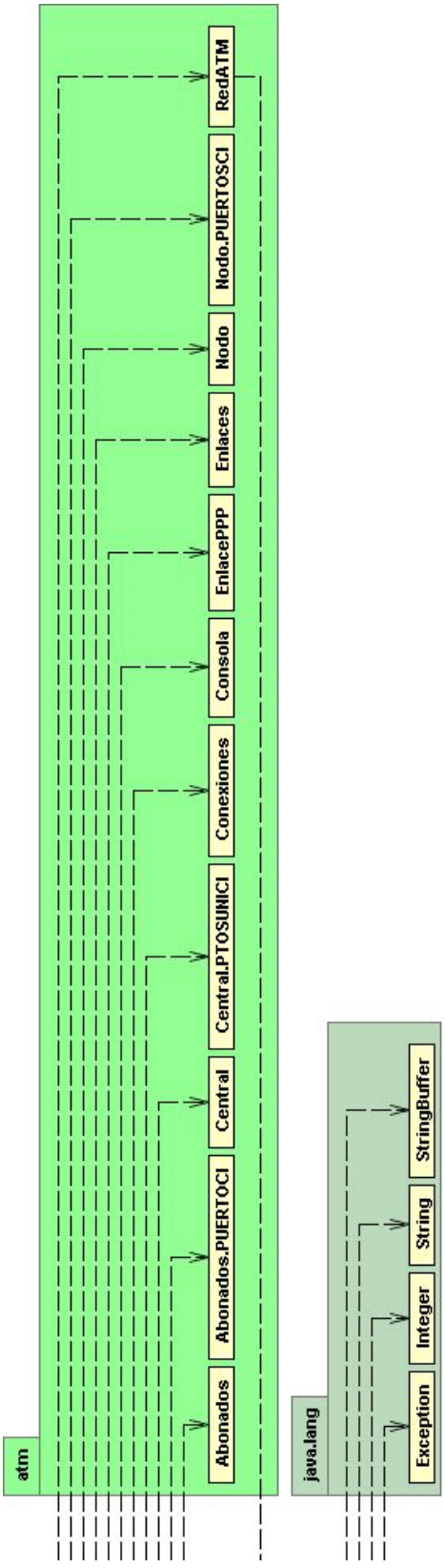



*Figura 54 Router (II)*

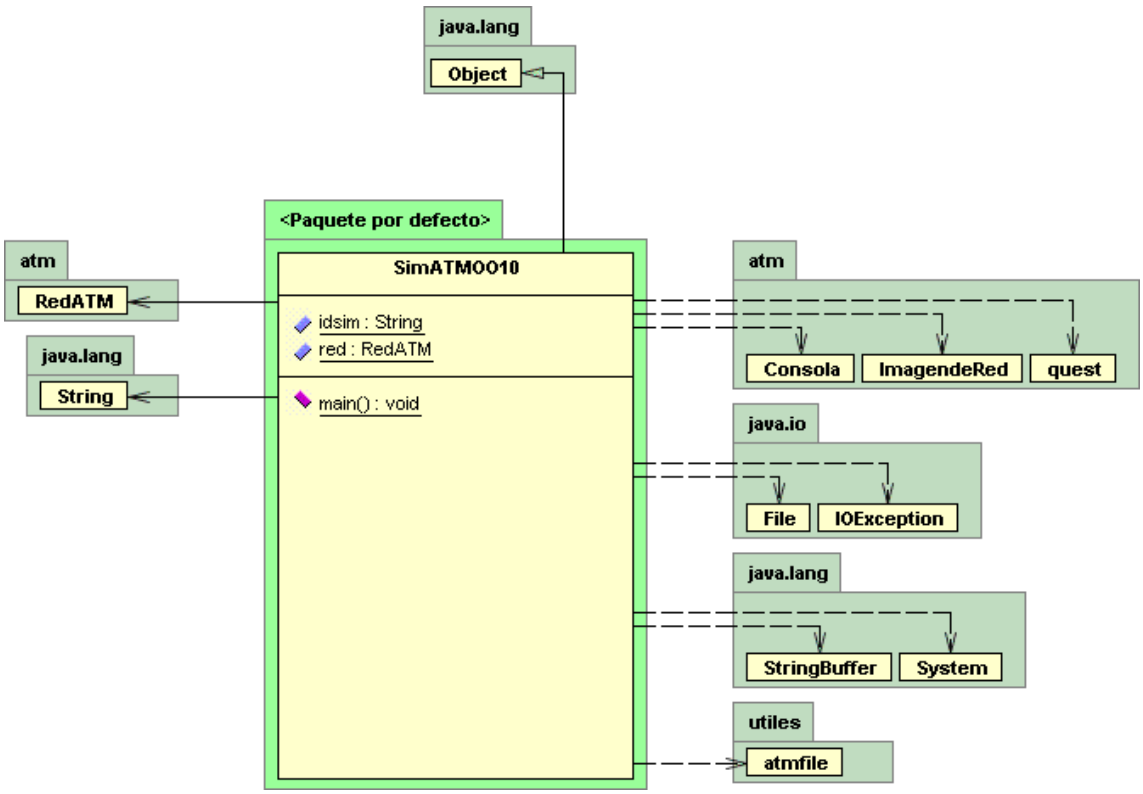



*Figura 55 SimATMOO10*

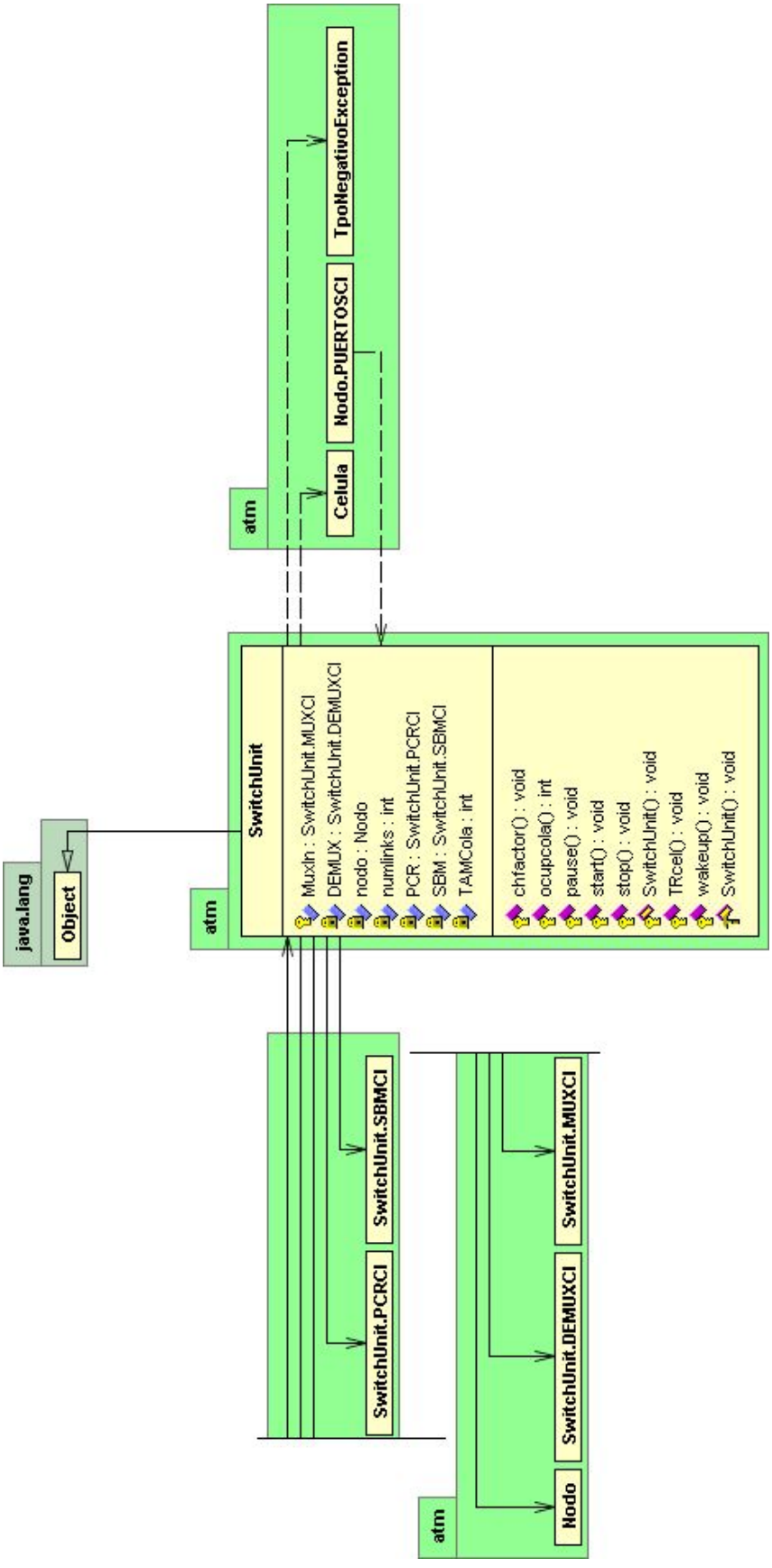



*Figura 56 SwitchUnit*

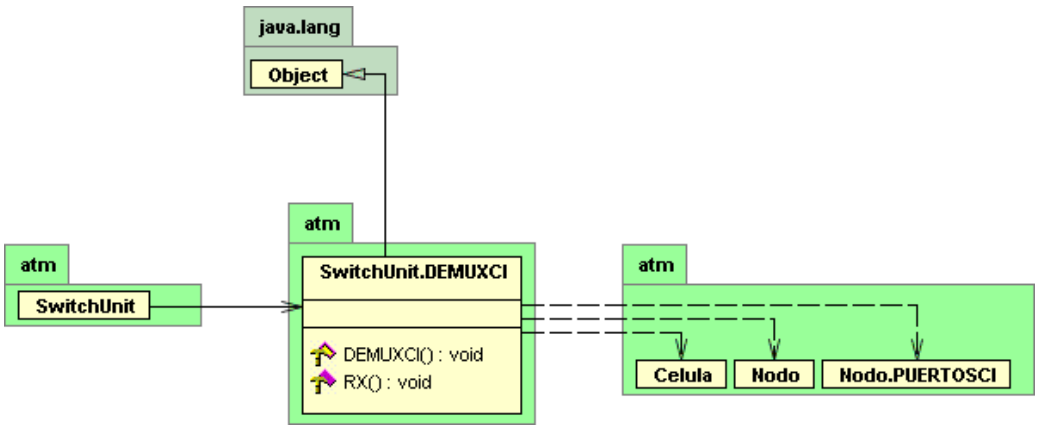



*Figura 57 SwitchUnit.DEMUXCI*

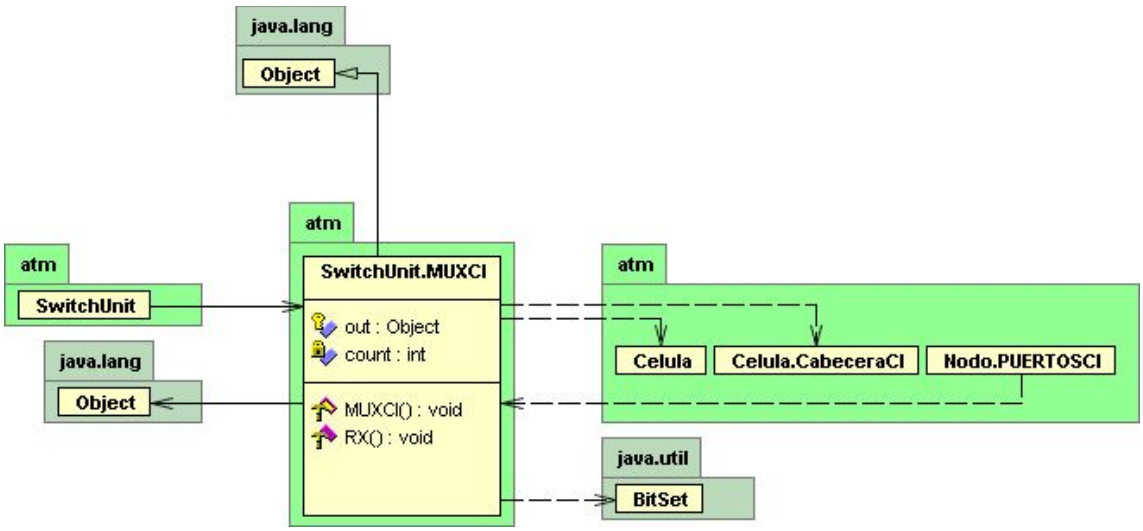



*Figura 58 SwitchUnit.MUXCI*

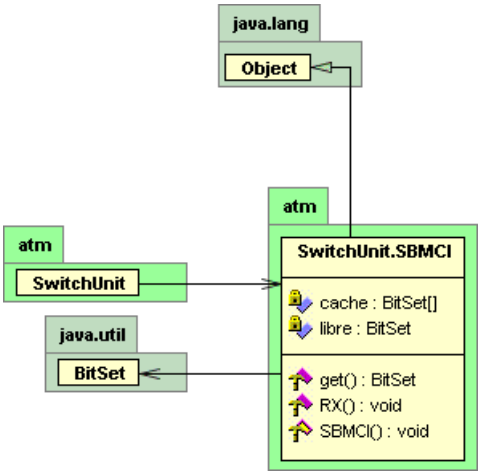



*Figura 59 SwitchUnit.SBMCI*

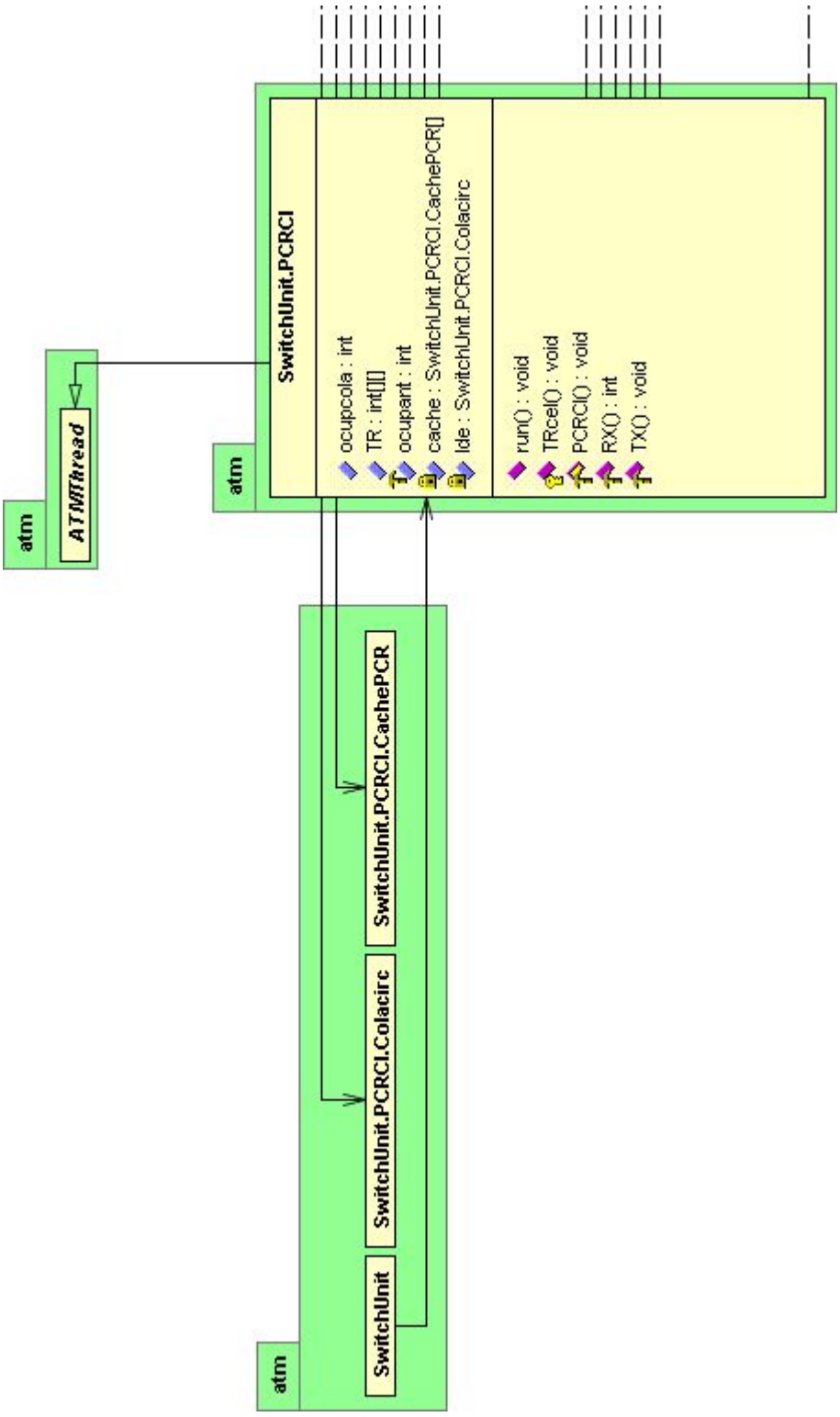



*Figura 60 SwitchUnit.PCRCI (i)*

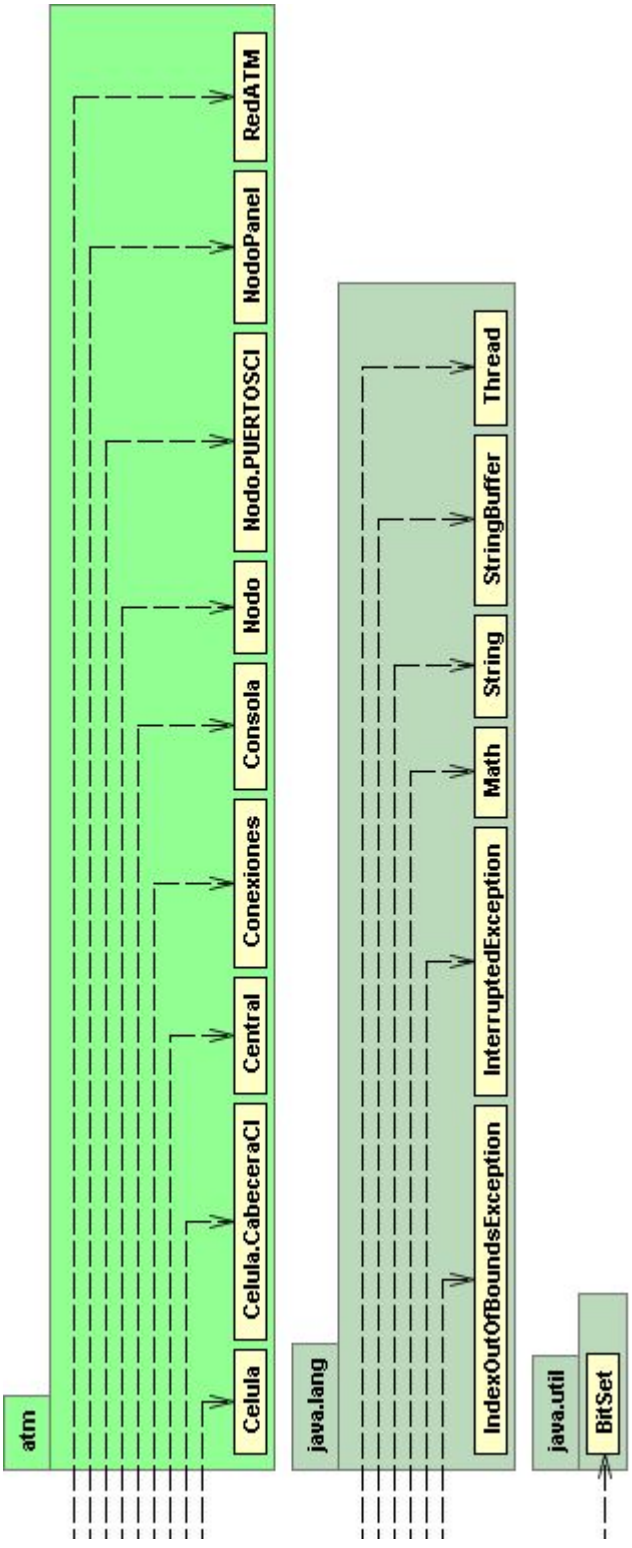



*Figura 61 SwitchUnit.PCRCI (II)*

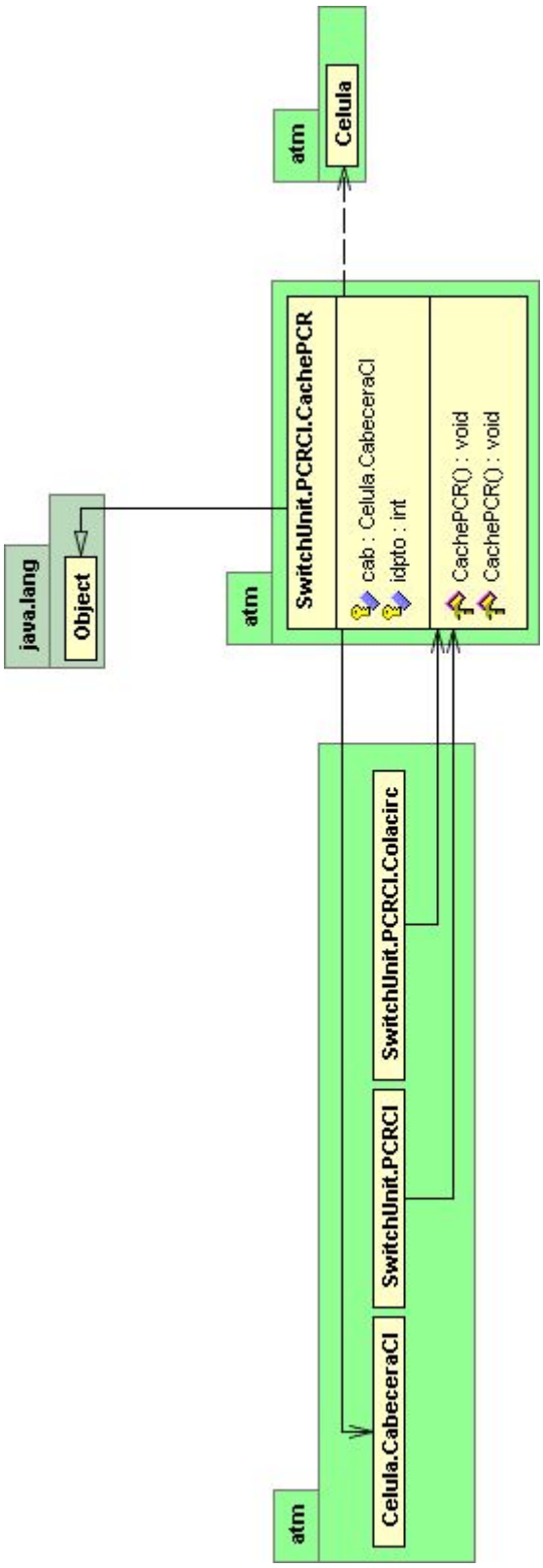



*Figura 62 SwitchUnit.PCRCl.CachePCR*

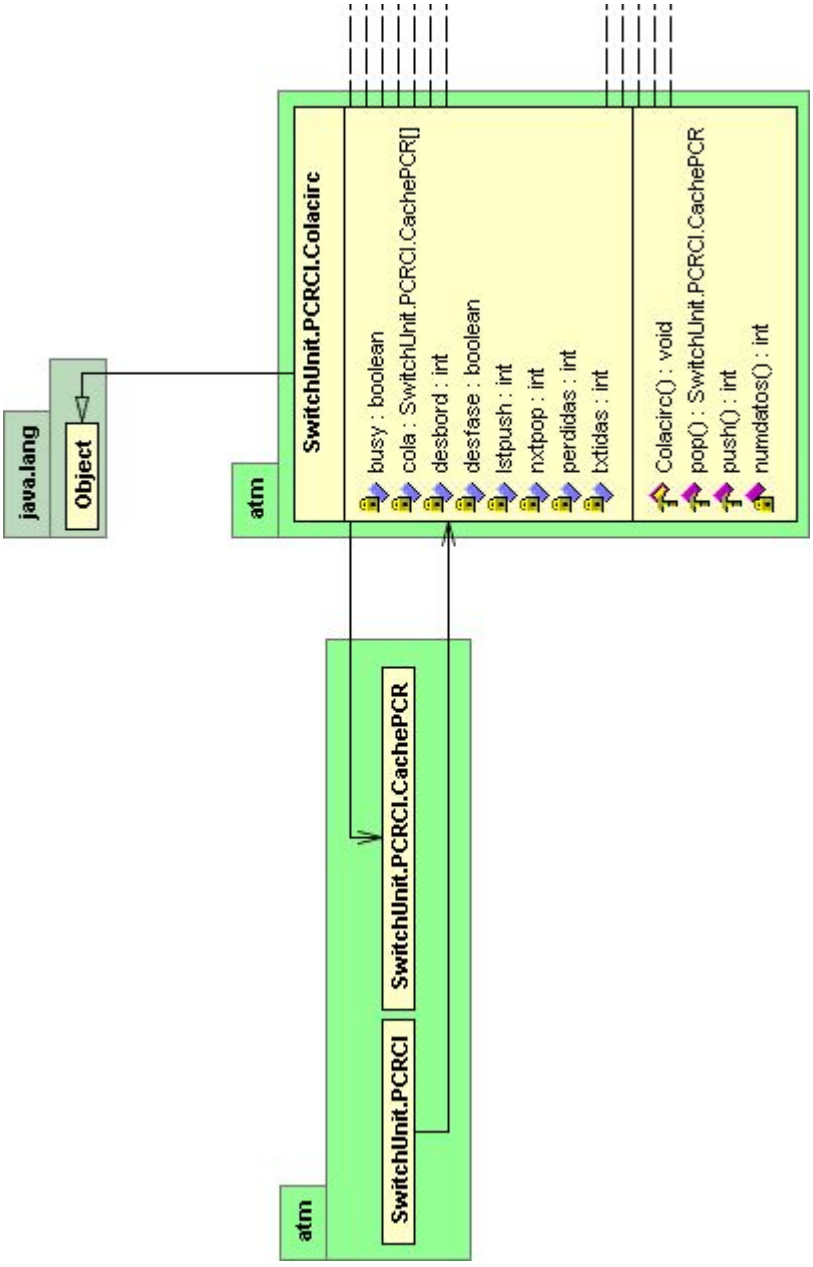



*Figura 63 SwitchUnit.PCRCI.Colacirc (I)*

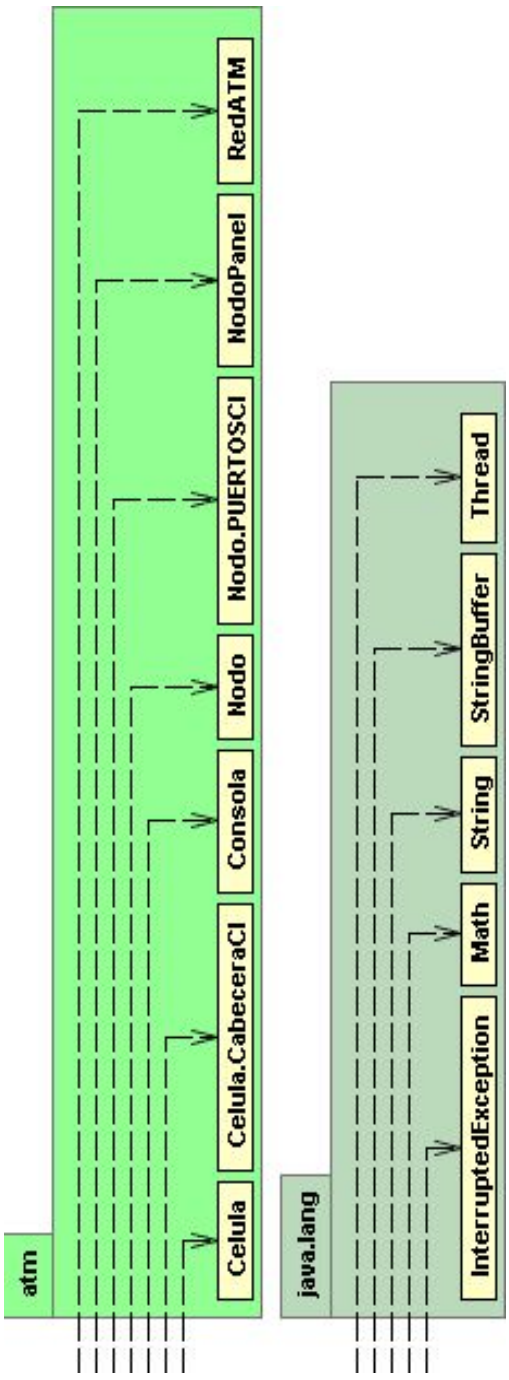



*Figura 64 SwitchUnit.PCRCl.Colacirc (II)*

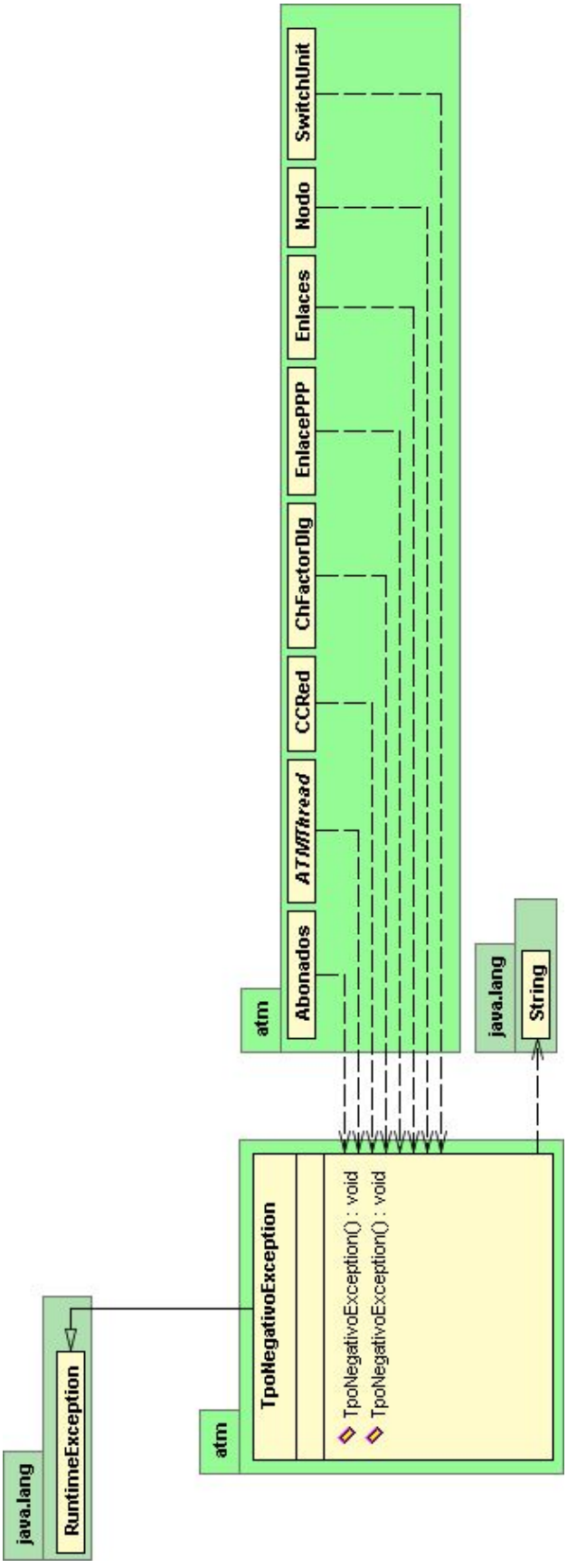



*Figura 65 TpoNegativoException*

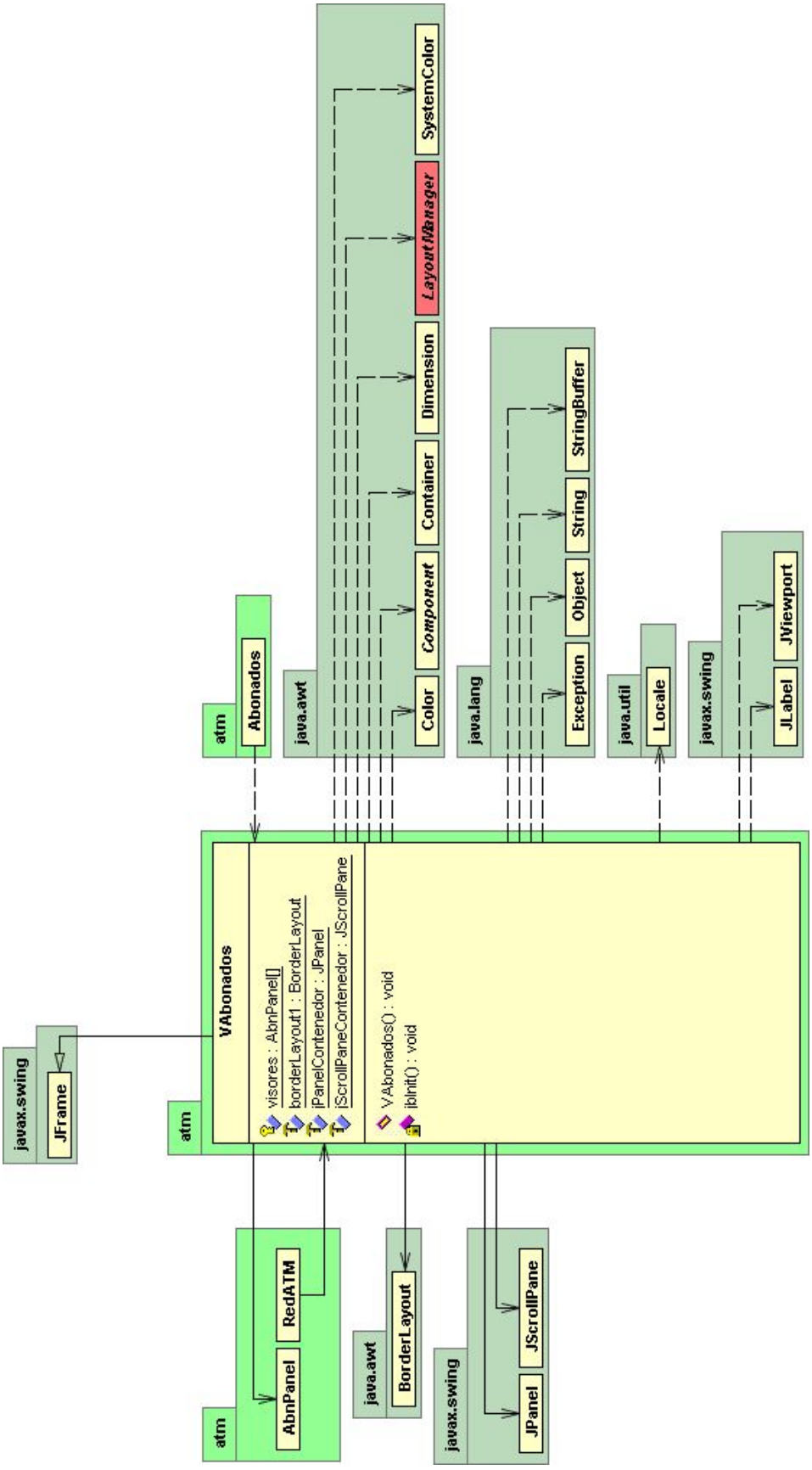



*Figura 66 VAbonados*

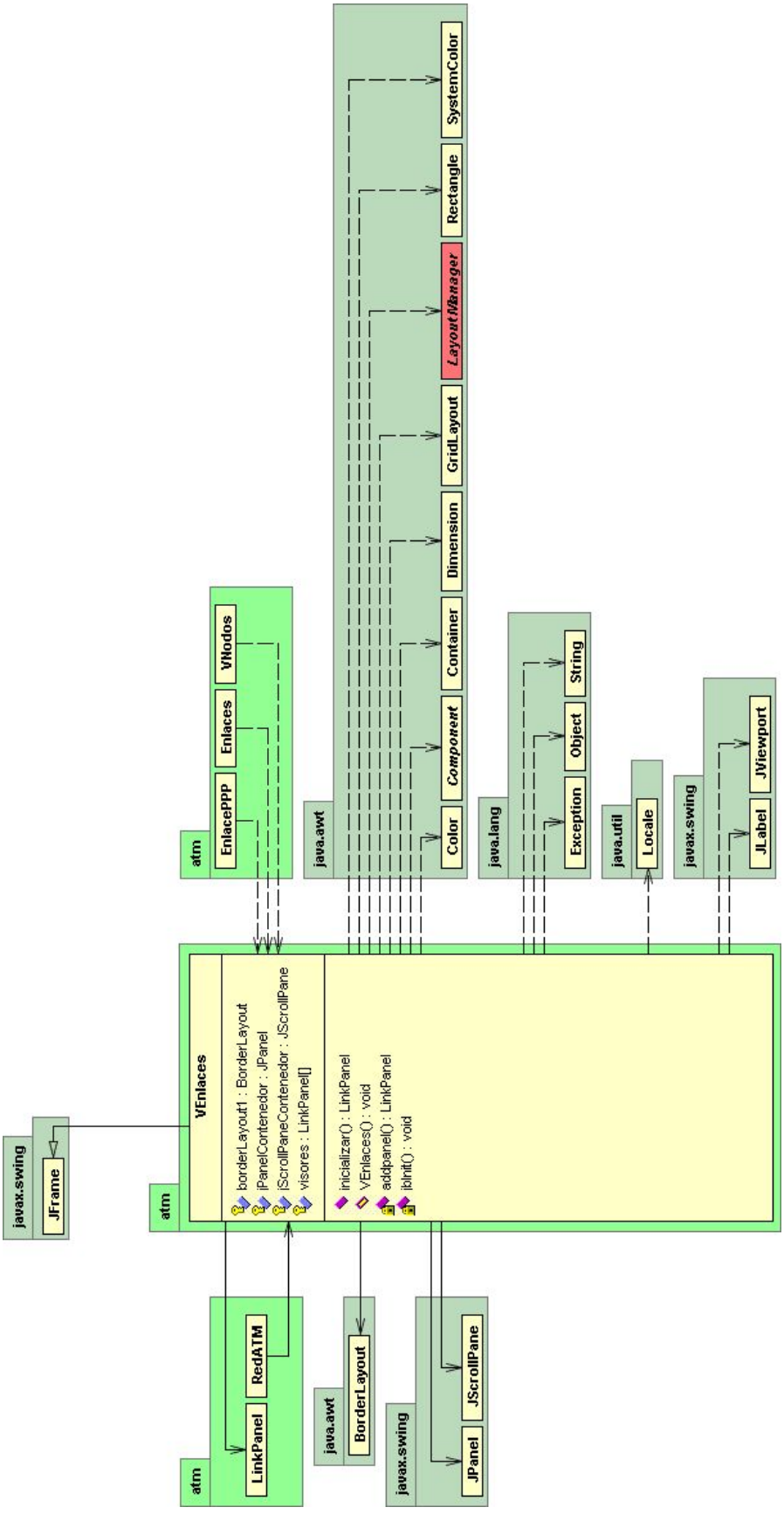



*Figura 67 VEnlaces*

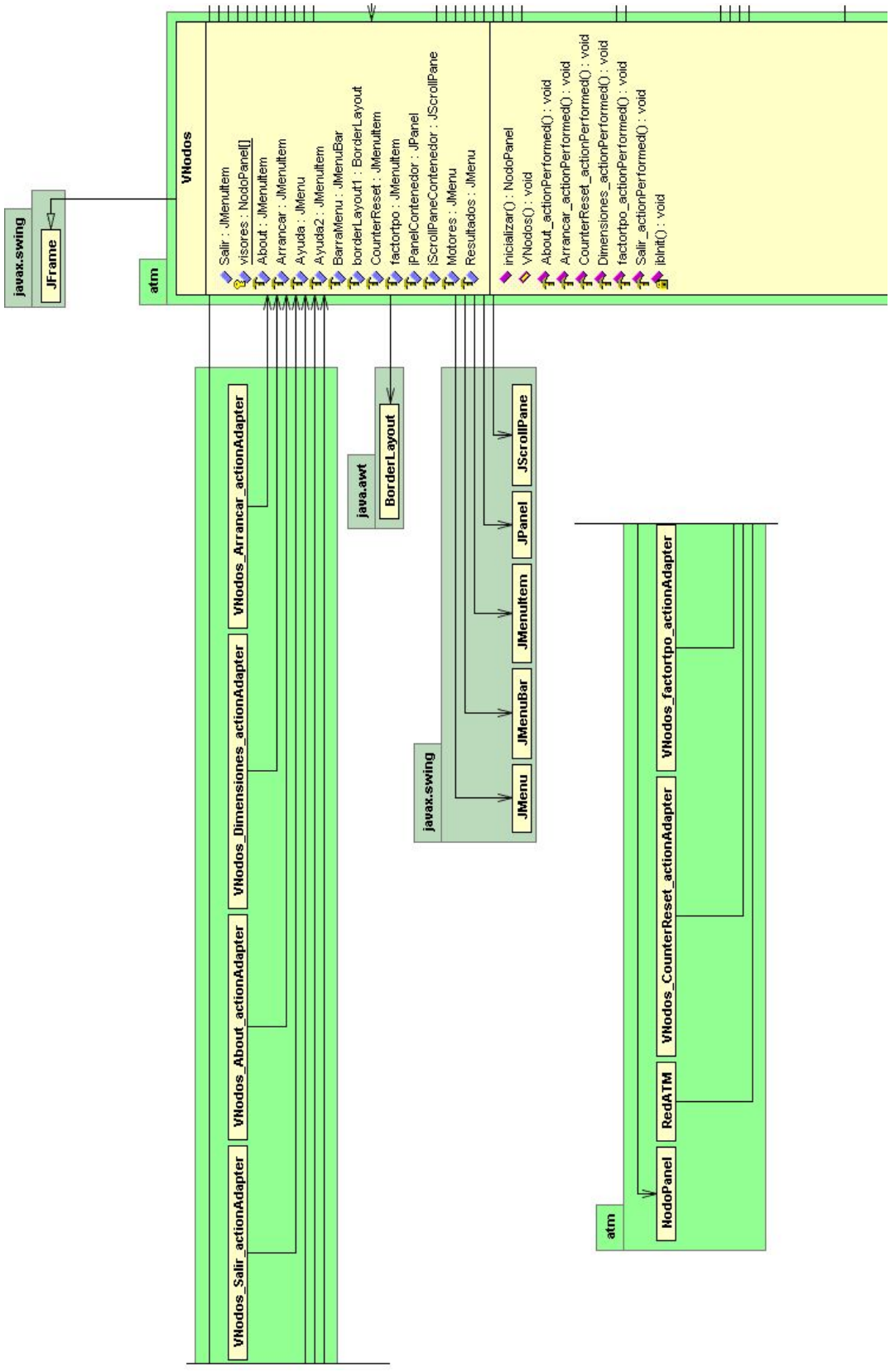



*Figura 68 VNodos (I)*

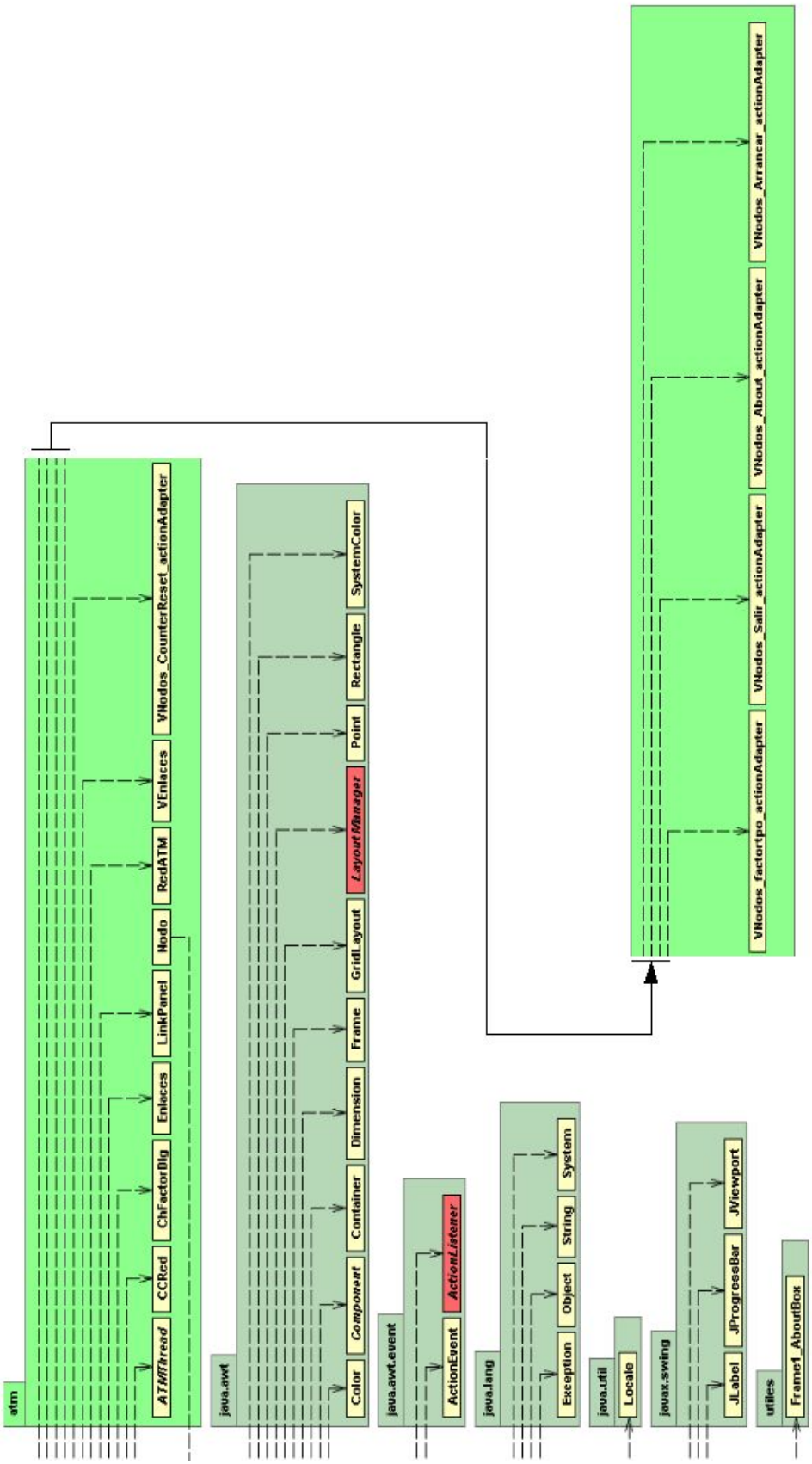



*Figura 69 VNodos (II)*

# 8 BIBLIOGRAFÍA


- *Balakrishnan, M., et al (1997): "Buffer losses vs. deadline violations for ABR traffic in an ATM switch: a computational approach". Telecommunications Systems, No. 7.*
- Bruce Eckel (2000), *Thinking in Java*, Prentice Hall.
- *Cox,* D. R. *(1984), "*Long-range dependence: A review"*,* en *Statistics: An Appraisal, David, H. A. y David, H. T (eds.). Ames, IA: Iowa State Univ. Press. Iowa, pgs. 55-74.*
- *Heo, J. W. et. al. (2000): "*Cell-level/call-level ATM switch simulator", *Telecommunication Systems,* No. 14, pgs. 291 a 309.
- *Schwartz, M. (1994): Redes de Telecomunicaciones: protocolos, modelado y análisis. Addison-Wesley Iberoamericana. Wilmington, Delaware.*
- *Sexton, M. y Reid (1997), A.: Broadband Networkin: ATM, SDH and SONET, Artech House. Norwood, MA.*
- Tsybakiov, B y Georganas, N. D. (1997): "On Self-Similar Traffic in ATM Queues: Definitions, Overflow Probability Bound, and Cell Delay Distribution"*, IEEE/ACM Transactions on networking*, vol. 5, nº 3.
- *Tsybakiov, B y Georganas, N. D. (1999): "Overflow and loss probabilities in a finite ATM buffer fed by self-similar traffic", Queueing Systems, No. 32, pgs. 233-256.*
- *UIT-T I.150 (09/1999) – Características funcionales del modo de transferencia asíncrono de la RDSI-BA.*

- *UIT-T I.361 (02/99) – Especificación de la capa de modo asíncrono de la RDSI-BA.*
- *UIT-T I.371 (03/2000) – Control de tráfico y control de congestión en RDSI-BA.*
- *UIT-T I.371.1 (11/2000) – Guaranteed frame rate ATM transfer capability*
- *UIT-T G.702 – Velocidades Binarias de la Jerarquía Digital.*
- UIT-T G.803 (03/2000) – Arquitectura de redes de transporte basadas en la jerarquía digital síncrona.
- *Youngho, L. et. al. (2001): "An ATM switch capacity allocation problem". Telecommunications Systems, No. 18:4.*
- Documentación de referencia de Java en Internet:

  *The Java Languaje Specification:*

  *http://java.sun.com/docs/books/jls/second_edition/html/j.title.doc.html*

  *The Java Virtual Machine Specification:*

  *http://java.sun.com/docs/books/vmspec/2nd-edition/html/VMSpecTOC.doc.html*