UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE

INGENIERÍA DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA:

DESARROLLO DE UNA HERRAMIENTA PARA LA VISUALIZACIÓN DE SIMULACIONES DE REDES MANET CON CACHÉ

INGENIERÍA DE TELECOMUNICACIÓN

Málaga, 2010 Lorena Belén Ríos Sepúlveda

UNIVERSIDAD DE MÁLAGA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

Titulación: Ingeniería de Telecomunicación

Reunido el tribunal examinador en el día de la fecha, constituido por:

D. _____

D. _____

D. _____

para juzgar el Proyecto Fin de Carrera titulado:

DESARROLLO DE UNA HERRAMIENTA PARA LA VISUALIZACIÓN DE SIMULACIONES DE REDES MANET CON CACHÉ

de la alumna Dª. Lorena Belén Ríos Sepúlveda

dirigido por D. Francisco Javier González Cañete

ACORDÓ POR: _____ OTORGAR LA CALIFICACIÓN DE

_____

y, para que conste, se extiende firmada por los componentes del Tribunal, la presente diligencia.

Málaga, a ____ de _____ de Año _____

El/La Presidente/a: El/La Vocal: El/La Secretario/a:

Fdo. _____ Fdo. _____ Fdo. _____

UNIVERSIDAD DE MÁLAGA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

DESARROLLO DE UNA HERRAMIENTA PARA LA VISUALIZACIÓN DE SIMULACIONES DE REDES MANET CON CACHÉ

REALIZADO POR:

Lorena Belén Ríos Sepúlveda

DIRIGIDO POR:

Francisco Javier González Cañete

DEPARTAMENTO DE: Tecnología Electrónica.

TITULACIÓN: Ingeniería de Telecomunicación

Palabras clave: Visualización de redes, MANET, Caché, Java, Java3D, NS-2, PDF.

RESUMEN:

En el presente proyecto fin de carrera se ha implementado una aplicación de escritorio que ofrece una completa herramienta para la visualización de simulaciones de redes MANET con estudios de estrategias de Caché. La aplicación toma los datos de entrada de archivos de traza generados por cualquier herramienta de simulación de redes que genere el formato creado para su descripción y los datos de movilidad de archivos con el formato de la herramienta de simulación NS-2. Por una parte, la herramienta sirve de interfaz para la representación, a modo de animación, de la actividad de la red, así como de la movilidad de los nodos. Por otra parte, el sistema desarrollado dota de la lógica necesaria para representar las estadísticas y el estado de las diferentes cachés de los nodos de la red en tiempo real. Por último, se generan informes en formato PDF con un estudio de los estadísticos de todos los nodos de la red en un determinado instante de tiempo, presentando histogramas para cada estadístico en particular.

Málaga, Junio de 2010

AGRADECIMIENTOS

A mis padres, Antonio y Ana,

a mis hermanas, Begoña y Paqui,

y a José Luis.

Agradecimientos

Este Proyecto Fin de Carrera representa la culminación de una fase importante de mi desarrollo profesional y sobre todo personal, dejando atrás muchas de las experiencias que me han hecho crecer de manera integral. Y lo más importante, forma parte de la continuación de mi proyecto de vida.

Deseo expresar mi mayor agradecimiento a mis padres, Antonio y Ana, de quienes he obtenido un apoyo incondicional para todas las decisiones de mi vida y a quienes les doy las gracias por haberme permitido estudiar lo que más deseaba, porque aunque las circunstancias económicas hayan dificultado la consecución de esta meta, me han brindado la motivación que necesitaba para lograr mis sueños. Y sobre todo, porque constituyen un referente para mí en la lucha por la vida y el trabajo duro.

A mis hermanas Begoña y Paqui, con las cuales he reñido por repartirnos el trabajo en el bar, porque son las luces de mi vida que me guían y apoyan cuando un examen sale mal o cuando una decisión ha sido la incorrecta.

A José Luis, porque siempre está ahí, ayudándome a combatir mis miedos e inseguridades, quien me ha regañado por no presentarme a un examen y quien me escucha cuando estoy insoportable. Gracias porque estás conmigo en todas mis decisiones y sabes sacar lo mejor de mí.

Gracias a todos vosotros, porque hemos conseguido finalizar esta carrera por realizar un trabajo en equipo.

A todos mis amigos y compañeros por compartir momentos tan especiales que han pasado a lo largo de mi vida.

Y a mi tutor Francis, por su esfuerzo, dedicación y colaboración en el desarrollo de este proyecto, quien ha sido capaz de transmitirme gran entusiasmo por el resultado obtenido.

Índice de Contenidos

Índice de Figuras

Índice de Tablas

Relación de Acrónimos

ANSI	*American National Standards Institute*
AODV	*Ad Hoc Distance Vector*
API	*Application Program Interface*
AWT	*Abstract Window Toolkit*
CNA	*CacheNode Application*
DS	*Data Server*
DSDV	*Destination-Sequenced Distance Vector*
DSR	*Dynamic Source Routing*
EJB	*Enterprise JavaBeans*
FIFO	*First In First Out*
GB	*GigaBytes*
GHz	*GigaHercios*
GCC	*GPL Compiler Collection*
GLUT	*openGL Utility Toolkit*
GPL	GNU *General Public License*
GUI	*Graphical User Interface*
GW	*GateWay*
HTML	*HyperText Markup Language*
IDE	*Integrated Development Environment*
IEEE	*Institute of Electrical and Electronics Engineers*
IETF	*Internet Engineering Task Force*
iNSpect	*interactive NS-2 protocol and environment confirmation tool*

IP	*Internet Protocol*
IR	Ingeniería de Requisitos
IRTF	*Internet Research Task Force*
J2EE	*Java 2 Enterprise Edition*
J2EE	*Java 2 Platform, Enterprise Edition*
J2ME	*Java 2 Micro Edition*
J2SE	*Java 2 platform, Standard Edition*
JAR	*Java ARchive*
JDK	*Java Development Kit*
JNI	*Java Native Interface*
JOGL	*Java OpenGL*
JPEG	*Joint Photographic Experts Group*
JRE	*Java Runtime Environment*
JSR	*Java Specification Request*
JVM	*Java Virtual Machine*
LAN	*Local Area Network*
LFU	*Last Frequently Used*
LRU	*Least Recently Used*
MANET	*Mobile Ad Hoc NETwork*
MB	*MegaBytes*
MVC	Modelo Vista Controlador
NAM	*Network Animator*
NS-2	*Network Simulator 2*
OSI	*Open System Interconnection*

PDF	*Portable Document Format*
PNG	*Portable Networks Graphics*
RAM	*Random Access Memory*
RERR	*Route ERRor*
RF	Requisitos Funcionales
RNF	Requisitos No Funcionales
RP	*Routing Protocol*
RREP	*Route REPly*
RREQ	*Route REQuest*
RTF	*Rich Text Format*
RWN	*Reconfigurable Wireless Networks*
SDK	*Software Development Kit*
SE	*Standard Edition*
SXO	*Size X Order*
TCP	*Transport Control Protocol*
TDS_D	*Time and Distance Sensitive – mainly Distance*
TORA	*Temporary Ordered Routing Algorithm*
TTL	Time To Live
UDP	*User Datagram Protocol*
UML	*Unified Modeling Language*
WLAN	*Wireless LAN*
WRP	*Wireless Routing Protocol*
WWW	*World Wide Web*
XML	*Extensible Markup Language*

ZRP *Zone Routing Protocol*

CAPÍTULO 1: Introducción

1.1. REDES *AD HOC*

La reciente proliferación de los dispositivos portátiles y la difusión de la tecnología sin cables han permitido prosperar a las denominadas redes inalámbricas *ad hoc*. Últimamente el interés por la comunicación en este tipo de redes ha aumentado considerablemente. La atención por parte de la comunidad científica es evidente, por las múltiples publicaciones ya presentes al cuidado de las instituciones de numerosos grupos de trabajo, como el IETF (*Internet Engineering Task Force*) y el IRTF (*Internet Research Task Force*), en torno a los cuales se centra la actividad de investigación. IETF creó el grupo de trabajo para redes *ad hoc* inalámbricas con el fin de estandarizar sus protocolos y sus especificaciones funcionales, así como para responder a las nuevas funcionalidades y revisiones que vayan surgiendo. Con el desarrollo de las comunicaciones móviles y las redes inalámbricas en general, se han propiciado una serie de escenarios en los que las limitaciones de las redes tradicionales pueden incluso impedir la explotación de un servicio dado, haciendo que cobre una gran importancia la investigación en este nuevo tipo de redes.

En el ámbito de las redes, se pueden distinguir dos tendencias diferenciadas: las redes con infraestructura y las redes sin infraestructura, las redes *ad hoc*. El primer grupo de redes necesitan de un conjunto de puntos de acceso para su operación, como las redes celulares existentes, donde se establecen unas áreas de cobertura que obligan a que el terminal móvil se encuentre dentro de alguna de dichas áreas. Además, la movilidad de los terminales quedará limitada por los puntos de acceso y la infraestructura en sí, y surge la necesidad de producir un traspaso suave entre diferentes estaciones, con un retardo que sea imperceptible y sin pérdida de datos. Por otro lado, en el segundo tipo de redes no se necesita ningún tipo de infraestructura fija, lo cual es de gran ventaja cuando las características de la duración y/o eventualidad de la comunicación no aconsejan la instalación de ningún tipo de infraestructura. Ejemplos de escenarios típicos de este segundo tipo de redes podrían ser lugares sin infraestructura preexistente (pensemos en expediciones en lugares poco accesibles, o en campos de batalla), donde dicha infraestructura ha fallado (por ejemplo, en situaciones de rescate o lugares con infraestructuras dañadas donde es necesario un despliegue rápido) o simplemente no son los adecuados para las necesidades del momento (como puede ser la interconexión de sensores de ultra-bajo consumo). O se puede pensar también en conferencias donde la

gente que participa puede formar una red temporal que permita la comunicación sin requerir los servicios de ninguna red existente.

Las redes *ad hoc* se basan en que se establece una red entre todos los terminales que se quieran comunicar, lo que implica que cada terminal o nodo debe estar dispuesto a retransmitir paquetes de datos para asegurar que los paquetes van desde el origen hacia su destino. Es decir, para la comunicación entre dos nodos no es necesario que haya comunicación directa entre ellos, esto es, se encuentren en sus zonas de cobertura. Basta que un nodo, en lugar de enviar directamente los paquetes al nodo destinatario, pueda encaminar los paquetes hacia otro nodo con el cual está en visibilidad radio y este último se encargue de retransmitir los paquetes hacia el destino. Se puede tener de esta manera caminos con saltos múltiples, o denominados *multihop*. Esta forma de comunicación está limitada por los rangos de transmisión de cada nodo particular, que normalmente es menor que los rangos de terminales en redes con infraestructura, sobre todo comparado con las redes celulares. Esto no significa que las redes celulares sean más eficientes que las redes *ad hoc*. De hecho, en el caso de que los nodos se encuentren a grandes distancias, se podrán comunicar a través de caminos de saltos múltiples, presentando además algunas otras ventajas, como que el establecimiento se realiza bajo demanda y que son tolerantes a fallos, puesto que la información podrá establecerse por otros caminos.

1.1.1. CARACTERÍSTICAS DE REDES MANET

Una red móvil *ad hoc* (MANET, *Mobile Ad hoc NETwork*) es una colección de dispositivos o nodos móviles autónomos que son capaces de comunicarse entre sí por radioenlaces inalámbricos, donde no existe una infraestructura de red fija. Como los terminales o nodos están retransmitiendo paquetes hacia los nodos de su rango de cobertura, surge la necesidad de algún tipo de protocolo de encaminamiento para tomar las decisiones acerca del enrutado. La administración de este tipo de redes se realiza de forma descentralizada, y crean escenarios en los cuales todos los nodos de la red participan en la toma de decisiones, ya sea respecto a los algoritmos de enrutamiento a utilizar como para la realización de las funciones propias del mantenimiento de la red. Para conseguir la autonomía que las caracteriza, los nodos colaboran retransmitiendo y enrutando los paquetes hacia otros nodos, de forma que puedan alcanzar su nodo destino, constituyendo por tanto redes de nodos con cierto grado de inteligencia. Es decir, en este tipo de redes, los nodos son, de forma simultánea, *hosts* y *routers*. Entre las características principales de este tipo de redes destacan:

- Topología dinámica, puesto que los nodos siguen patrones de movilidad que son independientes entre sí, lo que implica la variación continua de los nodos vecinos, es decir, van apareciendo nodos y desapareciendo de la zona de cobertura de cada nodo y debe establecer por tanto enlaces nuevos. Es más, si un nodo deja de formar parte de una ruta, la red tendrá que descubrir rutas nuevas. Un protocolo de encaminamiento para redes *ad hoc* inalámbricas debe gestionar la movilidad de sus nodos de manera eficiente y efectiva.
- Enlaces de ancho de banda limitado y capacidad variable, donde tanto el acceso al medio como la transmisión por el canal inalámbrico deben plantear mensajes de control mínimos. Además, la transmisión inalámbrica se caracteriza por anchos de banda reducidos y mayor probabilidad de error que en las transmisiones cableadas, puesto que en entornos inalámbricos la banda de radio está siempre compartida y limitada y por ello las tasas de datos que pueden ofrecer serán siempre menores. Esto requiere que los protocolos de encaminamiento deban usar óptimamente el ancho de banda reduciendo, en la medida de lo posible, los bits de las cabeceras y los paquetes de control. Estas restricciones en el ancho de banda también imponen restricciones en el mantenimiento de la información referente a la topología de la red, ya que el intercambio de información entre este tipo de nodos implica cierto desperdicio de ancho de banda útil para datos. Este es otro de los motivos por los cuales no son válidos los protocolos de encaminamiento tradicionales de las redes cableadas, ya que éstos necesitan información completa de la topología de la red, además que presuponen que no es cambiante, calculando así mediante algoritmos las rutas óptimas. El canal inalámbrico además presenta características variables con el tiempo respecto a la probabilidad de error de bit y su propia capacidad, por lo que el protocolo de encaminamiento deberá buscar las rutas menos congestionadas para disminuir la probabilidad de colisiones en el medio compartido.
- Limitaciones de energía y capacidad de procesamiento de los nodos, puesto que, debido a la naturaleza móvil de este tipo de redes, los dispositivos que la forman han de ser portátiles y ligeros, restringiendo el hardware y el software que los componen. Las baterías son limitadas, por lo que las transmisiones son de baja potencia y de alcances reducidos.

Además, hay que destacar la importancia del uso de Internet, lo que hace pensar en la obligada compatibilidad entre ésta y cualquier nueva estructura de redes (en nuestro caso particular, las redes *ad hoc*). De hecho, por la naturaleza de los terminales que se pueden

encontrar en una red *ad hoc*, como ordenadores portátiles y teléfonos móviles con cada vez más lógica, la compatibilidad con Internet se estima necesaria. En algunos de estos nuevos escenarios, los usuarios demandan acceso a redes externas, especialmente a Internet, como se observa en la Figura 1.1.

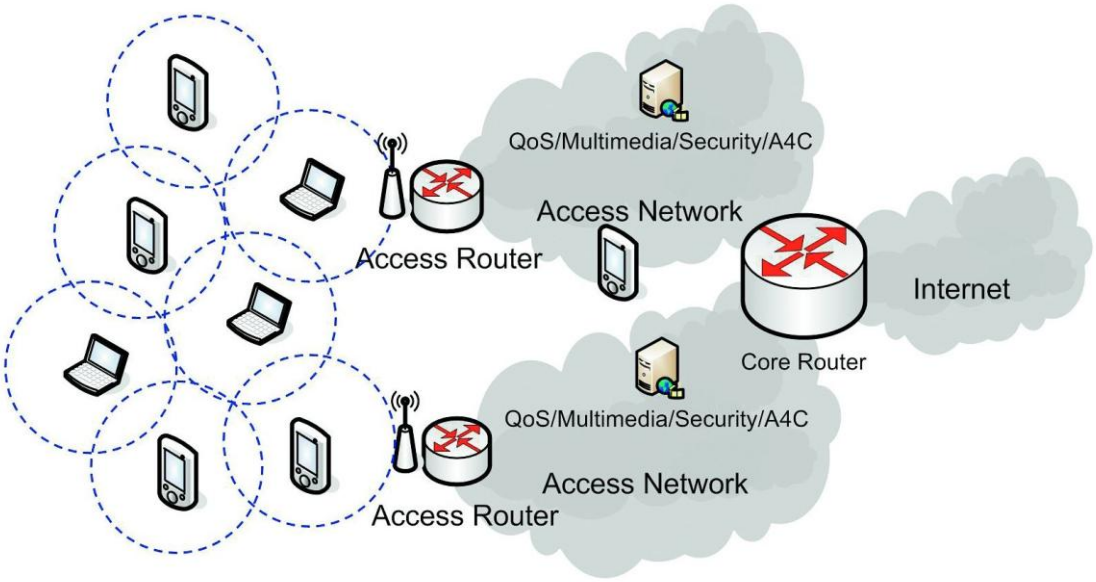


Figura 1.1. Escenario típico de una red *ad hoc* con conexión hacia Internet

1.1.2. PROTOCOLOS DE ENCAMINAMIENTO EN REDES MANET

Como se ha comentado anteriormente, los protocolos de enrutamiento utilizados en las redes cableadas no son aptos para ser utilizados en las MANET. La movilidad de los nodos que conforman la red y la aparición y desaparición de los mismos de las zonas de cobertura modifican los posibles enlaces que se puedan establecer entre ellos. Los protocolos clásicos de enrutamiento no están preparados para adaptarse a situaciones tan variantes. Existen varias propuestas de enrutamiento diseñadas específicamente para redes MANET, pero todavía no hay un consenso generalizado de cómo realizar esta tarea.

Los algoritmos de encaminamiento usados en las redes Ad-Hoc se pueden clasificar en dos grupos atendiendo al mecanismo de actualización de la información de encaminamiento:

- Basados en tablas de encaminamiento o **proactivos**. Estos algoritmos tratan de mantener continuamente actualizada la información necesaria acerca de la topología de la red para poder encaminar los paquetes de forma adecuada. Cada nodo mantiene una o más tablas con los datos para encaminar hacia cualquier otro nodo de la red. Una parte o bien todos los nodos de la red transmiten

periódicamente, lo que permite que el resto de nodos conozcan los enlaces que, en ese momento, existen entre todos ellos. Por tanto, los cambios en la topología de la red desencadenan una serie de mensajes de control para la reestructuración y/o creación de las nuevas rutas, manteniendo las tablas actualizadas. Dentro de esta categoría se encuentran: DSDV (*The Destination-Sequenced Distance-Vector Routing Protocol,* [1][2]), OLSR (*Optimized Link State Routing Protocol,* [3]), CGSR (*Cluster-head Gateway Switch Routing Protocol,* [1]) y WRP (*Wireless Routing Protocol,* [1][4]). Los protocolos anteriores difieren en el número de tablas utilizadas y en la política de envío de paquetes para mantener las tablas actualizadas. Este tipo de protocolos resultan eficientes en entornos de movilidad media de los nodos, puesto que la información de la topología de la red no cambia drásticamente, y la información recogida en los mensajes es suficiente para obtener las rutas. En cambio, en entornos de muy baja movilidad, el envío periódico repercute en el consumo de potencia, produciendo un mayor consumo energético de las baterías.

- Basados en encaminamiento bajo demanda o **reactivos**. En contraste con los algoritmos basados en tablas, las rutas son creadas sólo cuando se requiere enviar un paquete de datos. Cuando un nodo requiere una ruta hacia un destino concreto se inicia un proceso de descubrimiento de ruta. Este proceso termina cuando se encuentra un camino hacia el destino o cuando se examinan todas las alternativas y ninguna lleva al destino final. Cuando la ruta es descubierta, es necesario mantenerla (mantenimiento de ruta) hasta que el destino se vuelva inalcanzable o la ruta deje de ser necesaria. Esto tiene el inconveniente de introducir un mayor retardo a la hora de enviar los primeros paquetes, pero se presenta la ventaja de sobrecargar menos la red con paquetes de control. La movilidad que presentan los nodos y por tanto, la variabilidad de las rutas, entre otros factores, determinarán si esta aproximación es mejor a la anterior o no. Algunos ejemplos de este tipo de protocolos son: AODV *(Ad Hoc On-Demand Distance Vector Routing Protocol,* [1][5]), DSR *(Dynamic Source Routing Protocol,* [1][6]), TORA *(Temporary Ordered Routing Algorithm,* [1][7]), ABR *(Associative-Based Routing Protocol,* [1]) y SSR *(Signal Stability Routing Protocol,* [1]).
- Basados en una aproximación **híbrida**. Intentan combinar las mejores características de los anteriores. Así, se define una zona de cobertura de nodo, manteniendo, como en los protocolos proactivos, la información de los nodos que se encuentren a menos de un cierto número de saltos. Dicha cobertura se irá

obteniendo gracias a la recepción periódica de mensajes, de forma que el nodo podrá conocer la topología de los nodos que se acercan o alejan de la cobertura en función de la distancia en saltos a la que se encuentren. Cuando se quiera establecer conexión con un nodo que quede fuera del área de cobertura definida anteriormente, el terminal deberá iniciar un descubrimiento de ruta tal y como lo hacen los protocolos reactivos. El protocolo ZRP (*Zone Routing Protocol*, [8]) presenta un comportamiento proactivo dentro de un área limitada; los nodos que estén dentro de esa área del nodo intercambiarán mensajes de control como lo hacen los protocolos proactivos, mientras que cada vez que quiera comunicarse con un nodo que quede fuera de dicha área, tendrá que solicitar la información de encaminamiento según los protocolos reactivos.

Existe una clasificación diferente dentro de los protocolos de enrutamiento reactivos:

- Basados en la fuente (o en origen). En este tipo de protocolos los paquetes de datos trasportan la ruta completa de la fuente al destino, ruta establecida al enviar el primer paquete. DSR es un ejemplo protocolo basado en la fuente.
- Salto a salto. Los protocolos basados en este concepto únicamente llevan en la cabecera de los paquetes de datos la dirección del destino y la dirección del próximo salto. La información de enrutado la guardan los routers, y cada router decide sólo el siguiente salto. Un protocolo de encaminamiento muy extendido que pertenece a este grupo es AODV.

A continuación se describen las características de algunos de los principales algoritmos de encaminamiento para redes Ad-Hoc mencionados anteriormente.

- DSDV. Este protocolo basado en tablas de encaminamiento y descrito en [1][2] fue diseñado por Charles E. Perkins y Pravin Bhagwat. Cada nodo de la red mantiene una tabla de encaminamiento que contiene todos los posibles destinos y el número de saltos que daría un paquete que viajara hacia el destino especificado. Cada entrada posee un número de secuencia asignado por el nodo destino. Los números de secuencia permiten distinguir rutas antiguas de rutas modernas. Continuamente se deben enviar mensajes de actualización a través de la red para mantener la consistencia de las tablas. Para ayudar a minimizar la gran cantidad de tráfico que ocasionan estas actualizaciones, se utilizan dos tipos de paquetes. El primero recibe el nombre de *full dump*. Este paquete transporta toda la información disponible sobre el encaminamiento y puede requerir que su envío se divida en varias unidades

más pequeñas. Cuando los cambios en la red son pequeños, es raro que se use este tipo de paquete. En cambio, hay un segundo tipo que solo contiene la información que ha variado desde el último *full dump*. Este paquete se llama *incremental*. Los nodos disponen de una tabla adicional donde guardan los datos recibidos por los paquetes *incremental*. Las nuevas rutas contienen la dirección de destino, el número de saltos requeridos para alcanzar al destino, el número de secuencia asociado al destino y un nuevo número que identifica todo el mensaje. En el caso de que haya dos rutas distintas hacia un destino, se usará la que contenga el número de secuencia más moderno. Además, si ambos números coincidieran, la ruta con menor número de saltos sería la que se usaría. En general, DSDV es un protocolo aceptable en escenarios en los que todos los nodos intervienen en las comunicaciones y en los que la movilidad es media. Sin embargo, resulta poco eficiente en entornos de alta movilidad de los nodos puesto que necesita un nuevo número de secuencia hasta que la red está actualizada.

- DSR. Creado por David B. Johnson y Josh Broch [1][6]. Protocolo reactivo basado en la fuente, es decir, basado en el concepto de encaminamiento en origen. Calcula las rutas únicamente cuando es necesario y luego las mantiene. El encaminamiento en el origen es una técnica de enrutado que se caracteriza porque el nodo que realiza la petición determina la secuencia completa de los nodos por los cuales el paquete debe pasar. Este nodo lista explícitamente la ruta en la cabecera del paquete, identificando cada salto que debe producir el paquete, es decir, el nodo siguiente hacia el cual se debe transmitir el paquete en su camino hacia el nodo destino. Hay dos estados principales en el funcionamiento de DSR: descubrimiento de ruta y mantenimiento de ruta. Cuando un nodo quiere enviar un paquete a un destino, primero consulta su caché para determinar si dispone de una ruta hacia el destino. Si tiene una ruta válida, la usará para enviar el paquete. Sin embargo, si el nodo no dispone de dicha ruta iniciará un descubrimiento de ruta enviando un paquete RREQ (*Route REQuest*) hacia todos los nodos dentro de su rango de transmisión. RREQ identifica el nodo por el que solicita la ruta realizando el descubrimiento de ruta. Si el descubrimiento de ruta se realiza con éxito, el nodo que origina la petición de ruta recibe un paquete de respuesta de ruta (RREP, *Route REPly*) que contiene la secuencia de los saltos a través del cual puede alcanzar al nodo destino. Además de las direcciones de los nodos origen y destino de la petición, cada RREQ contiene un registro de ruta, donde se va acumulando la secuencia de saltos que realiza este paquete durante su propagación por la red

durante el descubrimiento de ruta. Mientras un nodo está utilizando una ruta, monitoriza de forma continuada la operación de dicha ruta, lo que se denomina mantenimiento de ruta. Cuando durante el mantenimiento de ruta se detecta un problema con la ruta en uso, el descubrimiento de ruta puede volver a usarse para obtener una nueva ruta actualizada para el destino. Las rutas completas se almacenan en cada nodo para evitar envíos innecesarios de RREQ. Para optimizar el proceso de descubrimiento de ruta, DSR utiliza la memoria caché de manera eficiente. Supóngase que un nodo recibe un RREQ donde no figura como el destino de tal petición y no aparece en el registro de ruta de dicho paquete, y para el cual no se encuentra en su listado de peticiones cursadas recientemente el par formado por la dirección del nodo origen y el identificador de la petición. Si el nodo presenta una entrada en su caché de rutas hacia el destino del RREQ, puede agregar esta ruta almacenada al registro de rutas del paquete, y además puede devolverla en un RREP hacia el nodo que originó la petición sin propagar el RREQ hacia los nodos de su cobertura. El mantenimiento de rutas se completa con el uso de paquetes de error en ruta (RERR, *Route ERRor*). Los paquetes de error en ruta son iniciados por un nodo cuando encuentra un problema en la transmisión con algún enlace. Cuando un RERR es recibido, el nodo que provocó el error es eliminado de la caché de rutas. También serán borradas todas las rutas en las que intervenga el enlace roto. El nodo origen utilizará una ruta alternativa almacenada en la caché o transmitirá un nuevo RREQ. El retardo del descubrimiento de ruta y el número total de paquetes transmitidos se reducen al permitirse añadir datos al mismo RREQ sin necesidad de crear otro mensaje para tal fin. DSR no utiliza mensajes de enrutamiento de forma periódica por lo que se reduce la sobrecarga del ancho de banda de la red, particularmente durante los períodos donde no se producen cambios significativos en la topología de la red. DSR presenta una gran ventaja por ser basado en la fuente, ya que por contener las rutas completas como parte del propio paquete, los lazos en la transmisión de paquetes no se pueden generar puesto que serían detectados y eliminados inmediatamente. Pero el inconveniente es que los mensajes para el encaminamiento son grandes como consecuencia del enrutado basado en la fuente, aumentando el tamaño de dicho paquetes con el tamaño de la red.

- AODV. Creado por Charles E. Perkins [1][5] como evolución de su anterior protocolo (DSDV, [1][2]) más una combinación de DSR [1][6]. Se trata de un protocolo reactivo con enrutamiento salto a salto, donde se mantuvo la idea de

mantener números de secuencia y tablas de encaminamiento, y tomó prestado de DSR el concepto de encaminamiento bajo demanda y los mecanismos de descubrimiento de ruta y mantenimiento de ruta. Cada nodo sólo almacena la información del siguiente salto en una ruta. Utiliza los números de secuencia para asegurar que no se presenten lazos en la transmisión y ofrece rápida convergencia cuando la topología de la red cambia. La optimización primordial que se consiguió en relación a su anterior diseño fue el decremento del tiempo de proceso, disminución del gasto de memoria y reducción del tráfico de control por la red. Los mensajes definidos en AODV son, como para DSR, RREQ, RREP, añadiendo además el error de ruta (RERR, *Route ERRor*). Estos mensajes se reciben vía UDP (*User Datagram Protocol*), y se utiliza el procesado de cabecera de IP (*Internet Protocol*). En cuanto los extremos de una comunicación tienen rutas válidas entre ellos, AODV deja de jugar papel alguno. Cuando se necesita una nueva ruta hacia un nodo destino, el nodo transmite un paquete RREQ hacia los nodos que se encuentran en su radio de transmisión. La ruta puede ser determinada cuando RREQ alcanza el propio nodo destino o bien cuando un nodo intermedio posee una ruta suficientemente moderna hacia el destino, es decir, una ruta válida en la que el número de secuencia es al menos tan grande como la contenida en RREQ. La ruta se vuelve disponible enviando un RREP hacia el nodo originario de la petición. Cada nodo que recibe RREQ almacena en su caché una ruta de vuelta hacia el nodo origen de la petición, de forma que RREP puede ser enviado directamente desde el destino o desde cualquier nodo que contenga información del destino hacia el origen por una sola ruta, sin necesidad de ser retransmitido por cada nodo intermedio a todos sus nodos vecinos. Si un nodo intermedio responde a un RREQ, el nodo destinatario de dicha petición no recibe ninguna copia de él, por lo que no es capaz de conocer la ruta hacia el nodo origen. Esto podría producir que el nodo destino comience un descubrimiento de ruta hacia el origen, por ejemplo, si se pretende establecer una comunicación por medio de TCP (*Transmission Control Protocol*). Así, para que el destino sea capaz de conocer una ruta hacia el nodo originario de la petición, este último debería establecer el *flag gratuitous RREP* en el paquete RREQ, cuando crea que el destino va a necesitar una ruta hacia él. De esta forma si un nodo intermedio responde a un RREQ con este *flag* establecido, devolverá un RREP tanto hacia el nodo origen como hacia el nodo destino. Los nodos monitorizan el estado de los saltos siguientes en las rutas activas. Para mantener las rutas, AODV requiere normalmente que cada nodo periódicamente

transmita un mensaje HELLO, con una tasa de envío por defecto. Si se produce un fallo en la recepción de tres mensajes consecutivos HELLO desde un nodo vecino, se considera que el nodo ha dejado de estar en su cobertura. Cuando se detecta un enlace obsoleto, se utiliza el mensaje RERR para indicar a otros nodos que se ha producido la pérdida de dicho enlace. El mensaje RERR indica los destinos que ahora son inalcanzables debido a la pérdida de un enlace. Para habilitar este mecanismo, cada nodo almacena una lista que contiene la dirección IP de cada uno de sus nodos vecinos que puedan utilizarlo como el siguiente salto hacia el nodo que es ahora inalcanzable. La sobrecarga de la red por enrutamiento aumenta con la movilidad de los nodos, pero no con el aumento del número de nodos. Además, se envían muchos paquetes, pero son de pequeño tamaño.

- ZRP. Creado por Zygmunt J. Haas y Marc R. Pearlman [8]. Es un protocolo híbrido a medio camino entre los algoritmos basados en tablas y los basados en encaminamiento bajo demanda. Es utilizado en una clase particular de redes Ad-Hoc llamadas RWN (*Reconfigurable Wireless Networks*). Estas redes se caracterizan por tener gran cantidad de nodos, mucha movilidad y alto tráfico. Los protocolos anteriores no satisfacían las necesidades específicas de estas redes y los autores se decidieron a crear un nuevo protocolo. ZRP usa zonas similares a *clusters*, en las que los nodos que actúan de bordes se van seleccionando dinámicamente. Además, el radio de estas zonas se reajusta sobre la marcha según las condiciones de la red. Se pueden usar protocolos distintos para comunicarse dentro de las zonas y entre zonas distintas.

1.2. OBJETIVOS

1.2.1. NECESIDAD

Como se ha expuesto anteriormente, el efecto de la movilidad de los nodos hace que el número de saltos entre el nodo origen y el destino varíe dinámicamente, además de que las rutas por donde viajan los paquetes se irán modificando consecuentemente.

Con respecto a la conexión a Internet, algunas de las tecnologías web pueden ser adaptadas a las características de las redes móviles ad-hoc. Es decir, se puede tener un escenario donde uno de los nodos sea una pasarela hacia Internet (GW, *GateWay*), a través de la cual se puede vincular la MANET a servidores HTTP (*HyperText Transfer Protocol*). Pero el acceso a dichos servidores web debe ser estudiado con las peculiaridades de las

redes MANET: el GW no siempre estará accesible permanentemente puesto que la movilidad de los nodos puede conducir a que éste sea inalcanzable. A esta problemática se une la posibilidad de crear cuellos de botella en la comunicación hacia Internet ya que se trata de un medio compartido, y constituye un enlace común para todos los nodos de la red.

El principio de funcionamiento de la red *ad hoc* en estudio sería el siguiente: un nodo solicita un documento a un servidor de datos genérico y la petición es enrutada a través de la red *ad hoc* usando el algoritmo de enrutado definido para dicha red. Cuando el servidor de datos recibe la petición, éste responde enviando el documento solicitado al nodo. Teniendo en cuenta los requisitos planteados el tráfico podría verse notablemente reducido gracias al uso de esquemas de caché particularizados para estas redes *ad hoc* ([9] y [10]).

Este proyecto se va a centrar en escenarios basados en diferentes esquemas de caché:

- **Caché local** [9]: es la forma más simple en que se puede implementar un esquema de caché. Cada nodo va a disponer de una caché local donde almacenará los documentos solicitados con anterioridad, de forma que si el nodo en cuestión necesita de nuevo el mismo documento, éste podrá ser servido directamente desde la caché local (siempre que el documento en cuestión fuera válido, es decir, no estuviera obsoleto).
- **Intercepción de peticiones** [9]: la caché local de cada nodo a su vez tendrá las funcionalidades de un *proxy*, de forma que cada nodo que se encuentre en la ruta de una petición desde el nodo origen hasta el servidor de datos podrá responder a la petición si tiene una copia válida del documento solicitado en su caché local.
- **Redirección de peticiones** [9]: se extiende el planteamiento anterior de intercepción de peticiones de forma que se tenga en cuenta la distribución de los documentos en la MANET. Más concretamente, se puede conseguir que los nodos almacenen el número de saltos existentes hasta donde pueden encontrarse los documentos, pudiendo estar localizados en un nodo a una distancia inferior que el servidor de datos.
- **Intercepción de peticiones con recursos de encaminamiento** [10]: es el denominado *Cross-layer Interception Caching*, donde se propone incluir cierta información en los mensajes de enrutamiento generados al crear la ruta hasta el nodo servidor de datos. De esta forma, cuando un nodo solicita un documento, inicialmente debe crear una ruta hasta el servidor de datos y por tanto los nodos que reciban dichos mensajes de enrutamiento pueden responder informando si tienen una copia válida del documento buscado.

Debido al rápido crecimiento de las redes inalámbricas en general, se han requerido herramientas capaces de ofrecer de forma gráfica las conclusiones obtenidas a partir de los datos de simulaciones de redes. Las necesidades de un entorno de visualización para los resultados de las simulaciones realizadas con NS-2 han constituido la base fundamental para la creación de este proyecto.

1.2.2. VISUALIZADORES DE REDES MANET

Este estudio se ha realizado con el apoyo de NS-2 (*Network Simulator 2*) [11], un popular entorno para simulación de redes, cuyo número de usuarios ha crecido enormemente en los últimos años. Aunque inicialmente fue creado para la simulación de redes cableadas, actualmente su funcionalidad ha sido extendida para trabajar con redes inalámbricas, incluyendo WLAN (*Wireless Local Area Network*), redes de sensores y redes MANET. En un escenario típico de NS-2, el usuario describe la configuración de la red y parámetros (como la movilidad y patrones de tráfico), que se codifican como entradas de datos, y cuando la simulación ha finalizado, NS-2 almacena los resultados de la simulación en un archivo de texto denominado archivo de traza. Generalmente, los archivos o ficheros de traza contienen enormes cantidades de datos en texto, codificados según una especificación dada. Por tanto, existe la necesidad de una herramienta que transforme los archivos de trazas en una animación visual del proceso de simulación. Esta herramienta debería posibilitar al investigador usar el sistema visual, una ventaja indiscutible en términos de esfuerzo para hacer análisis y depurar un protocolo de red. La animación podría incluir la distribución o trazado de la red, los patrones de movilidad de los nodos, las transferencias de paquetes, detalles sobre la transmisión de datos, valores de estructuras internas de los nodos, etc. La herramienta NAM (*Network ANimator*) [11], que es el visualizador por defecto para los resultados de simulación de este entorno, es capaz de procesar grandes cantidades de datos rápidamente. Además, las visualizaciones añaden análisis estadísticos. Por estas razones, NAM fue diseñado para proveer una interfaz gráfica al usuario para la creación de topologías de red cableadas, y es capaz de mostrar los enlaces y los flujos de paquetes. Desafortunadamente, no ha sido adaptado para las nuevas características de redes inalámbricas, y no es capaz de mostrar la movilidad de los nodos. Existió un intento a finales de los años 90 de añadir dicha capacidad a NAM denominado *ad-hockey* [12], que fue introducido por el proyecto Monarch [13] de CMU (*Carnegie Mellon University*) pero no fue continuado y las versiones recientes de NS-2 no lo incluyen.

Se van a discutir a continuación algunas de las soluciones que existen para la visualización y animación de redes basadas en simulaciones con NS-2.

Actualmente, existe una herramienta de simulación que se está empezando a extender como apoyo a NS-2 para cubrir esta necesidad denominada iNSpect (*interactive NS-2 protocol and environment confirmation tool*) [14][15]. Se trata de una herramienta desarrollada en C++ que permite el análisis de redes inalámbricas simuladas en NS-2. Soporta simulaciones de redes inalámbricas y parece que existe la intención de soportar redes cableadas para un futuro. Su principal característica es que es capaz de manejar diferentes tipos de archivos de entrada, aceptando archivos de traza propios de NS-2 y archivos de movilidad utilizados por NS-2, pero necesita un tipo especial de traza denominada *vizTrace*. Adicionalmente, puede validar el propio NS-2 ya que es capaz de comprobar si NS-2 maneja el patrón de movilidad de forma correcta. Por último, es capaz de ofrecer parámetros estadísticos de la red; sin embargo, no ofrece información acerca de los datos de la simulación actual, como el tipo de paquetes procesados, el tamaño, las direcciones IP de los nodos origen y destino de cada salto, ni es capaz de dar ninguna información acerca de estructuras internas de los nodos. Los parámetros de las redes a representar por dicho visualizador están bastante controlados y estipulados, y por tanto, no responden al estudio de la nueva funcionalidad de esquemas de caché expuesto en este proyecto. Además, para el uso de la herramienta, el usuario deberá introducir los archivos de trazas según el formato adecuado para su reconocimiento, sin la posibilidad de añadir las nuevas variables estadísticas de interés.

Huginn [16] es otro entorno para simulaciones de redes inalámbricas en NS-2 escrito en C++. Su principal característica es el uso de gráficos 3D y un sofisticado esquema para la presentación de los datos. El usuario puede especificar un diagrama de flujo que define el filtrado de los datos, a través del cual se puede indicar la información en la quiere que el entorno se centre. Sin embargo, Huginn tiene las mismas limitaciones de las herramientas previas, y no es capaz de acceder a estructuras internas de los nodos, sin poder mostrar la información de las simulaciones con las necesidades planteadas.

EXAMS (*EXtensible Animator for Mobile Simulations*, [17]) es un completo simulador que añade funcionalidades no presentes en los simuladores previos, como la posibilidad de mostrar información interna del estado de los nodos, además que es capaz de mostrar las coberturas de nodos, e incluso ofrecer datos estadísticos de la red, aunque no contienen los datos de nuestro interés. Su principal ventaja es que representa una infraestructura que, con ayuda del diseñador del protocolo y de los desarrolladores, se pueden añadir módulos de

protocolos de enrutamiento desarrollados a posteriori para que sea capaz de interpretarlos. Uno de sus inconvenientes es que no realiza interpolación de posiciones de los nodos, si bien los resultados que muestra en la animación tienen una relación 1 a 1 respecto a las líneas del archivo de trazas, sin poder ofrecer una completa validación del patrón de movilidad seguido por los nodos. Además, no ofrece la posibilidad de mostrar los parámetros que se buscan en el estudio realizado.

Por último, MobiSim [18] es una aplicación gratuita que se ha desarrollado con el fin de dar soporte a los investigadores de redes MANET para el estudio de la movilidad en este tipo de redes. El usuario podrá crear escenarios de movilidad, simularlos de forma gráfica, evaluar medidas sobre las trazas y reconocer patrones de modelos de movilidad de nodos inspeccionando dichas trazas. Soporta varios formatos de texto para las trazas y contiene una amplia gama de modelos de movilidad, que además pueden ser entremezclados. Pero el inconveniente que ofrece desde el punto de vista de la necesidad planteada es que no es capaz de interpretar el tráfico producido en la red, y por tanto, no se puede utilizar para obtener los datos necesarios del estudio de los esquemas de caché anteriormente citados.

1.2.3. SOLUCIÓN PROPUESTA

El análisis visual de entornos inalámbricos es muy importante por tres áreas clave de la investigación de simulaciones basadas en NS-2:

1) validar la precisión de la salida de un modelo de movilidad y/o los archivos con la topología de los nodos usados para realizar la simulación. Es decir, una herramienta de visualización hace posible estudiar una nueva generación de patrones de movilidad más realistas y los comportamientos complejos de los nodos.
2) validar las nuevas versiones del propio simulador NS-2. Nuevas versiones van apareciendo regularmente y los investigadores son los responsables de verificar que sus simulaciones siguen siendo válidas con los modelos implementados en NS-2.
3) analizar los archivos de trazas de NS-2 resultantes de la simulación. Es más, tras desarrollar nuevos protocolos y técnicas, los resultados de la animación deben ser analizados tanto estadísticamente como visualmente para comprobar que concuerdan con las hipótesis y/o análisis realizado.

Además, para la necesidad planteada del estudio de redes MANET con caché local, es necesario disponer de una herramienta que se ajuste a él, y las soluciones anteriormente mencionadas no ofrecen una solución a los nuevos requisitos.

Debido a todo esto se presenta una solución alternativa en este proyecto, de forma que se desarrolle una aplicación de escritorio que cubra las nuevas necesidades y sea capaz de representar los datos relacionados con la movilidad de los nodos respecto al tiempo, el tráfico que circula por la red, el estado de las memorias caché y de las caché de redirecciones de cada uno de los nodos y que, adicionalmente, pueda generar un informe externo que contenga un resumen de estadísticos para todos los nodos de la red válidos en un determinado instante de tiempo. Además, la herramienta creada estará orientada exclusivamente al estudio de las cachés planteado en el epígrafe 1.2.1. Se plantea Java [19] como lenguaje de programación para tal fin, puesto que presenta las siguientes características:

- Es multiplataforma. Pensemos que de los simuladores alternativos anteriormente presentados son también multiplataforma pero algunos requieren de la instalación de compiladores GCC (*General public license Compiler Collection*) para el sistema operativo Windows.
- Existen entornos de desarrollo comúnmente utilizados.
- Tiene potentes API (*Application Program Interface*) y librerías para el manejo de funciones gráficas. Una de esas API será la que se utilice para la programación gráfica, de reducido tamaño en memoria y con gran facilidad de instalación para cualquier sistema operativo.
- Sólo requiere de la máquina virtual Java (JVM, *Java Virtual Machine*) que no presenta problemas para su instalación (en muchas ocasiones ya viene instalada), y de la API utilizada para su ejecución, ya sea sobre plataformas Windows como Linux.
- Soporta la construcción de interfaces GUI (*Graphical User Interfaces*) a través de las librerías SWING y AWT (*Abstract Window Toolkit*).
- Las aplicaciones Java son fácilmente desplegables a través de archivos JAR (*Java ARchive*).
- Presenta una amplia aceptación por parte de la comunidad académica y de investigación.
- Es software libre.

Una ventaja clara de la herramienta creada es que es capaz de animar una MANET sin necesitar la ejecución del entorno NS-2, ya que toma como datos de entrada los archivos de movilidad de los nodos (que es una entrada al propio NS-2) y los archivos resultantes de simulaciones realizadas con éxito por NS-2 (es decir, salida de NS-2). Es capaz por tanto de realizar un post-procesado, reduciendo así la sobrecarga. Otra característica a destacar es que la aplicación no está limitada a los resultados de simulación de NS-2, ya que es capaz de generar visualizaciones a partir de ficheros de traza y de movilidad de los nodos que sean conformes al formato aceptado.

1.3. ESTRUCTURA DEL DOCUMENTO

El contenido de la presente memoria se encuentra estructurado en siete capítulos, los cuales van a ir presentando toda la información necesaria para comprender el estudio teórico de cachés que se quiere visualizar, así como la descripción del desarrollo de la aplicación de escritorio y su manual de funcionamiento, finalizando con la posibilidad de extender la funcionalidad presentada en la aplicación desarrollada.

- En el Capítulo 1, se presenta la introducción de la memoria, explicando brevemente las redes MANET, las ventajas e inconvenientes asociados a ellas, así como los protocolos de encaminamiento más comúnmente utilizados en este tipo de redes, y se plantea la necesidad del proyecto, el estado del arte y la solución propuesta, que es la solución que se desarrolla a lo largo de los demás capítulos. Se introduce además el resto de capítulos que conforman el documento.
- En el Capítulo 2, se explican las tecnologías utilizadas para el desarrollo del proyecto, tanto para las fases de diseño y desarrollo como para la codificación.
- En el Capítulo 3, se pretende aproximar al estudio realizado en relación a las cachés en los nodos, indicando las ventajas del uso de estas técnicas para optimizar algunos parámetros de las redes MANET.
- En el Capítulo 4, se comienza introduciendo una serie de conocimientos previos de la disciplina de Ingeniería del Software, necesarios para abordar el tema de las características de las fases del diseño software, así como las ventajas del modelado de sistemas software. Posteriormente, se centra en el objetivo de este proyecto, realizando el análisis de los requisitos de la aplicación, el diseño de la misma, así como añadiendo diagramas explicativos necesarios para comprender el análisis y el diseño del software que finalmente se ha codificado.

- En el Capítulo 5, se presenta la última parte del diseño software, que conforma a la verificación y validación del software, que contiene el plan de pruebas que se ha llevado a cabo, en el cual se detallan todas y cada una de las pruebas realizadas al sistema para comprobar su correcto funcionamiento, al margen de las realizadas durante la fase de desarrollo.
- En el Capítulo 6, se adjunta el manual de usuario mostrando los controles que presenta la aplicación, de forma que el usuario pueda explotar todas las ventajas de la herramienta desarrollada.
- En el Capítulo 7, se destacan las conclusiones y las líneas futuras, donde se hace un pequeño balance del proyecto, su utilidad, y se propone una idea de cuáles podrían ser las posibles líneas de investigación que permitan extender la funcionalidad del sistema realizado.

Ha sido necesaria la introducción de un anexo:

- En el Apéndice A, se proporciona un listado con el formato de mensajes que la herramienta interpreta, así como el significado de los mismos.

CAPÍTULO 2: Tecnologías empleadas

2.1. JAVA

Prácticamente desde que empezaron a surgir lenguajes de programación de alto nivel su ciclo de vida siempre ha sido el mismo, nacen, se extienden, y pasado un cierto tiempo, surge otro que lo supera y el primero cae en el olvido. Desde hace unos años parece que este paradigma ha cambiado, o al menos, se ha frenado bastante con la aparición de Java.

El lenguaje de programación Java, desarrollado por Sun Microsystems bajo las directrices de James Gosling y Bill Joy, está diseñado para ser un lenguaje independiente de la máquina que sea lo suficientemente seguro para atravesar redes. Inicialmente, la mayoría del entusiasmo de Java se centró en aplicaciones embebidas para WWW (*World Wide Web*), denominadas *applets*. Los *applets* pueden ser programas independientes por sí mismos, o incluso pueden representar la interfaz para programas descargados de un servidor y ejecutados en el lado del usuario. Más recientemente, Java ha incorporado un conjunto de herramientas para la construcción de interfaces gráficos para aplicaciones del lado del usuario, lo que le ha permitido ser una plataforma popular para el desarrollo de software de aplicaciones. También ha llegado a ser una plataforma importante para aplicaciones del lado del servidor utilizando la interfaz *servlet*, para aplicaciones de entorno empresarial utilizando tecnologías como EJB (*Enterprise JavaBeans*) y para modernas aplicaciones distribuidas.

Los inicios de Java se pueden asociar al investigador jefe de Sun Microsystems, Bill Joy, hacia el año 1990. El lenguaje de programación Java fue denominado originalmente *Oak*, y fue diseñado para su uso en aplicaciones para electrónica de consumo por James Gosling. *Oak* debía ser independiente de la plataforma, dado el gran número de modelos que ya había en el mercado, por lo cual se optó por un lenguaje interpretado. Además, el nuevo lenguaje debía ser robusto y a la vez sencillo para evitar errores por parte del programador que pudieran llevar al cuelgue del sistema. Esto motivó que se eliminaran las características que hacían que el código fuera más propenso a errores, como la herencia múltiple. El resultado fue un lenguaje que tenía similitudes con C y C++ y que no estaba ligado a un tipo de CPU concreta. Tras varios años de experiencia con el lenguaje, y las significativas contribuciones de Ed Frank, Patrick Naughton, Jonathan Payne y Chris Warth fue relanzado para Internet y revisado sustancialmente hacia el lenguaje que es hoy

en día. Más tarde se le cambiaría el nombre de *Oak* a Java, por cuestiones de propiedad intelectual, al existir ya un lenguaje con el nombre *Oak*. Se supone que le pusieron ese nombre mientras tomaban café (Java es también el nombre de un tipo de café, originario del este de Asia, de la isla del mismo nombre), aunque algunos autores afirman que el nombre deriva de las siglas de **J**ames Gosling, **A**rthur **V**an Hoff, y **A**ndy Bechtolsheim, que son algunas de las personas que le dieron la forma que presenta en la actualidad.

2.1.1. CARACTERÍSTICAS DEL LENGUAJE DE PROGRAMACIÓN JAVA

El lenguaje de programación Java es un lenguaje concurrente de propósito general orientado a objetos y basado en clases, específicamente diseñado para que contenga las mínimas dependencias de implementación como sea posible, y para que sea lo suficientemente simple para que los programadores puedan adquirir fluidez en el lenguaje. Éste es su lema fundamental: permite a los desarrolladores de aplicaciones que escriban el programa una vez, y sean capaces de ejecutarlos en cualquier parte.

Java es tanto un lenguaje interpretado como compilado. El código fuente de Java se convierte en simples instrucciones binarias, muy parecidas al código máquina de un microprocesador. Sin embargo, mientras el código fuente en C o C++ se redefine como instrucciones nativas para un modelo o procesador particular, el código en Java es compilado en un formato universal, que son las instrucciones de su máquina virtual. El código compilado de Java o *bytecodes* se ejecutan por el intérprete de Java en tiempo de ejecución sobre la máquina virtual de Java (JVM).

Históricamente, los intérpretes han sido considerados lentos, pero en el caso de Java, es relativamente rápido porque ejecuta los *bytecodes*. Además, Java ha sido diseñado para que las implementaciones del software en el sistema de ejecución puedan optimizar su rendimiento al compilar los *bytecodes* a código máquina nativo sobre la marcha. Sun defiende que el código Java se puede ejecutar casi tan rápido como el código compilado nativo y que además, mantiene la portabilidad y seguridad.

JVM sí que es dependiente del sistema operativo, y se puede instalar de forma gratuita. El sistema en ejecución lleva a cabo todas las actividades normales de un procesador real, pero lo hace en un entorno virtual y seguro. Ejecuta el conjunto de instrucciones y controla la pila de memoria. Lo más importante es que la forma de proceder de JVM se realiza en concordancia con una especificación abierta estrictamente definida que puede hacer que

sea implementado por cualquiera que quiera producir una máquina virtual de Java que sea compatible. La máquina virtual y la definición del lenguaje proveen de forma conjunta una especificación completa; no hay características de Java que no estén definidas y que sean dependientes de la implementación. Por ejemplo, Java especifica el tamaño de todos sus tipos de datos primitivos, sin dejar que sean dependientes de cada implementación. El intérprete de Java es relativamente ligero y ocupa poco espacio, y puede ser implementado de forma que sea compatible con una plataforma en particular. El intérprete puede ser ejecutado como una aplicación separada, o puede ser embebido en otro software, como en un navegador web. La misma aplicación en Java, gracias a los *bytecodes*, puede ser ejecutada en cualquier plataforma que provea un entorno de ejecución para Java, como se muestra en la figura 2.1:

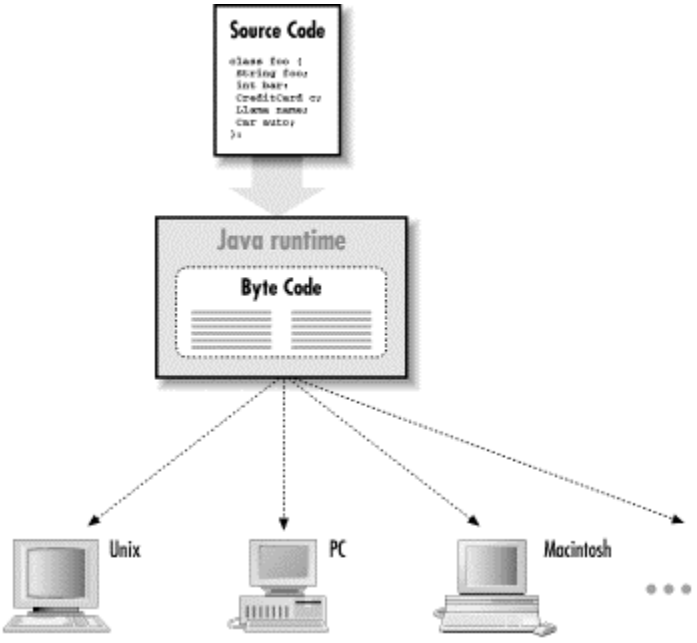


Figura 2.1. *Bytecodes* de Java

La unidad fundamental del código Java es la clase. Como en otros lenguajes orientados a objetos, las clases son componentes de la aplicación que contienen código ejecutable y datos. Las clases compiladas de Java se distribuyen en un formato binario universal que contiene los *bytecodes* y otra información de clase. Las clases se pueden almacenar en archivos en el sistema local o en un servidor. Las clases se localizan y cargan en tiempo de ejecución cuando son necesarias para una aplicación. Java además contiene un número de clases que dependen de la plataforma, denominados métodos nativos, que sirven de pasarela entre el sistema de ventanas, la red y el sistema de archivos. El resto de Java se encuentra escrito completamente en java; de ahí a su portabilidad.

Hasta la fecha, la plataforma Java ha atraído a más de 6,5 millones de desarrolladores de software [20]. Se utiliza en los principales sectores de la industria de todo el mundo y está presente en un gran número de dispositivos, equipos y redes.

2.1.2. JDK, J2SE Y JRE

Conocido originalmente como Kit de desarrollo de software de Java (JDK, *Java Development Kit*), el kit de desarrollo de software de Java 2 (SDK, *Software Development Kit*) es lo que se utiliza para crear programas para J2SE (*Java 2 Platform, Standard Edition,* [21]). La segunda versión de Java (Java 2) añadió numerosas mejoras respecto de la primera versión, incluyendo una librería GUI (Swing), accesibilidad y librerías 2D, funciones de arrastrar y soltar, soporte a algunos archivos de audio, certificados digitales y herramientas de seguridad mejoradas. SDK incluye todas las herramientas y bibliotecas estándar de Java necesarias para crear *applets* y aplicaciones. Si se desea crear programas de servidor, *servlets* y otros programas de servidor, necesitará tener J2EE (J*ava 2 Enterprise Edition*), que funciona sobre J2SE. Y para el desarrollo de *midlets* (programas que se ejecutan en dispositivos de información móviles) y aplicaciones inalámbricas se tiene J2ME (*Java 2 Micro Edition*).

SDK está lleno de herramientas en línea de comandos para la creación y ejecución de programas Java. Además, contiene un compilador para convertir el código fuente Java en programas ejecutables. También se puede encontrar la manera de documentar, eliminar errores e integrar programas Java con programas basados en C/C++, además de aplicaciones CORBA. Desde editar hasta compilar y ejecutar, todas estas tareas de desarrollo pueden realizarse gratuitamente; ninguna de ellas requiere coste alguno, al menos por lo que respecta a las plataformas de referencia Sun (actualmente perteneciente a la compañía Oracle).

JRE (*Java Runtime Environment*) es el entorno en el que se ejecutan los programas Java. Para los *applets*, es el navegador el que proporciona este entorno. Por otra parte, para las aplicaciones, se puede proporcionar el JRE con los programas, o bien confiar en que el usuario ya lo tiene instalado. Las aplicaciones pueden reemplazar el JRE fácilmente y podrá asegurarse de que el usuario está usando la versión con la que se probó el programa.

Cuando se comienza a crear programas en Java, muchos desarrolladores comienzan con el equipo de desarrollo de Sun, el SDK. Lo mejor, como se ha dicho anteriormente, es que es gratuito y siempre se puede trabajar con la última versión. Todo lo que hay que hacer es proporcionar el editor preferido y ya se tiene todo lo necesario. Como parte del equipo de

desarrollo se obtiene JRE y un conjunto de bibliotecas para sus programas. La combinación del entorno de ejecución y las bibliotecas suele denominarse *Java Platform*. Con cada nueva versión de Java, el conjunto de bibliotecas cambia, pero el entorno de ejecución subyacente permanece inalterado.

Las bibliotecas que acompañan a cada versión de Java son lo que realmente diferencia a Java de la mayoría de los otros lenguajes de programación. Así, cuando se conoce su versión de Java, se conocen las bibliotecas que puede esperar encontrar. Las bibliotecas disponibles con la plataforma Java le permiten hacer casi cualquier cosa. Para empezar a trabajar con Java, se deberá descargar el SDK [22]. Durante la historia de Java, Sun ha introducido varias veces una nueva convención de nombres para el versionado de Java, tanto para las propias versiones de Java, como para las herramientas de desarrollo y el entorno de ejecución. Por ejemplo, JDK 1.2 fue llamado *Java 2 Platform, Standard Edition 1.2 (J2SE 1.2)*. Más recientemente, Sun anunció que la quinta generación de su edición estándar sería denominada *Java 2 Platform, Standard Edition 5.0* (*J2SE 5.0*), en lugar del esperado *Java 2 Platform, Standard Edition 1.5.0* (*J2SE 1.5.0*). Así, el factor 5.0 identifica al número de versión externamente, mientras que el factor 1.5.0 es usado como el número identificador de la versión pero internamente. Antes de salir la última versión, se acordó simplificar más aún el nombre, de forma que el factor ".0" sería eliminado. Así, a la última versión de Java se le conoce como Java SE 6 (*Java Platform, Standard Edition 6*). Esa es la versión utilizada para el desarrollo de la aplicación, de forma que externamente se representa como Java SE 6 pero internamente es la versión 1.6.0. Concretamente, se ha utilizado la actualización 16 de Java SE 6, coincidiendo con la versión de JRE. Esta versión ha sido desarrollada bajo las pautas de la especificación JSR (*Java Specification Request*) 270 [23].

2.2. NETBEANS

El entorno de desarrollo y plataforma para aplicaciones Netbeans [24] es una herramienta de código abierto, escrito en lenguaje Java. Provee servicios comunes para la creación de aplicaciones de escritorio, como manejo de ventanas y de menús, y es el primer entorno integrado de desarrollo (IDE, *Integrated Development Environment*) que soporta completamente las características de JDK 6.0. La plataforma e IDE Netbeans es gratuita, tanto para uso comercial o no comercial, y está soportado por Sun Microsystems. Se encuentra disponible para diversos sistemas operativos, como Windows, Mac, Linux y

Solaris. Normalmente suele descargarse de Internet [25] junto con el JDK, lo que facilita la instalación de forma conjunta. Este IDE presenta facilidades a los desarrolladores de Java para la creación de aplicaciones de escritorio, web, empresariales y para móviles, así como soporte para otros lenguajes de programación como C/C++, PHP, JavaScript y Ajax, Ruby y Ruby on Rails, etc. Además, existen tutoriales muy completos acerca de su funcionamiento [26]. En la figura 2.2 se presenta una instantánea de la plataforma Netbeans.

Utilizar IDE para desarrollar aplicaciones ahorra considerable tiempo. Además, un IDE puede almacenar tareas repetitivas a través de macros, y es de gran ayudar a la hora de depurar programas, puesto que la mayoría de ellos incluyen la detección de errores sintácticos, además de los errores en tiempo de compilación.

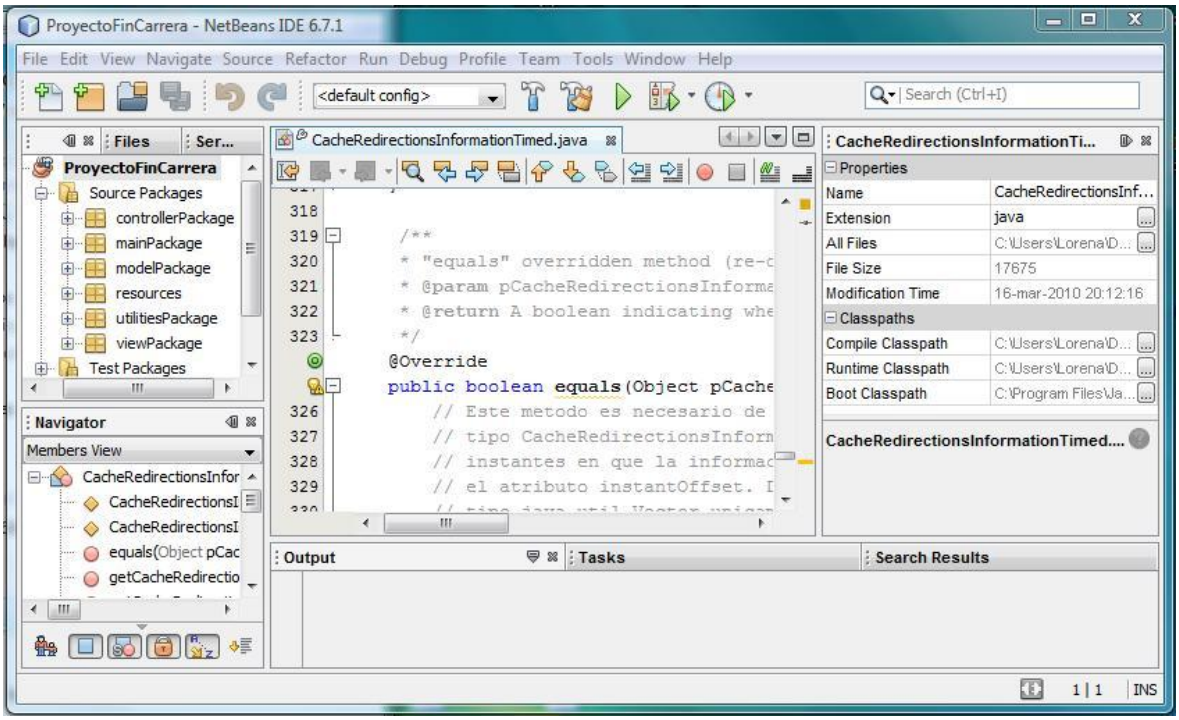


Figura 2.2. Entorno Integrado de Desarrollo Netbeans

Pero este entorno no se ha utilizado únicamente para la codificación de la aplicación. Netbeans ofrece muchas más funcionalidades gracias a la posibilidad de utilizar complementos o *plugins*, que no son más que aplicaciones adicionales que son ejecutadas por la aplicación principal (en este caso Netbeans) e interactúan por medio de la API. Uno de estos complementos ha sido utilizado para las fases de análisis y diseño de la aplicación con el fin de obtener los diagramas UML (*Unified Modeling Language*), que se presentarán en el capítulo 4.

2.3. PDF

PDF (*Portable Document Format*, [27]) es un formato de almacenamiento electrónico de documentos ampliamente utilizado. Su popularidad se debe, entre otras características, a su habilidad para reproducir salidas de alta calidad en una gran variedad de plataformas diferentes. Fue creado por Adobe Systems Incorporated. Adobe provee una aplicación gratuita denominada Acrobat Reader para la lectura de documentos PDF, y está disponible en los sistemas operativos más utilizados, como GNU/Linux, Windows, Unix, Mac y otros.

Está especialmente ideado para documentos susceptibles de ser impresos, ya que especifica toda la información necesaria para la representación final del documento, determinando todos los detalles de cómo va a quedar, y sin requerir procesos anteriores de ajuste ni maquetación. Algunas de sus características principales son:

- Es multiplataforma, pudiéndose visualizar en cualquier sistema operativo con la misma estructura que el documento original.
- Puede integrar cualquier combinación de texto, elementos multimedia como vídeos y sonidos y elementos de hipertexto como vínculos y marcadores.
- Es uno de los formatos más extendidos en Internet para el intercambio de documentos.
- Se trata de una especificación abierta, facilitando que se puedan encontrar herramientas de software libre para visualizar y modificar documentos en formato PDF.
- Puede cifrarse para proteger su contenido e incluso firmarlo digitalmente.

Por último, puede crearse desde varias aplicaciones exportando el archivo en cuestión. En esta característica radica el uso de documentos PDF para la visualización de los estadísticos calculados en la aplicación. La herramienta desarrollada presenta la funcionalidad extra de generar informes de estadísticos bajo demanda para cualquier instante de tiempo de la simulación, exportando toda esta información en archivos PDF, como otras herramientas hacen. El informe en PDF, denominado "*statistics.pdf*", se genera gracias al uso de las bibliotecas gratuitas de Java iText, jCommom y JFreeChart, que se verán con más detalle en el siguiente epígrafe. La estructura y el aspecto del documento PDF generado por la aplicación se explicarán con más detalle en el epígrafe dedicado al manual de usuario.

2.4. BIBLIOTECAS EXTERNAS DE JAVA

2.4.1. iTEXT

iText [28] es una biblioteca de código abierto que se utiliza para crear y manipular archivos PDF, RTF (*Rich Text Format*) y HTML (*HyperText Markup Language*). Fue escrita, entre otros, por Bruno Lowagie y Paulo Soares. Para la aplicación desarrollada en este proyecto, se utiliza durante la generación de documentos PDF sobre la marcha, es decir, en el mismo instante de la petición. Es ideal para aquellos desarrolladores que pretenden crear aplicaciones de escritorio o aplicaciones web mejoradas con la creación dinámica y/o manipulación de documentos PDF. iText no se trata de una herramienta en el usuario final, sino que se incorpora a modo de clases externas que se pueden utilizar para la automatización del proceso de creación y manipulación de documentos PDF. La necesidad de creación de este tipo de documentos bajo demanda reside en dos factores clave: la información que contiene depende de los datos de entrada de la aplicación y la exigencia de la forma de presentación deseada, como la inclusión de histogramas y de tablas. Por estos dos factores, por la posibilidad que ofrece para crear documentos que sean independientes de la plataforma, y por estar desarrollado para el lenguaje de programación Java (entre otros lenguajes de programación), se ha pensado en iText como la solución más acertada. La versión utilizada de iText para el desarrollo de la aplicación es la 5.0.0.

2.4.2. JFREECHART

JFreeChart [29] es una biblioteca para Java gratuita que se utiliza para la generación de diagramas profesionales de calidad en las aplicaciones que la usen. Dentro de las características principales destaca que se trata de una API bien documentada, soporta una gran cantidad de tipos de diagramas, presenta un diseño flexible y es capaz de generar los diagramas en diversos formatos, incluyendo imágenes en PNG (*Portable Network Graphics*) y JPEG (*Joint Photographic Experts Group*). La versión utilizada de JFreeChart para el desarrollo de la aplicación es la 1.0.13, y necesita de la biblioteca JCommom para su correcto funcionamiento.

2.4.3. JCOMMON

JCommon [30] es una colección de clases de utilidad para aplicaciones Java usada por JFreeChart, así como por otros proyectos. Esta librería incorpora una gran cantidad de clases para diversas funcionalidades que resultan muy útiles para numerosas aplicaciones. Particularmente, es muy utilizada por la magnífica utilidad de calendario que incorpora, así

como por el manejo del centrado automático de ventanas en la pantalla que ofrece, entre otras. La versión utilizada de JCommon para el desarrollo de este proyecto es la 1.0.16.

2.5. JAVA COMO LENGUAJE DE PROGRAMACIÓN GRÁFICA

Para muchos programadores gráficos, la idea de utilizar Java resultaba, en principio, una mala solución. De hecho, existen muchas ideas preconcebidas acerca del uso de este lenguaje para fines gráficos, como las siguientes: es demasiado lento para programación de juegos, tiene fugas de memoria, es demasiado a alto nivel y no está soportado en consolas de juegos. Pero no dejan de ser críticas sin fundamento real, o al menos que no corresponden a las versiones actuales de Java, puesto que puede presentar velocidades cercanas a las que presenta C++ (es más, con cada versión se va mejorando, e incluso un cliente de Java [31] se atreve a decir que Java SE 6 es del orden del 20% al 25% más rápido que J2SE 5.0), las fugas de memoria pueden evitarse con buenas técnicas de programación, y Java es de alto nivel pero ofrece acceso al hardware para gráficos y a dispositivos externos. Además muchas de las mejoras realizadas en Java SE 6 están relacionadas con el uso de OpenGL [32] y DirectX [33] para la carga gráfica. Java se está extendiendo en la actualidad para aplicaciones gráficas, y existe un número actualmente en crecimiento de excelentes juegos comerciales basados en Java, entre los que se encuentran: *Tribal Trouble* [34], *Puzzle Pirates* [35], *Call of Juarez* [36] y *Star Wars Galaxies* [37]. Con todo esto, se ha elegido el lenguaje de programación Java para el desarrollo de la aplicación del proyecto por las características anteriormente expuestas, que muestran que es potente para la creación de escenarios gráficos. Además, hay que considerar las facilidades ofrecidas para GUI, y sobre todo, que es indudablemente uno de los lenguajes de programación que más se utiliza en la actualidad [38] facilitando así el futuro mantenimiento y/o extensión de la aplicación.

2.5.1. HERRAMIENTAS PARA LA PROGRAMACIÓN GRÁFICA BASADAS EN EL LENGUAJE JAVA

En la actualidad, existen diferentes proyectos para el desarrollo de herramientas para la programación gráfica basados en Java, pero nuestro estudio se va a centrar en los dos principales, que son Java3D [39] y JOGL (*Java OpenGL*) [40].

Java3D es una API de alto nivel basada en grafos de escena para la programación de aplicaciones con gráficos en 3D, que se encarga de la construcción de las estructuras de datos que contienen los objetos que aparecen en la escena 3D. Se ejecuta sobre OpenGL o Direct3D [41] y, comparado con otras soluciones, no es únicamente un envoltorio para esas API gráficas, sino más bien una interfaz que encapsula la programación gráfica utilizando un concepto orientado a objetos real. Una escena se construye a partir de un grafo de escena que representa los objetos que deben ser mostrados. Este grafo de escena está estructurado como un árbol que contiene varios elementos que son necesarios para mostrar los objetos. Java3D y su documentación se encuentran disponibles para la descarga de forma separada, y no son parte del JDK.

JOGL es una biblioteca que sirve de envoltorio para acceder a la OpenGL (que está escrita en C) mediante la programación en Java, pero ofreciendo una abstracción no tan alto nivel como Java3D. La API OpenGL es llamada por JOGL gracias a la JNI (*Java Native Interface,* [42]). Actualmente está siendo desarrollado por Sun Microsystems. JOGL permite acceder a la mayoría de características disponibles para los programadores de C, con la excepción de las llamadas a ventanas realizadas en GLUT (openGL Utility Kit), ya que Java contiene sus propios sistemas de ventanas con AWT y SWING, y algunas extensiones de OpenGL. JOGL se diferencia de otras bibliotecas Java para OpenGL en que simplemente expone las funciones de la API OpenGL, basadas en lenguaje C, por medio de métodos contenidos en unas pocas clases, en lugar de intentar realizar un mapeo completo del código OpenGL para transformarlo y adaptarlo al paradigma de orientación a objetos.

La fina capa de abstracción proporcionada por JOGL hace que la ejecución sea muy eficiente, aunque resulta mucho más difícil de programar que otras bibliotecas de mucho más alto nivel como Java3D. La naturaleza procedural y de máquina de estados de OpenGL es inconsistente con la forma habitual de programar en Java, lo cual puede dejar perplejos a muchos programadores. Por esto se ha decidido utilizar Java3D para el desarrollo del proyecto.

2.5.2. JAVA3D

La API Java3D es una interfaz de programación utilizada para realizar aplicaciones y *applets* con gráficos en tres dimensiones. Proporciona a los desarrolladores un alto nivel de abstracción para crear y manipular objetos geométrios 3D y para construir las estructuras utilizadas en el **renderizado** de dichos objetos. Se pueden describir grandes mundos

virtuales utilizando estos constructores, que proporcionan a Java3D la suficiente información para hacer un renderizado de forma eficiente. Java3D proporciona a los desarrolladores de gráficos 3D la principal característica de Java: escribe una vez y ejecútalo donde sea. Java3D es parte del conjunto de APIs JavaMedia, lo cual hace que esté disponible en un gran número de plataformas. También, se integra correctamente con Internet ya que tanto los applets como las aplicaciones escritas utilizando Java3D tienen acceso al conjunto completo de clases de Java.

Los objetos geométricos creados por los constructores residen en un **universo virtual**, que luego es renderizado. La API está diseñada con flexibilidad para crear universos virtuales precisos de una amplia variedad de tamaños, desde astronómicos a subatómicos.

Un programa Java3D crea ejemplares de objetos y los sitúa en un estructura de datos de **escenario gráfico**. Este escenario gráfico es una composición de objetos 3D en una estructura de árbol que especifica completamente el contenido de un universo virtual, y cómo va a ser renderizado.

Java3D introduce algunos conceptos que no se consideran habitualmente como parte de los entornos gráficos, como el sonido espacial 3D. Las posibilidades de sonido permiten proporcionar una experiencia más realista al usuario.

Java3D está desarrollado por Sun Microsystems, es gratuita y está disponible para la mayoría de las plataformas [43]. Hay versiones para Solaris/Sparc, Solaris/x86, Linux, Linux/x64, Mac OS X, Windows y Windows/x64. La instalación es realmente sencilla, ya que sólo es necesaria la descarga de un archivo de Internet, y los requisitos que debe cumplir el equipo son los mismos que se necesitan para poder trabajar con Java en general. Es requisito tener instalado esta API para poder ejecutar la aplicación.

La API define unas 100 clases presentadas en el paquete corazón denominado javax.media.j3d. Además, se usan otros paquetes para escribir programas Java3D. Uno de estos paquetes es com.sun.j3d.utils, al que normalmente se le conoce como clases de utilidades de Java3D. El paquete de las clases corazón incluye sólo las clases de menor nivel necesarias en la programación Java 3D. Las clases de utilidades son adiciones a las clases corazón y se dividen en cuatro categorías: cargadores de contenidos, ayudas a la construcción del escenario gráfico, clases de geometría y utilidades de conveniencia. Al utilizar las clases de utilidades se reduce significativamente el número de líneas de código en un programa Java3D. Además de las clases de los paquetes corazón y de utilidades, todo programa 3D usa clases de los paquetes java.awt, javax.swing y javax.vecmath. Los

paquetes java.awt y javax.swing definen las clases para crear una ventana donde mostrar el renderizado. El paquete javax.vecmath define clases de vectores matemáticos para puntos, vectores, matrices y otros objetos matemáticos. La versión utilizada de esta API es la 1.5.1.

2.5.2.1. Significado de Renderización

La renderización es el proceso de generar una imagen (imagen en 3D o una animación en 3D) a partir de un modelo a través de una aplicación. El modelo es una descripción en tres dimensiones de objetos en un lenguaje o estructura de datos estrictamente definidos. El modelo debería contener geometría, punto de vista, textura e información de iluminación. De una forma más sencilla, este proceso consiste en todas aquellas tareas que una aplicación de diseño y/o animación 3D lleva a cabo para obtener una imagen o animación final. Se utiliza en la producción de imágenes en 3D para juegos, diseño computacional, efectos especiales del cine y la televisión, etc. En el caso de gráficos en 3D, el renderizado puede hacerse lentamente (pre-rendericado) o en tiempo real. El pre-renderizado es un proceso computacional intensivo que es utilizado generalmente para la creación de películas y su resultado es de altísima calidad. Además, en el prerrenderizado, todos los movimientos y cambios en las escenas en 3D ya fueron prefijados antes del inicio de la renderización. En cambio, el renderizado en tiempo real es más usado en los juegos en 3D y suele procesarse a través de tarjetas aceleradoras de 3D, por ser un proceso sumamente pesado. En este caso, todos los movimientos y cambios en la escena son calculados en tiempo real, pues los movimientos del jugador no son predecibles.

Son millones los cálculos matemáticos que deben realizarse para procesar un modelo en 3D y resultar en una imagen renderizada. En general, en el proceso de cálculo se pueden tener en cuenta tonalidades, texturas, sombras, reflejos, transparencias, translucidez, iluminación (directa, indirecta y global), profundidad de campo, desenfoques por movimiento, ambiente, etc. Además a todo eso hay que agregarle los distintos objetos poligonales en 3D de la escena.

2.5.2.2. Características de Java3D

Las aplicaciones en Java3D construyen los distintos elementos gráficos como objetos separados y los conectan unos con otros mediante una estructura en forma de árbol denominada **grafo de escena** [44]. La aplicación manipula los diferentes objetos utilizando los métodos de acceso, de modificación y de unión definidos en su interfaz. El modelo de programación basado en el grafo de escena de Java3D proporciona un mecanismo sencillo y flexible para representar y renderizar escenas. El grafo de escena contiene una

descripción completa de la escena o universo virtual. Esta descripción incluye datos sobre la geometría, información de los distintos atributos, así como información de visualización.

El sistema de coordenadas que se utiliza en Java3D es el siguiente: la parte positiva del eje de ordenadas (*y*) es el sentido ascendente de la gravedad, la parte positiva del eje de abscisas (*x*) es horizontal hacia la derecha y la parte positiva de eje *z* está dirigido hacia el observador, como se muestra en la figura 2.3:

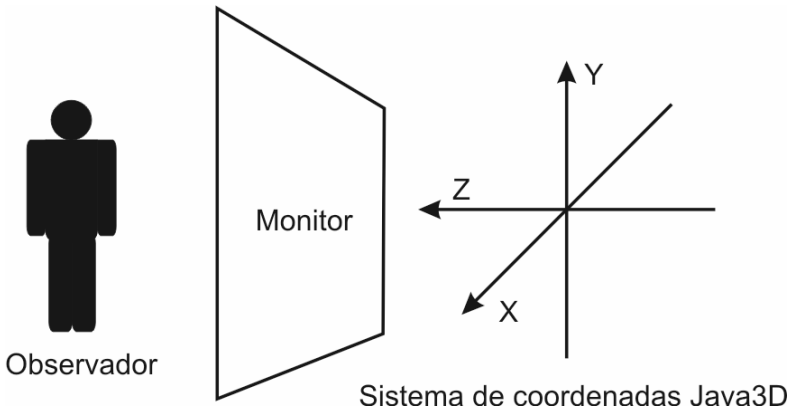


Figura 2.3. Representación del sistema de coordenadas en Java3D

El grafo de escena, como cualquier otro grafo, es una estructura de datos compuesta de nodos y arcos:

- Nodos: un nodo es un elemento de datos. Los nodos del grafo de escena se corresponden con instancias de clases Java3D. A los nodos padre, se les denomina nodos grupo.
- Arcos: un arco es una relación ente elementos de datos (representados por los nodos). Los arcos representan dos tipos de relaciones entre las instancias de Java3D, a saber:

  1. La relación más habitual es la relación padre–hijo. Un nodo grupo puede tener varios hijos, pero sólo un padre. Un nodo hoja puede tener un padre, pero no hijos.

  2. La otra relación es la de referencia. Una referencia asocia un objeto del tipo *NodeComponent* con un nodo del grafo de escena. Los objetos *NodeComponent* definen tanto la geometría como los atributos de apariencia que se utilizan para renderizar los objetos visuales.

Los grafos de escena de Java3D se construyen utilizando objetos *Node* unidos por relaciones padre–hijo formando una estructura de árbol. En una estructura de árbol, un nodo es la raíz. Se puede acceder al resto de los nodos del árbol siguiendo los arcos que

parten del nodo raíz. Un grafo de escena está formado por árboles cuya raíz se sitúa en los objetos *Locale*. Los *NodeComponent* y los arcos de referencia no forman parte, realmente, del árbol del grafo de escena. Existe sólo un camino desde la raíz de un árbol hasta cada una de las hojas y, de igual forma, sólo hay un camino desde la raíz de un grafo de escena hasta cada uno de los nodos hoja. Cada camino de grafo de escena en Java3D especifica completamente la información del estado de su hoja. La información del estado incluye datos como la localización, orientación y tamaño de un objeto visual. Por lo tanto, los atributos de cada objeto visual dependen directamente de su camino de grafo de escena. El renderizador de Java3D utiliza esta propiedad y renderiza las hojas en el orden que él determina que es más eficiente, de forma que el programador no tiene que preocuparse del orden de renderización. En la figura 2.4 se presenta la jerarquía de clases principales del paquete javax.media.j3d.

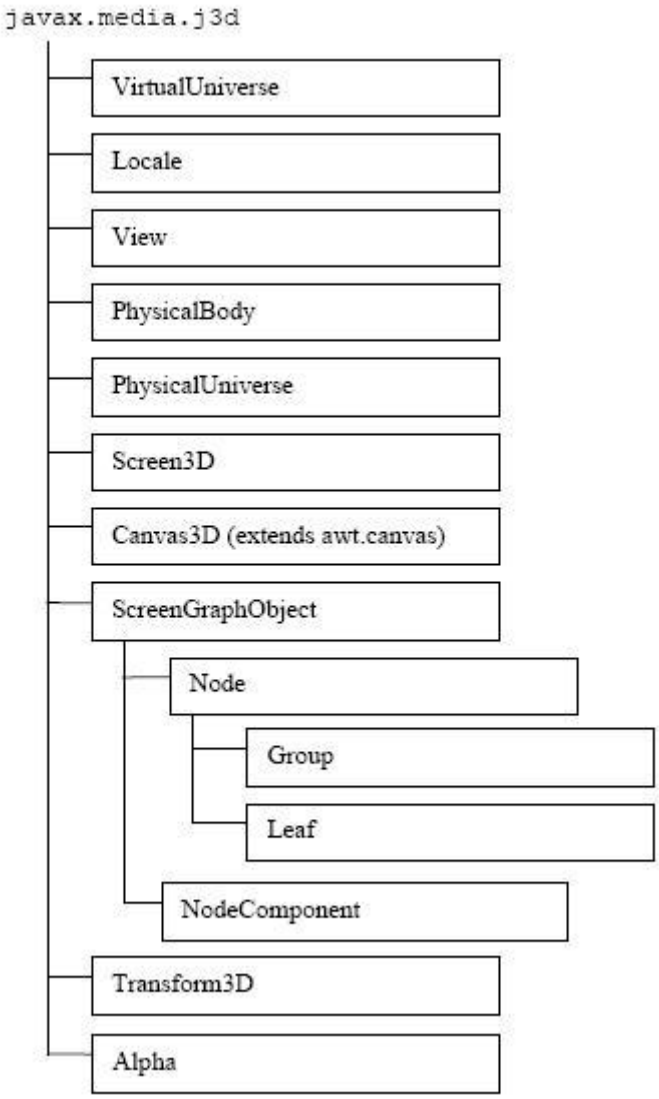


Figura 2.4. Jerarquía de clases del paquete javax.media.j3d

Cada escenario gráfico tiene un sólo *VirtualUniverse*. Este objeto tiene una lista de objetos *Locale*, los cuales proporcionan una referencia a un punto en el universo virtual. Podemos pensar en los objetos *Locale* como marcas de tierra que determinan la localización de los objetos visuales en el universo virtual. Cada objeto *Locale* puede servir de raíz para varios sub-gráficos del escenario gráfico.

SceneGraphObject es la superclase de casi todas las clases corazón y de utilidad de Java 3D y tiene dos subclases: *Node* y *NodeComponent*. Su jerarquía de clases se puede observar en la figura 2.5.

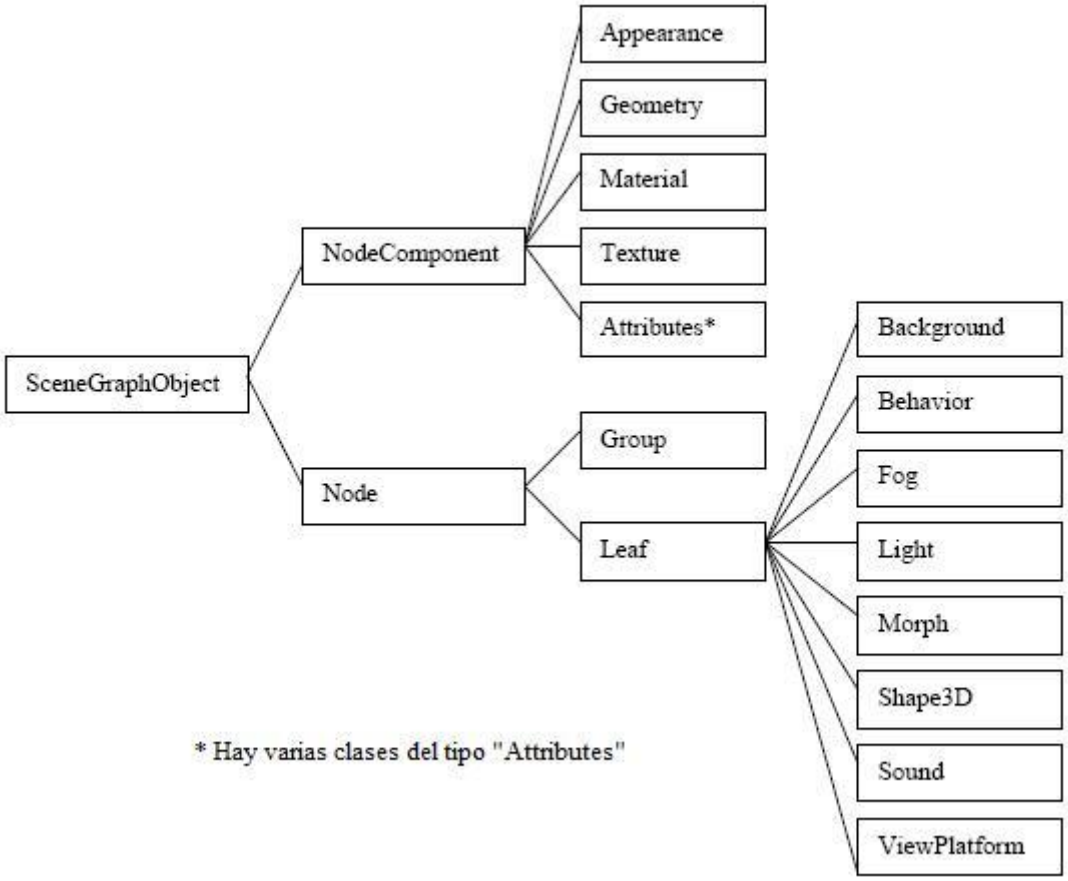


Figura 2.5. Jerarquía de clases que derivan de la clase *SceneGraphObject*

La clase *Node* es la superclase abstracta de las clases *Group* y *Leaf*, que son las que componen los escenarios gráficos, y define algunos de los métodos importantes de sus subclases. La clase *Group* es la superclase usada para la especificación de localización y orientación de objetos visuales en el universo virtual. Dos de las subclases de *Group* son: *BranchGroup* y *TransformGroup*. Los objetos *Group* son nodos de agrupación de propósito general, que tienen un solo padre y un número arbitrario de hijos. Un nodo *BranchGroup* es la raíz de un subgrafo de una escena que puede compilarse como una unidad, unirse a un universo virtual o incluirse como hijo de un nodo de agrupación en otro

subgrafo. Los nodos T*ransformGroup* especifican una transformación espacial sencilla utilizando un objeto Transform3D que puede colocar, orientar y escalar todos sus hijos. La clase *Leaf* es la superclase usada para especificar la forma, el sonido y el comportamiento de los objetos visuales en el universo virtual. Algunas de las subclases de *Leaf* son: *Shape3D*, *ViewPlatform*, *Behavior*, *Light*, y *Sound*. Estos objetos podrían referenciar a *NodeComponents*. La clase *Shape3D* da soporte a la creación de objetos geométricos y contiene dos componentes: una referencia a la forma geométrica y a su componente de apariencia. Los nodos *ViewPlatform* definen una plataforma de visualización que se referencia mediante un objeto *View*. La posición, orientación y escala de las transformaciones desde el grafo de escena hasta el nodo *ViewPlatform* especifican la localización del punto de vista y hacia qué dirección está orientado. Los nodos hoja *Behavior* permiten que una aplicación modifique el grafo de escena en tiempo de ejecución, especificando animaciones o interacción con objetos visuales. Un comportamiento es especificado para un objeto visual y produce cambios en su posición, orientación, color u otros atributos. Los nodos *Light* representan clases abstractas que definen las propiedades de luz comunes a todos los nodos. La clase *NodeComponent* es la superclase usada para especificar la geometría, la apariencia, la textura y las propiedades del material de un nodo *Leaf*.

Existen dos grupos de objetos utilizados en los programas basados en Java3D que se deben tener en cuenta. Por un lado, los objetos que están relacionados con el objeto *ViewPlatform* son: *View*, *Canvas3D*, *Screen3D*, *PhysicalBody* y *PhysicalEnvironment*. El objeto *View* es el objeto principal de la visualización ya que es el que determina toda la información necesaria para generar la escena 3D. *Canvas3D* representa una ventana en la que Java3D dibujará las imágenes, y contiene una referencia a un objeto *Screen3D* e información que describe el tamaño, forma y localización del objeto *Canvas3D* dentro del objeto *Screen3D*. *Screen3D* es un objeto que contiene información relativa a las propiedades físicas de la pantalla. Java3D separa la información relativa a la pantalla en un objeto independiente para evitar la duplicación de información, en caso de que varios objetos *Canvas3D* compartan una sola. *PhysicalBody* es el objeto que contiene información de calibración relativa al cuerpo físico del usuario, y *PhysicalEnvironment* es el objeto que contiene información de calibración del mundo físico. Por otro lado, el segundo grupo de objetos que son los responsables de definir la apariencia (color, textura) y geometría de los nodos *Leaf* son los *NodeComponents*, que no forman parte del escenario gráfico pero son referenciados por dichos nodos *Leaf*.

2.6. NS-2

En la actualidad, NS-2 [11] es una de las implementaciones de simuladores de redes que más importancia ha adquirido en los últimos años. Las dos razones principales que lo han llevado a tal éxito se puede decir que son: abundancia de documentación en línea [45][46], y el hecho de que la distribución posee licencia GPL (*GNU General Public License,* [47]). Entre los usos más habituales que se le puede dar a este tipo de simuladores se encuentran:

- Simular estructuras y protocolos de redes de todo tipo (satélite, inalámbricas, cableadas, etc.)
- Desarrollar nuevos protocolos y algoritmos y comprobar su funcionamiento.
- Comparar distintos protocolos en cuanto a prestaciones.

Se trata de un simulador de tiempo discreto donde el avance de tiempo depende de la temporización de eventos, eventos que son mantenidos por un planificador. Su elaboración se inició en 1989 con el desarrollo de REAL Network Simulator [11]. Inicialmente, NS-2 fue ideado para redes fijas, sin embargo, el grupo o proyecto Monarch de CMU [13] desarrolló una ampliación para el análisis de redes inalámbricas donde se incluyen las principales propuestas de redes *ad hoc* así como de redes WLAN.

NS-2 está basado en dos lenguajes de programación, ya que posee un simulador orientado a objetos escrito en C++, y un intérprete OTcl (extensión orientada a objetos de Tcl) para ejecutar los scripts de usuario. NS-2 contiene una biblioteca rica en objetos para redes y protocolos. Existen dos jerarquías de clases: la jerarquía compilada en C++ y la jerarquía interpretada en OTcl, con una correspondencia de uno a uno entre ellas. La jerarquía compilada en C++ permite alcanzar eficiencia en la simulación y tiempos de ejecución mejores. Esto es particularmente útil para la definición detallada y la operación de los protocolos. El script de usuario en OTcl contiene la topología de red, una serie de protocolos específicos y aplicaciones que se desean simular (cuyo comportamiento ya se encuentra definido en la jerarquía compilada) y la forma en que se quiere obtener la salida del simulador. Un evento es un objeto de la jerarquía en C++ con un identificador único, un tiempo planificado y un puntero al objeto manejador del evento. El planificador guarda una estructura de datos ordenada con los eventos que deben ejecutarse y los va invocando uno a uno a través de la llamada al manejador de cada evento.

CAPÍTULO 3: Estrategias de caché en redes MANET

3.1. SITUACIÓN DE PARTIDA

Las redes MANET están compuestas de dispositivos inalámbricos que se pueden comunicar entre ellos sin ningún enrutador específico. Como se ha visto en el capítulo 1, las tareas de encaminamiento se transfieren a los nodos móviles, de forma que dos nodos distantes entre sí (no directamente conectados a través de un enlace inalámbrico) pueden intercambiar paquetes gracias a la colaboración de los nodos intermedios que retransmiten y enrutan los paquetes hacia el nodo destino. Debido a la naturaleza de los dispositivos que pueden formar parte de una red MANET, se espera que puedan solicitar acceso a Internet en cualquier lugar e instante de tiempo. De esta manera, un miembro de la MANET puede ser un GW (*GateWay*) hacia Internet, a través del cual la red puede acceder a servidores HTTP. Sin embargo, para realizar una conexión a Internet, se deben tener en cuenta una serie de peculiaridades propias de los nodos móviles en una MANET:

- La movilidad de los nodos puede llevar a situaciones donde el GW sea inalcanzable; consecuentemente, el GW no estará permanentemente disponible.
- El medio inalámbrico es compartido y de escaso ancho de banda, por lo que las conexiones a Internet pueden forzar a que el GW sufra cuellos de botella al estar los enlaces sobrecargados con el tráfico entrante y/o saliente.

Por tanto, los esquemas que se planteen para la conexión a Internet en las redes MANET deben considerar las desconexiones temporales del GW, así como la reducción del tráfico de datos y de señalización a través de él. Estos esquemas pueden beneficiarse del concepto del uso de la memoria caché en la web; así, los dispositivos de una MANET almacenarán documentos que previamente hayan sido solicitados a un servidor HTTP. Los nodos móviles podrán beneficiarse de que él mismo u otros nodos de la MANET puedan contener los documentos que soliciten, de forma que la petición no sea cursada al servidor HTTP. Si esto ocurre se reducirán los recursos consumidos en los enlaces hacia el GW. Además, se podrá cubrir el caso en que el GW no se encuentre disponible, ya que cualquier nodo intermedio de la red MANET que contenga un documento solicitado lo podrá servir al nodo solicitante, disminuyendo a su vez el tráfico generado por la red.

En este capítulo se va a realizar un estudio más profundo del esquema de caché planteado en el tema introductorio y que es la necesidad real para este proyecto. Para la simulación de los diferentes escenarios, se ha definido una capa de aplicación que no es más que un clásico protocolo de tipo petición/respuesta. Los nodos móviles envían peticiones similares a los mensajes GET de HTTP [48] que incluyen información acerca del identificador numérico del documento y del instante de expiración de la petición. Si un mensaje GET necesita más tiempo que el marcado por el instante de expiración para alcanzar su destino, el mensaje es descartado. Antes de enviar cada petición, el nodo comprobará si el documento está almacenado en su caché local, tal y como se define en la primera estrategia. Una vez que la petición alcanza su destino, el servidor de datos DS o cualquier nodo intermedio del camino hacia el servidor (dependiendo de las siguientes estrategias de caché) responde con un mensaje RESP, que también es similar al mensaje RESP de HTTP, añadiendo además el tamaño y el instante de expiración del documento. La red *ad hoc* que se va a utilizar para ilustrar un ejemplo de cada uno de los esquemas de caché se muestra en la figura 3.1. En ella se observan líneas entre algunas parejas de nodos representando las conexiones inalámbricas existentes. Aunque el GW aparezca separado del servidor de datos DS (*Data Server*) que almacenará todos los documentos, se va a considerar que se encuentran físicamente en el mismo nodo para simplificar el problema. Los nodos 1, 2, 3 y 4 son nodos de usuario o nodos cliente, y solicitan documentos a DS.

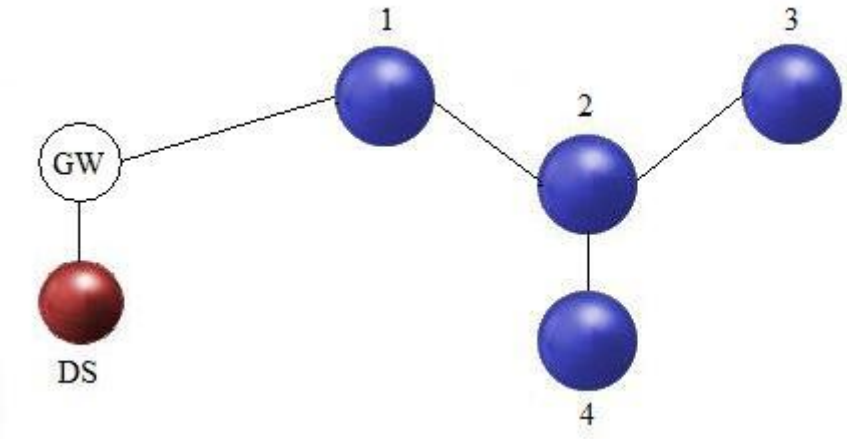


Figura 3.1. Ejemplo de red ad hoc

Se va a representar a continuación un escenario basado en la red planteada en la figura 3.1 sin que se tenga en cuenta ninguna estrategia de las que se verán a continuación, que corresponde al caso típico de petición a un servidor y respuesta. En la figura 3.2 se muestra el intercambio de mensajes generado por una petición de documento entre el nodo 3 y DS a través de los nodos 2 y 1, sin considerar el mecanismo de intercepción. En este caso, da

igual que el nodo 3, el nodo 2, o el nodo 1 tengan o no una copia válida del documento solicitado en su caché local, puesto que la petición se va a enrutar hacia el destino gracias al algoritmo de encaminamiento que esté implementando la red.

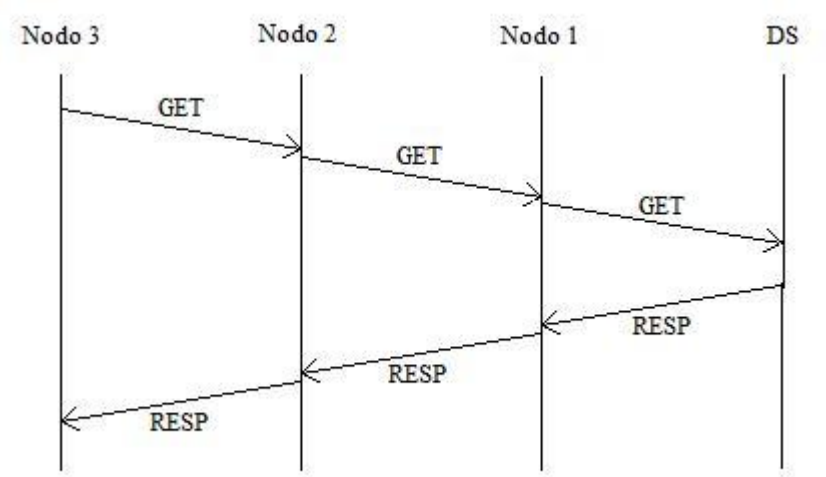


Figura 3.2. Petición-Respuesta sin estrategia de caché

3.2. ESTRATEGIAS DE CACHÉ

3.2.1. CACHÉ LOCAL

Se trata de la forma más simple en que se puede implementar un esquema de caché. Cada nodo va a disponer de una caché local donde almacenará los documentos solicitados con anterioridad, de forma que si el nodo en cuestión necesita de nuevo el mismo documento, éste podrá ser servido directamente desde dicha caché local (siempre que el documento en cuestión fuera válido, es decir, no estuviera obsoleto). A esta situación se le conoce como **acierto en caché local**, y se caracteriza por reducir el tráfico en la red y el retardo en la recepción del documento, así como la energía consumida por los dispositivos inalámbricos.

Existen una serie de parámetros de la caché local a tener en consideración para explicar su funcionamiento. Así, se define un tamaño del espacio de almacenamiento, el tiempo de expiración y la política de reemplazo de los documentos almacenados en caché.

- El espacio reservado para el almacenamiento en cada uno de los nodos puede variar con las características de la red. Es un factor que influye directamente en los

aciertos en la caché, de forma que para espacios de almacenamiento reducidos, la probabilidad de solicitar un documento que se encuentre en caché disminuye.

- Cada documento en caché tendrá un tiempo de expiración a partir del cual la información que contiene queda obsoleta, denominado también Tiempo de Vida o TTL (*Time To Live*). De esta forma, se podrá tener un cierto control de los documentos que deben desecharse de la caché por no contener información válida. Como se puede pensar, en las redes MANET bajo estudio, este tiempo influirá de forma decisiva en los aciertos producidos en la caché local.
- La política de reemplazo representa el algoritmo que se utiliza para determinar qué documentos de la caché local son elegibles para abandonar la caché cuando no hay espacio disponible, dejando espacio para otro nuevo documento. Algunas de las políticas de reemplazo utilizadas son: FIFO (*First In First Out*), donde la entrada de caché a eliminar es la más antigua; LFU (*Last Frequently Used*), en la que se mantiene un contador de visitas de forma que el que tenga el valor más bajo es el elegido para el reemplazamiento; y LRU (*Least Recently Used*), donde se favorecen a las entradas que se consultaron recientemente, de forma que los últimos documentos solicitados tienen más probabilidad de ser requeridos en el futuro. El algoritmo de reemplazo a utilizar en las cachés locales será aquel que sea el más afín al comportamiento del tráfico HTTP de Internet.

Consideremos el ejemplo planteado anteriormente de petición de un documento desde el nodo 3 al servidor de datos DS sobre la red de la figura 3.1. Si el nodo 3 está implementando a nivel de aplicación el esquema de caché aquí presentado, pasaría a buscar en su caché local el documento solicitado antes de enviar hacia el nodo 2 el mensaje GET con el identificador del documento. En caso de acierto en caché local, no se pondría en la red ningún mensaje, pero si ocurre un fallo en caché, el envío de mensajes necesarios sería el mismo que el planteado en la figura 3.2

Se va a presentar a continuación el problema formalmente. Sea $M = \{w_1, w_2, ..., w_w\}$ el conjunto de *w* nodos móviles en una MANET. Se define $U = \{1, 2, ..., n\}$ como el universo de *n* documentos que pueden ser solicitados por los nodos móviles, donde *s(i)* representa el tamaño del documento *i* con $1 \leq i \leq n$. Se define *TTL(i)* como el instante en el que el documento *i* expira y su información se vuelve obsoleta. Se representa $R_j = \{r_{j1}, r_{j2}, ..., r_{jm}\}$ como la secuencia de peticiones realizadas por un nodo *j*, donde r_{jk} denota el documento pedido por el nodo móvil *j* en el instante *k*, teniendo en cuenta que $r_{jk} \in U$. El destino de las

peticiones en R_j es un nodo fijo w_D (servidor de datos) que tiene acceso a todos los documentos de *U*. Se supone que el servidor de datos es accedido a través de un GW. Pasemos a formalizar el comportamiento de la caché local en cada nodo. B_{jk} denota el conjunto de documentos almacenados en la caché del nodo *j* en el instante *k*, donde $B_{jk} \subset U$. El conjunto de documentos almacenados en caché deben satisfacer las ecuaciones (1) y (2):

$$\sum_{i \in B_{jk}} s(i) \leq S_j \quad \textbf{(1)}$$

$$\forall i \in B_{jk} \Rightarrow TTL(i) > k \quad \textbf{(2)}$$

donde S_j representa el tamaño de la caché local del nodo *j*. La propiedad (2) establece que los documentos almacenados en una caché no deben ser obsoletos; por tanto, la secuencia de estados $(B_{j0}, B_{j1}, ..., B_{jm})$ indica los estados por los que pasa la caché del nodo *j* cuando se van resolviendo las peticiones que realiza en R_j. B_{j0} es el estado de la caché inicialmente cuando está vacía y B_{jm} es el estado de la caché cuando todas las peticiones que realiza el nodo *j* han sido servidas.

Se define $p_{ij} = \{w_i, w_z, w_{z+1}, ..., w_j\}$ como la ruta activa entre el nodo *i* y el nodo *j*, formada por el conjunto de nodos que son necesarios para alcanzar el nodo *j* desde el nodo *i*. Solo dos nodos consecutivos en un camino w_z, w_{z+1} están directamente conectados, es decir, existe un enlace inalámbrico entre ellos. Si $p_{ij} = \varnothing$ entonces no existe ruta creada para alcanzar el nodo *j* desde el nodo *i*; en otro caso, $card(p_{ij}) \geq 2$ (la cardinalidad del conjunto p_{ij}), ya que p_{ij} contendrá al menos a los nodos *i* y *j*. Se puede definir la distancia entre el nodo *i* y *j* como (3):

$$dist(w_i, w_j) = card(p_{ij}) - 1 \quad \textbf{(3)}$$

Esta distancia representa el número de saltos necesarios en la red multisalto para alcanzar al nodo *j* desde el nodo *i*. Cuando la distancia entre ambos nodos origen y destino es 1, dichos nodos son vecinos, puesto que se encuentran a un salto. En el otro extremo, se considera $dist(w_i, w_j) = \infty$ si no es posible crear una ruta entre los nodos *i* y *j*.

Se define la secuencia de aciertos en caché local en el nodo *j* como $(lh_{j1}, lh_{j2}, ..., lh_{jm})$, donde lh_{jm} representa:

$$lh_{jm} = \begin{cases} 1 & \text{si } r_{jk} \in B_{j(k-1)} \\ 0 & \text{si } r_{jk} \notin B_{j(k-1)} \end{cases} \quad (4)$$

Cuando $lh_{jm}=1$, se produce un acierto en caché en el nodo j; en otro caso, se produce un error en caché puesto que el documento solicitado por el nodo propio nodo j no se encuentra en su caché local en dicho instante.

Dada una secuencia de peticiones R_j en el nodo j, una caché local de tamaño S_j y un estado inicial de caché B_{j0}, entonces la política de reemplazo producirá una secuencia de estados de caché de la forma $(B_{j1},...,B_{jm})$. Para $k=1,..,m$, se tiene la ecuación (5)

$$B_{jk} = \begin{cases} (B_{j(k-1)} - \varepsilon_{jk}) \cup \{r_{jk}\} & \text{si } r_{jk} \notin B_{j(k-1)} \\ B_{j(k-1)} & \textit{en otro caso} \end{cases} \quad (5)$$

En la primera opción considerada en la ecuación anterior, el documento solicitado no se encuentra previamente almacenado en la caché del nodo j, causando un fallo de caché. El término $\varepsilon_{jk} \subset B_{j(k-1)}$ indica el conjunto de documentos que deben ser eliminados de la caché para hacer espacio para el nuevo documento solicitado r_{jk}. Si hay espacio en caché suficiente para el nuevo documento, el término ε_{jk} estará vacío. En el segundo caso, se produce un acierto en caché local y dicha caché permanecerá inalterada.

3.2.2. INTERCEPCIÓN DE PETICIONES

En esta aproximación, la caché local de cada nodo tendrá las funcionalidades de un *proxy*, de forma que cada nodo intermedio que se encuentre en la ruta seguida por una petición desde el nodo origen hasta el servidor de datos podrá responder a la petición si tiene una copia válida del documento solicitado en su caché local. Esta estrategia parece muy acertada si se considera que un grupo de nodos, como podrían ser los usuarios de una empresa o una universidad, comparten intereses afines y por tanto, podrían solicitar documentos similares. Esta intercepción de la petición reduce la latencia percibida por los usuarios ya que el documento es servido por un nodo intermedio, que se encuentra más cerca del nodo solicitante que el propio servidor. Además, también se decrementa el tráfico en los enlaces hacia Internet así como la carga de los servidores remotos.

Para mostrar un ejemplo del mecanismo de caché aquí planteado, nos vamos a basar de nuevo en la red *ad hoc* mostrada en la figura 3.1. Imaginemos que el nodo 2 quiere

solicitar el documento A a DS. Esta petición se recibe en el nodo 1 y es retransmitida por el mismo hacia el servidor de datos, según el protocolo de encaminamiento que esté implementando la red. DS responderá con el documento utilizando la misma ruta desde el nodo 1 al 2. Al recibir el nodo 2 el documento A, lo almacenará en su caché local. Supongamos que el nodo 3 solicita el mismo documento A al servidor de datos y existe una ruta válida entre el nodo 3 y el servidor DS. Cuando un nodo intermedio de la ruta hacia DS recibe el mensaje GET y no es el destino de la petición, mientras este mensaje se está procesando a nivel AODV (o a nivel del protocolo de encaminamiento utilizado), se consulta a la capa de aplicación si existe una copia válida del documento solicitado en la caché local. Si se da el caso, la capa AODV será informada de que el nodo va a responder directamente a la petición, y que no debe retransmitir la misma. De esta forma, la petición llegaría al nodo 2, que comprobaría si tiene dicho documento en caché y es válido. Si es así, respondería con una copia del documento A. Con este mecanismo de intercepción de la petición, se reduce el número de saltos de seis (3-2-1-DS-1-2-3) a dos (3-2-3). La reducción de número de saltos implica que se reduce tanto la latencia que percibe el nodo 3 como el gasto energético de los nodos. Esto implica además que disminuye el tráfico en los nodos que forman parte del camino hacia el servidor de datos puesto que se reduce notablemente el número de retransmisiones de las peticiones y respuestas, así como la carga de los enlaces hacia el GW, ayudando a evitar posibles cuellos de botella. En la figura 3.3 se observa la transferencia de mensajes producidos entre los nodos 3 y 2 a causa de la intercepción en el nodo 2.

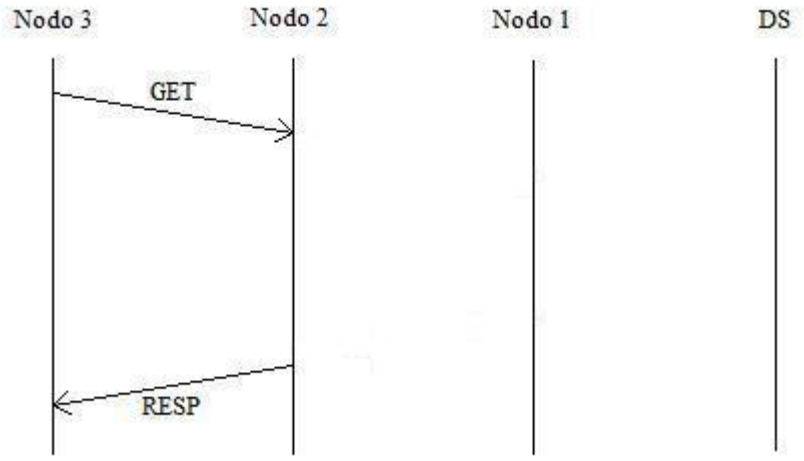


Figura 3.3. Petición-Respuesta con intercepción del mensaje GET

A la situación en la que un nodo en la ruta de la petición hacia el servidor DS intercepta una petición se le llama **acierto de intercepción en caché**. Para diferenciarlo del otro

mecanismo de intercepción que se describirá más adelante, en el visualizador desarrollado se le denotará como acierto de intercepción en caché remota.

Se define formalmente la secuencia de aciertos de intercepción en caché del nodo *j* como $(ih_{j1},ih_{j2},...,ih_{jm})$ donde ih_{jk} está definido en (6):

$$ih_{jk} = \begin{cases} 1 & \text{si } r_{jk} \notin B_{j(k-1)} \wedge r_{jk} \in B_{i(k-1)} \mid i \neq j \wedge i \neq DS \wedge w_i \in p_{jDS} \\ 0 & \textit{en otro caso} \end{cases} \quad (\mathbf{6})$$

donde w_i representa un nodo intermedio en la ruta de la petición r_{jk} desde el nodo *j* a DS que además tiene una copia válida del documento solicitado en su caché local. Cuando $ih_{jm}=1$ entonces se considera acierto de intercepción de caché en el nodo *j*. Como la distancia $dist(w_j,w_i)$ es necesariamente menor a la distancia $dist(w_j,w_{DS})$, la cantidad de saltos necesarios para satisfacer la petición disminuye, así como el tiempo para servir la petición.

3.2.3. REDIRECCIÓN DE PETICIONES

Se extiende el planteamiento anterior de intercepción de peticiones de forma que se tenga en cuenta la distribución de los documentos en la MANET, reduciendo así la longitud de las rutas hacia el nodo que sirve las peticiones. Más concretamente, se puede conseguir que los nodos almacenen información acerca de la distancia (en número de saltos) hasta un nodo que contenga un determinado documento en su caché local. De esta forma, cuando un nodo sea nodo intermedio en el camino de una petición de un documento, podrá comprobar si ese documento se encuentra en un nodo más cercano que el destino de la petición, es decir, el servidor de datos. Esta información se puede extraer dinámicamente de los propios mensajes retransmitidos por cada nodo (tanto peticiones como respuestas).

Veamos un caso práctico sobre la red planteada en la figura 3.1. Imaginemos que el nodo 4 solicita un documento A al nodo DS. La petición pasará a través de los nodos 2 y 1 hasta DS, de forma que el nodo 2 sabrá que el nodo 4 tendrá el documento A y que se encuentra a un salto. Análogamente, el nodo 1 tendrá conocimiento de que en el nodo 4 se podrá encontrar el documento A y que éste se encuentra a dos saltos. Cuando el servidor de datos responda al nodo 4 con el documento pedido a través de los nodos 1 y 2, estos nodos 1 y 2 sabrán que el documento A se encuentra en DS, a 1 y 2 saltos respectivamente. Si se diera el caso de que el nodo 3 solicitara el mismo documento A al nodo DS, dicha petición

se encaminaría hacia el nodo 2, que tiene almacenada información acerca de ese documento. Así, una vez recibida la petición en el nodo 2, podrá comprobar que tanto el nodo 4 y el nodo DS tienen el documento A, y que se encuentran a uno y dos saltos respectivamente. El nodo 2 podrá redireccionar la petición hacia el nodo 4 que es el que define la ruta más corta. Esta redirección reduce la cantidad de saltos de seis (3-2-1-DS-1-2-3) a cuatro (3-2-4-2-3). El intercambio de mensajes producido en esta situación viene representado en la figura 3.4. Cuando la redirección se realiza correctamente, se denomina **acierto de redirección**. Las ventajas que supone esta redirección son la reducción tanto de la latencia percibida por el nodo 3, como del tráfico generado en la red, ya que la cantidad de mensajes generados por la retransmisión de las peticiones y respuestas se reduce, pudiendo disminuir los posibles cuellos de botella en el GW.

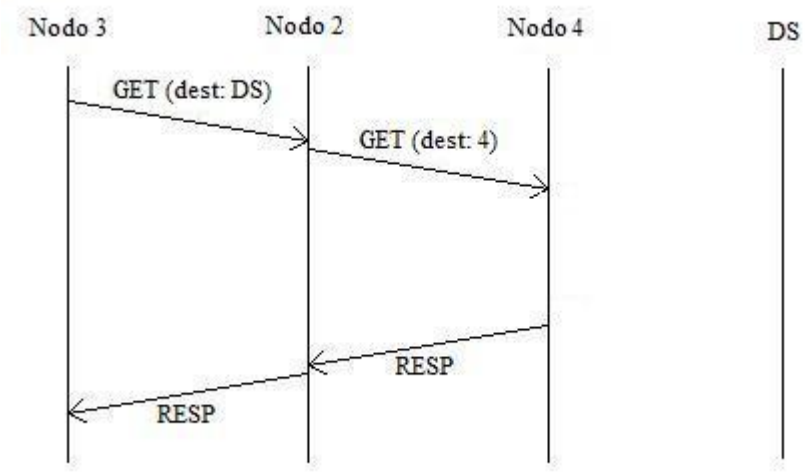


Figura 3.4. Petición-Respuesta con redirección del mensaje GET

Un nuevo ejemplo de aplicación de la redirección de peticiones contempla la posibilidad de que se realice en el nodo que origina la petición, es decir, redireccionar desde el origen y no únicamente en la ruta de la petición. Si se supone la misma situación anterior, si el nodo 2 quiere solicitar el documento A, este nodo puede redireccionar la petición al nodo 4 (a un salto) en lugar de hacia DS (a dos saltos), ya que posee la información suficiente acerca de la distancia en saltos a la que se encuentra el documento para tomar esa decisión.

Desafortunadamente, la redirección de peticiones tiene algunos problemas que deben tenerse en cuenta: la movilidad de los nodos, la desconexión de los nodos y el reemplazo de los documentos en las cachés locales.

- La movilidad y la desconexión de los nodos en las redes *ad hoc* puede ocasionar que la información que mantienen los nodos acerca de los documentos diseminados en la red y las distancias a las que se encuentran éstos pueda dejar de ser válida al romperse los enlaces inalámbricos existentes. Por ello, se implementa un sistema de alarma que indique al nodo que ha realizado la petición que se ha producido un error en la redirección, pudiendo solicitar de nuevo el documento puesto que no ha sido servido satisfactoriamente. Este **error de redirección** se tiene en cuenta a la hora de visualizar los paquetes en la red, y es un parámetro estadístico bajo estudio.
- El reemplazo de los documentos en las cachés locales puede producir la misma situación descrita anteriormente, de forma que, al redireccionar una petición, es posible que un documento no pueda ser servido porque ya no se encuentra en caché a causa de la política de reemplazo que se haya implementado. Por ejemplo, si se considera el ejemplo anteriormente expuesto de la red de la figura 3.1, si el nodo 3 solicita el documento A y el nodo 2 decide redireccionar la petición hacia el nodo 4, pero ha sucedido que la política de reemplazo del nodo 4 ha decidido eliminar dicho documento en cuestión, entonces se producirá un error de redirección. Una posible solución a este problema es que el nodo 4, al recibir la petición del documento A, le pueda indicar de alguna manera con un mensaje especial al nodo 2 que ya no tiene el documento. Con este mensaje especial, todos los nodos que se encontrasen en la ruta entre el nodo 4 y 3 podrían refrescar la información acerca de la caché del nodo 4 para que actualizaran que el nodo 4 ya no tiene el documento A. De esta forma, el nodo 3 podrá solicitar de nuevo el documento A, produciendo un total de diez saltos (3-2-4-2-3-2-1-DS-1-2-3), respecto a los seis saltos que habrían sido necesarios (3-2-1-DS-1-2-3) si no se hubiera implementado el mecanismo de redirección.

Con el fin de reducir el número de errores de redirección ocurridos por las situaciones planteadas anteriormente, se propone utilizar más datos para conocer la expiración de la situación de los documentos en la red MANET. Para este fin, a la información utilizada para realizar la redirección se añadirá un tiempo de expiración de dicha información, que coincidirá con el valor del TTL de los documentos.

Finalmente, este método para el almacenamiento de la información en un nodo sobre la situación de los documentos en otras cachés podría ampliarse si se considera el modo promiscuo de los nodos en la red *ad hoc*. Así, los nodos podrían tener en cuenta no sólo la

información acerca de los documentos que ellos retransmiten por las peticiones y respuestas, sino también procesando los paquetes que retransmiten sus nodos vecinos.

Definamos formalmente el mecanismo de redirección. Para ello, hay que introducir el concepto de caché de redirecciones, que se encargará de almacenar información acerca de la distancia en saltos a la que se encuentran determinados documentos del universo *U*. Para esta caché de redirecciones no se ha definido una política de reemplazo, por lo que la información que contiene podría ser obsoleta. Los diferentes estados que puede presentar vendrán definidos por las actualizaciones de la información que contienen sus entradas cuando el nodo reciba nuevos datos para esas entradas. Con esta información, un nodo podrá ser capaz de redireccionar una petición hacia un nodo de forma que el número de saltos se reduzca. Vamos a denominar BR_{jk} a la información almacenada en el nodo *j* en el instante *k* acerca de la situación de ciertos documentos del universo *U* en la red, como la distancia en saltos, el nodo que lo contiene, y el TTL. BR_{j0} es el estado de la caché de redirecciones inicialmente cuando está vacía, e irá modificando la información que contiene a medida que los nodos forman parte de las retransmisiones debidas a peticiones de documentos de otros nodos. Incluso si el nodo se encuentra en modo promiscuo, podrá añadir información acerca de las retransmisiones de peticiones de documentos que realicen los nodos en su radio de cobertura, es decir, sus nodos vecinos. Se define formalmente la secuencia de aciertos de redirección en el nodo *j* como $(rh_{j1}, rh_{j2}, ..., rh_{jm})$ donde rh_{jk} está definido en (7):

$$rh_{jk} = \begin{cases} \text{si } r_{jk} \notin B_{j(k-1)} \wedge r_{jk} \notin B_{i(k-1)} \wedge r_{jk} \in BR_{i(k-1)} \wedge r_{jk} \in B_{r(k-1)} \mid \\ 1 \qquad\qquad i \neq j \neq r \wedge i \neq DS \wedge\ r \neq DS \\ \qquad\qquad \wedge w_i \in p_{jDS}\ \wedge \exists p_{ir}\ \wedge \exists p_{rj}\ \wedge\ dist(w_i, w_r) < dist(w_i, w_{DS}) \\ 0 \quad \textit{en otro caso} \end{cases} \tag{7}$$

donde w_i representa un nodo intermedio en la ruta de la petición r_{jk} desde el nodo *j* a DS que además no tiene una copia válida del documento solicitado en su caché local, y que tiene información acerca de que el documento solicitado se encuentra en el nodo *r*, redireccionando la petición hacia él. La redirección se va a producir tanto si el nodo *r* se encuentra en la ruta del nodo *j* a *DS* como si no, siempre que se pueda establecer con éxito una ruta entre los nodos *i* y *r*, y además cuando el servidor de datos *DS* se encuentre a menor distancia del nodo *i* que el nodo *r*, o lo que es lo mismo, cuando $dist(w_i, w_r)$ sea inferior a $dist(w_i, w_{DS})$. Además, deberá poder crearse una ruta (si no existía) desde el nodo

r hasta el nodo *j* para que pueda enviar la respuesta. Cuando $rh_{jm}=1$ entonces se considera **acierto de redirección** en el nodo *j*.

3.2.4. INTERCEPCIÓN DE PETICIONES CON RECURSOS DE ENCAMINAMIENTO

El procedimiento de intercepción de caché presentado en el apartado 3.2.2 funciona en el caso de que entre el nodo que solicita el documento y el destino exista un camino establecido. Pero imaginemos la situación que se presenta en la figura 3.5 en la que aparece una red *ad hoc* aislada. Es ese caso, el nodo 1 se ha salido fuera del área de cobertura del nodo DS (que en el ejemplo planteado anteriormente contenía también a GW para mayor facilidad), y por tanto, el nodo 2 ahora no puede establecer ruta hacia él. Como se observa, los nodos 3 y 4 se encuentran dentro de la zona de cobertura del nodo 2, y cuando el nodo 4 realiza una petición del documento A a DS, detecta que no existe camino establecido hacia el GW puesto que el protocolo de encaminamiento no ha encontrado la forma de alcanzar el servidor satisfactoriamente. En este caso, la petición no podrá ser servida incluso sabiendo que el nodo 2 tiene una copia válida del documento A en su caché local, ya que la búsqueda de ruta es el primer paso a realizar.

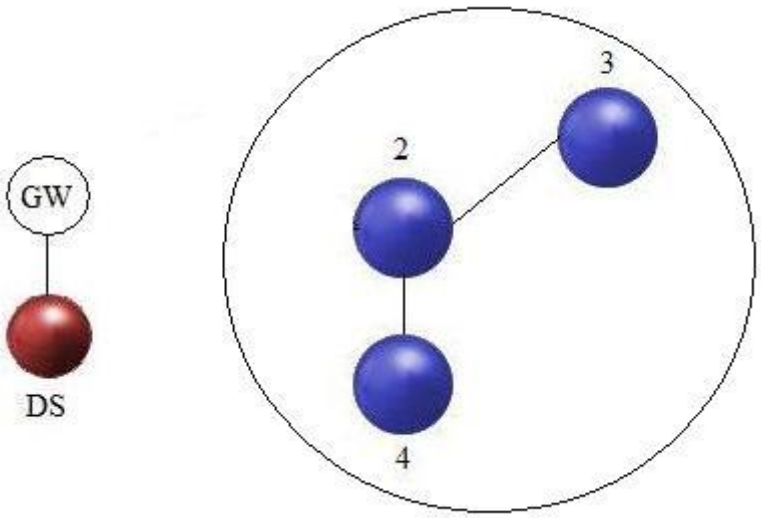


Figura 3.5. Red MANET aislada

Para evitar este problema se ha propuesto el denominado *Cross-layer Interception Caching*, es decir, un esquema de caché para la intercepción de documentos con compartición de información entre las diferentes capas. El concepto *cross-layer* [49] está actualmente muy extendido, sobre todo con la aparición de redes inalámbricas, debido a las particularidades del canal inalámbrico.

3.2.4.1. Concepto de *cross-layer*

Tradicionalmente, los protocolos de red se han diseñado en base a modelos que dividen las tareas de la red en capas, de forma que cada capa es diseñada y operada de manera independiente, con interfaces entre capas que son estáticas e independientes. Así, el modelo OSI (*Open System Interconnection*, [50]), divide las tareas de la red en capas y define una arquitectura de servicios a ser provista por capas individuales. Esto permite explotar la ventaja de modularidad en el diseño de un sistema. La arquitectura prohíbe la comunicación directa entre capas no adyacentes; y la comunicación entre capas adyacentes está limitada a procedimientos de llamadas y respuestas. En cambio, en los sistemas dinámicos la interacción de protocolos de diferentes capas puede llegar a ser muy complejo debido a la existencia de parámetros y a la naturaleza no lineal de los protocolos en las diferentes capas. Gracias a la explotación cuidadosa de la interacción de protocolos *cross-layer,* se puede llegar a hacer más eficiente el desempeño de los protocolos de transmisión en diferentes escenarios de redes inalámbricas.

Los diseñadores tienen dos opciones a la hora de diseñar protocolos. En primer lugar, los protocolos pueden ser diseñados respetando las reglas de la arquitectura de referencia. En la arquitectura de capas, esto significa que los protocolos de capas superiores únicamente hacen uso de los servicios de capas inferiores y no se preocupan por los detalles de cómo los servicios están siendo provistos. Sin embargo, el diseño de *cross-layer* se refiere a protocolos diseñados para explotar la dependencia entre las capas para obtener ganancias en desempeño. Así, estos protocolos violan la arquitectura de referencia permitiendo, por ejemplo, la comunicación directa entre protocolos de capas no adyacentes, o compartiendo variables entre capas. Tales violaciones son realizadas en *cross-layer*.

3.2.4.2. Planteamiento del mecanismo *cross-layer*

En este mecanismo se propone incluir cierta información en los mensajes de encaminamiento generados al crear la ruta hasta el nodo servidor de datos. De esta forma, cuando un nodo solicita un documento, se debe crear en primer lugar una ruta hasta el servidor de datos. Los nodos que reciban dichos mensajes de enrutamiento pueden responder informando si tienen una copia válida del documento buscado. Se ha propuesto involucrar al algoritmo de enrutado en el proceso de búsqueda de documentos en la red MANET.

Supongamos que se utiliza el protocolo AODV para la búsqueda de rutas. Cuando un nodo solicita un documento a DS y no existe una copia válida del mismo en su caché local ni una ruta válida hacia el servidor de datos, el nodo solicitará la creación de una ruta enviando un mensaje RREQ hacia todos sus nodos vecinos con la finalidad de encontrar una ruta hacia un nodo destino, en nuestro caso, el GW. En este mensaje RREQ se podría añadir cierta información acerca del documento requerido, como su identificador. Considerando este mecanismo de intercepción, si un nodo intermedio en la ruta hacia el servidor DS recibe la petición RREQ y tiene una copia válida (no obsoleta) del documento que viene indicado en dicho mensaje RREQ, entonces este nodo responderá con un RREP que contenga la identificación del documento. Si el primer mensaje RREP que recibe el nodo que ha solicitado el documento contiene la información añadida de dicho documento, la petición GET será enviada hacia el nodo que envió la respuesta RREP. En otro caso, la petición GET será enviada al servidor DS. En el ejemplo de red MANET aislada planteado en la figura 3.5, supongamos que el nodo 4 solicita el documento A. En ese caso, su capa de aplicación enviará un mensaje GET hacia la capa AODV, que a su vez transmitirá a todos sus nodos vecinos un mensaje RREQ con la identificación del documento A para poder crear una ruta hacia el nodo destino, en este caso DS. Cuando el nodo 2 reciba el mensaje RREQ, verificará si lleva algún tipo de información del documento solicitado en él y, si es así, extraerá dicha información para poder buscarlo en su caché local. Si el nodo 2 tiene una copia válida (que no haya expirado) del documento solicitado, será capaz de responder al nodo 4 utilizando el mensaje RREP de AODV e incluyendo la información del documento solicitado. De esta forma, el nodo 4 recibirá el mensaje RREP con la información acerca del documento solicitado y la capa de aplicación será informada del nuevo destino de la petición. Entonces se habrá creado una ruta entre los nodos 2 y 4, pudiendo pedir el documento directamente a través del mensaje GET desde el nodo 4 al nodo 2. El intercambio de mensajes generado en esta situación está representado en la figura 3.6. En ella se puede observar que para el caso del nodo 4 se presentan las capas de aplicación y AODV, mostrando la interacción entre capas que es propia de este mecanismo *cross-layer*. Cuando el nodo 4 reciba el mensaje RREP, indicará a su capa de aplicación que el nuevo destino de la petición a cursar deberá ser el nodo 2, y no el nodo servidor de datos DS, destino inicial para el que se estaba buscando la ruta.

Cuando este mecanismo se realiza satisfactoriamente, se denomina **acierto de intercepción en caché con recursos de encaminamiento**.

Utilizando este procedimiento, los nodos en una red MANET podrían acceder a cualquiera de los documentos que estuviesen disponibles en toda esa red, incluso si el servidor de datos se encontrase inaccesible.

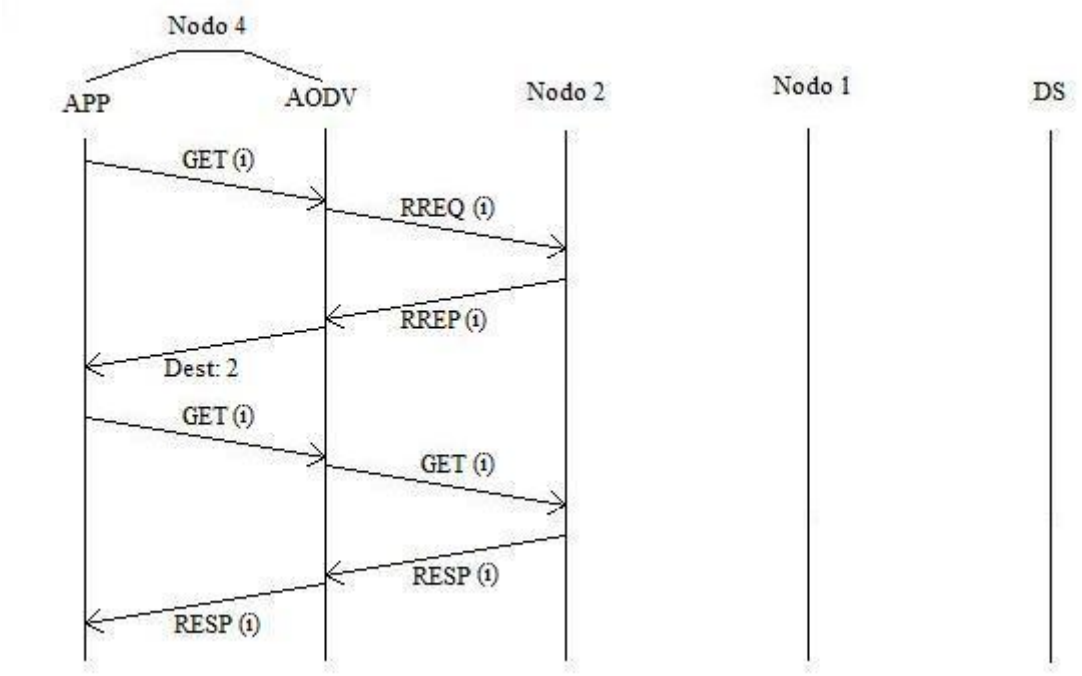


Figura 3.6. Petición-Respuesta con intercepción con recursos de encaminamiento

Se define la secuencia de aciertos de intercepción en caché en la creación de la ruta solicitada por el nodo *j* y resuelta en el nodo *i* como $(rpih_{j1}, rpih_{j2}, ..., rpih_{jm})$, donde $rpih_{jm}$ está definida en (8).

$$rpih_{jk} = \begin{cases} 1 & \text{si } r_{jk} \notin B_{j(k-1)} \wedge r_{jk} \in B_{i(k-1)} \mid i \neq j \wedge i \neq DS \wedge p_{jDS} = \varnothing \\ 0 & \textit{en otro caso} \end{cases} \quad \textbf{(8)}$$

La variable *i* indica el nodo inalámbrico que responde a la petición (cuando recibe el mensaje RREQ) de una ruta desde el nodo *j* al nodo servidor DS porque tiene una copia válida del documento solicitado en su caché local. Cuando $rpih_{jm}=1$ significa que se ha producido un acierto de intercepción en caché durante la creación de la ruta en el nodo *j*. De nuevo, como necesariamente $dist(w_j, w_i)$ es inferior que $dist(w_j, w_{DS})$, entonces la cantidad de saltos necesarios para satisfacer la petición es menor, así como el retardo que percibe el nodo *j* para obtener el documento. Además, el documento podría ser servido incluso si no existe ruta entre el nodo *j* y el servidor DS, es decir, si $dist(w_j, w_{DS}) = \infty$.

CAPÍTULO 4: Fases de diseño software del proyecto

4.1. INTRODUCCIÓN A LA INGENIERÍA DEL SOFTWARE

La Ingeniería del Software es una disciplina de la ingeniería que engloba a todos los aspectos de la producción de software, incluyendo los procesos técnicos del desarrollo de software, gestión de los proyectos software y el desarrollo de herramientas, métodos y teorías en las que se apoya la producción de software, desde las etapas iniciales de la especificación del sistema, hasta el mantenimiento de éste durante su utilización. Es importante destacar que la Ingeniería del Software y la ciencia de la computación son conceptos diferentes. Así, la ciencia de la computación comprende la teoría y los fundamentos en los que se basa la Ingeniería del Software, y la Ingeniería del Software comprende los aspectos prácticos para desarrollar y entregar un software útil. Además, es importante conocer en qué se diferencia el concepto de ingeniería de sistemas del de Ingeniería del Software anteriormente presentado. El concepto de ingeniería de sistemas es muy amplio e incluye todos los aspectos del desarrollo de sistemas informáticos, incluyendo el hardware, el software y la ingeniería de procesos, mientras que la Ingeniería del Software no es más que una parte de esa ingeniería de sistemas. Un buen software debe tener la funcionalidad y el rendimiento requeridos por el usuario, además de permitir su mantenimiento, y ser confiable y fácil de utilizar:

- Mantenibilidad: El software debe escribirse de tal forma que pueda evolucionar para cumplir las necesidades de cambio de los clientes. Éste es un aspecto crítico debido a que el cambio en el software es una consecuencia inevitable de un cambio de necesidades por parte del usuario final.
- Confiabilidad: La confiabilidad del software tiene un gran número de características, entre las que destacan la fiabilidad, la protección, y la seguridad. Un software confiable no debe causar daños económicos o físicos en el caso de que se produzca un fallo del sistema.
- Eficiencia: El software no debe hacer que se malgasten los recursos del sistema, como la memoria y la unidad de procesamiento. Por lo tanto, la eficiencia incluye

optimizar parámetros como tiempos de respuesta y de procesamiento, utilización de la memoria, etc.

- Usabilidad: El software debe ser fácil de utilizar, sin esfuerzo adicional, por el usuario para quien está diseñado. Esto significa que debe tener una interfaz de usuario apropiada y una documentación adecuada.

Un proceso del software es un conjunto de actividades y resultados asociados que generan un producto de software. Existen cuatro actividades fundamentales de los procesos que son comunes, que son:

1. Especificación del software, donde los clientes e ingenieros definen el software a producir y las restricciones sobre su operación.
2. Desarrollo del software, donde el software se diseña e implementa, cumpliendo con la especificación.
3. Validación del software, donde se comprueba el software para asegurar que realiza lo que el cliente quiere, cumpliendo los requisitos.
4. Evolución del software, donde el software se modifica para adaptarlo a los cambios requeridos por el cliente y por el mercado.

De las fases genéricas del proceso de desarrollo del software, nos vamos a centrar en este capítulo en la fase de definición y la fase de desarrollo. La fase de pruebas del software se estudiará en el capítulo cinco.

4.2. FASE DE DEFINICIÓN DEL SOFTWARE

4.2.1. CARACTERÍSTICAS

Dentro de esta fase se dan varias tareas principales, como la planificación del proyecto software y el análisis de requisitos, que será el objeto de nuestro estudio. Los requisitos para un sistema son los servicios que el cliente requiere y las restricciones bajo las cuales el sistema debe operar, es decir, las restricciones operativas. El proceso de descubrir, analizar, documentar y verificar estos servicios y restricciones se denomina Ingeniería de Requisitos (IR). Existen prácticas recomendadas para la especificación de los requisitos del software que se debe desarrollar y sobre la redacción del documento de requisitos, como la recomendación recogida en el estándar IEEE/ANSI 830-1993 (*Institute of Electrical and Electronics Engineers/American National Standards Institute*). El documento de requisitos

es la declaración oficial de qué es lo que se le pide a los desarrolladores, sirviendo de contrato entre el cliente y la empresa que desarrolla el software, e incluye tanto los requisitos como una especificación del sistema, sin ser un documento de diseño. En el estándar IEEE/ANSI 830-1993 se desarrollan una serie de pautas y consejos para realizar una buena especificación de los requisitos software, indicando qué debería contener dicha especificación, así como los parámetros para definir la calidad de la misma.

4.2.2. ANÁLISIS DE REQUISITOS

El proceso de análisis de requisitos consiste en descubrir, analizar y validar los requisitos del sistema, y realizar su correcta gestión. La obtención y análisis de los requisitos implica que el personal técnico trabaje con los clientes para descubrir el dominio de la aplicación, los servicios que el sistema debería proporcionar y las restricciones operacionales del sistema. Se utiliza el término *stakeholders* para referirse a todos los participantes en este proceso. El análisis de requisitos tiene una perspectiva múltiple, porque los participantes representan los diferentes puntos de vista, lo que facilita un estudio completo del problema. Para cada uno de los puntos de vista se crearán una serie de escenarios que darán lugar, posteriormente, a los denominados casos de uso, que se verán con más detalle más adelante. La validación de requisitos consiste en responder a preguntas como: ¿proporciona el sistema las funciones que mejor soportan las necesidades del cliente?, ¿hay algún conflicto entre requisitos?, ¿están incluidas todas las funciones requeridas por el usuario? y ¿se pueden implementar los requisitos según el presupuesto y la tecnología disponible? El coste de los errores en los requisitos es alto, ya que arreglar un error de los requisitos después de la entrega puede ocasionar un coste varias veces superior que el generado por arreglar un error de implementación. Por esta razón esta fase de validación es importante. Por último, es necesario gestionar los cambios de los requisitos durante las fases de diseño y desarrollo del sistema, puesto que, inevitablemente, los requisitos son incompletos e inconsistentes por naturaleza y emergen nuevos requisitos o se modifican los planteados cuando cambian las necesidades del negocio.

Los requisitos deberían redactarse en lenguaje natural debido a que tienen que ser comprendidos por personas que no son técnicos expertos. Sin embargo, se pueden expresar requisitos del sistema más detallados de forma más técnica. Una técnica ampliamente usada es documentar la especificación del sistema como un conjunto de modelos del sistema. Estos modelos son representaciones gráficas que describen el problema a resolver y el sistema que debe ser desarrollado, y son claves para las fases de diseño y de desarrollo

del sistema. En los siguientes apartados se dará importancia al modelado del sistema y de su contexto.

Un requisito puede abarcar desde una declaración abstracta de alto nivel de un servicio que ofrece la aplicación hasta la especificación formal de una función del sistema. Algunos de los problemas que surgen durante el proceso de IR son debidos a no haber realizado una clara separación entre estos diferentes niveles de descripción. Los requisitos se pueden clasificar en dos grandes grupos:

- Requisitos de usuario: Declaraciones en lenguaje natural y en forma de diagramas de los servicios que el sistema debe proporcionar al usuario y de las restricciones bajo las que debe funcionar. Están escritos por los clientes. Únicamente deben especificar el comportamiento externo del sistema y, deben evitar, en la medida de lo posible, las características de diseño del sistema. Pueden surgir diversos problemas cuando se redactan con frases de lenguaje natural, como la falta de claridad, puesto que algunas veces es difícil utilizar el lenguaje de forma precisa y no ambigua, y la confusión de requisitos, porque no se distinguen claramente unos de otros cuando representan una funcionalidad similar.
- Requisitos de sistema: Es una especificación funcional en forma de documento estructurado donde se establecen con detalle los servicios, funciones y restricciones del sistema. El nivel de detalle y especificación es intermedio, de forma que sea entendible tanto para el cliente o usuario final como para los desarrolladores software. Se puede decir que son versiones extendidas de los requisitos del usuario que son utilizados por los ingenieros del software como punto de partida para el diseño del sistema, agregando detalle y explicando cómo debe proporcionar el sistema los requisitos de usuario.

A menudo, los requisitos de sistemas software se clasifican en funcionales y no funcionales:

- RF (Requisitos Funcionales): Describen la funcionalidad o los servicios que se espera que el sistema suministre, la manera en que éste reaccionará a determinadas entradas y cómo se debe comportar en situaciones particulares. Dependen del tipo de software, del tipo de sistema donde se usará y de los posibles usuarios. Pueden ser tanto requisitos de sistema (con información más detallada) como de usuario (con información de más alto nivel).

- RNF (Requisitos No Funcionales): Son restricciones de los servicios o funciones ofrecidos por el sistema. Incluyen restricciones de tiempo como tiempos de respuesta, restricciones debidas a estándares, fiabilidad, capacidad E/S, etc. Los requisitos no funcionales normalmente se aplican al sistema en su totalidad, no a servicios individuales. Como pasa con los requisitos funcionales, pueden ser tanto requisitos de sistema (con información más detallada) como de usuario (con información de más alto nivel).

4.2.2.1. Descripción de los Requisitos de usuario de la aplicación

Se presenta a continuación un listado con los requisitos de usuario de la aplicación, añadiendo una breve descripción de cada uno de ellos e indicando en cada caso si se trata de un RF o RNF.

Estos requisitos van a ir agrupados según un determinado grupo de funcionalidades en diferentes tablas. Estos requisitos de usuario se describen en las tablas 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8 y 4.9.

Id.	DESCRIPCIÓN	RF	RNF
Requisitos generales			
R-1	El software deberá proporcionar un medio para representar la información obtenida a partir del fichero de trazas producido por NS-2 con los mensajes debidos a la actividad en la red (*.txt*).	X	
R-2	El software deberá proporcionar un medio para representar la información obtenida a partir del fichero de trazas con los mensajes debidos a la movilidad de los nodos en la red (*.pos*).	X	
R-3	La aplicación deberá ser portable y poder visualizarse en los siguientes sistemas operativos bajo Windows: Windows XP, Windows Vista, Windows 7.		X
R-4	La aplicación deberá ser portable y poder visualizarse en sistemas operativos bajo Linux.		X
R-5	La aplicación deberá ser desarrollada completamente utilizando software libre.		X
R-6	La aplicación deberá evitar que se produzcan errores que impliquen el cierre del programa, y en general, mostrarse estable ante todos los posibles eventos.		X
R-7	La interfaz de la aplicación deberá favorecer su usabilidad. Debe contener controles bien definidos y diferenciables.		X
R-8	El software deberá ofrecer un control donde se puede comenzar con la carga de un escenario nuevo, dejando la aplicación como si estuviera recién arrancada.	X	

Tabla 4.1. Requisitos de usuario: Requisitos generales

Id.	DESCRIPCIÓN	RF	RNF
Requisitos de la carga de ficheros			
R-9	La aplicación deberá permitir la carga de los dos ficheros (de salida de la simulación *.txt* y de movilidad de los nodos *.pos*) en cualquier orden, indistintamente.		X
R-10	La aplicación deberá permitir la creación del escenario para realizar la visualización de los movimientos de los nodos a través de la carga única del archivo de movilidad de los nodos (*.pos*)	X	
R-11	La aplicación deberá verificar la integridad de los datos del escenario obtenidos a través de ambos ficheros de entrada.	X	
R-12	Durante la carga de los ficheros, el software deberá indicar si ha detectado un identificador de nodo no válido en alguno de los mensajes.	X	
R-13	La apertura de ficheros arrancará desde el directorio local de la aplicación.		X

Tabla 4.2. Requisitos de usuario: Carga de ficheros

Id.	DESCRIPCIÓN	RF	RNF
Requisitos de la carga del escenario			
R-14	El software deberá situar los nodos en la posición inicial cargada desde el archivo de movilidad de los nodos (*.pos*) cuando se cree el escenario.	X	

Tabla 4.3. Requisitos de usuario: Carga del escenario

Id.	DESCRIPCIÓN	RF	RNF
Requisitos de la navegación de la animación			
R-15	La aplicación deberá presentar controles para avanzar y/o retroceder temporalmente por la animación desplazando un ítem sobre un eje temporal.	X	
R-16	La aplicación deberá presentar controles para avanzar y/o retroceder temporalmente por la animación introduciendo saltos temporales relativos al instante actual.	X	
R-17	La aplicación deberá presentar controles para avanzar y/o retroceder temporalmente por la animación introduciendo instantes absolutos.	X	
R-18	La aplicación deberá presentar controles para avanzar y/o retroceder temporalmente por la animación pausándose en los eventos relativos a los nodos que estén destacados o seleccionados.	X	
R-19	El software deberá ofrecer la posibilidad de pausar y reanudar la animación.	X	
R-20	El software deberá ofrecer la posibilidad de parar la animación.	X	
R-21	La aplicación deberá ofrecer un control para modificar la relación de velocidades entre la animación base y la que se pretende mostrar.	X	
R-22	La aplicación mostrará en todo momento el instante actual de la animación.	X	

Tabla 4.4. Requisitos de usuario: Navegación de la animación

Id.	DESCRIPCIÓN	RF	RNF
Requisitos de la vista del escenario			
R-23	La aplicación deberá ser capaz de ajustar las diferentes posiciones de los nodos como mínimo hasta el orden de milisegundos como consecuencia de la animación por saltos o desplazamientos temporales.	X	
R-24	El software deberá ser capaz de leer las dimensiones del escenario real desde el archivo de movilidad de los nodos (*.pos*) y cargar inicialmente el escenario virtual completamente dentro de la zona de la ventana principal de la aplicación habilitada para ello.		X
R-25	La aplicación deberá ofrecer la posibilidad de cambiar el tamaño de los nodos representados.	X	
R-26	La aplicación deberá presentar controles para la visualización de las coberturas de nodos y de los identificadores de nodos.	X	
R-27	La vista ofrecida por la aplicación del escenario virtual deberá ser al cargarse, por defecto, en planta (protección horizontal).	X	
R-28	El punto de vista ofrecido por la aplicación del escenario virtual deberá poder ser girado en torno a 90 grados.	X	
R-29	El escenario virtual generado deberá contener coordenadas representadas cada 100 metros.	X	
R-30	La rejilla presentada en el escenario deberá hacerse más o menos densa en función de la cercanía al escenario virtual.	X	

Tabla 4.5. Requisitos de usuario: Vista del escenario

Id.	DESCRIPCIÓN	RF	RNF
Requisitos del estudio de la caché local			
R-31	La aplicación deberá ofrecer al usuario la posibilidad de consultar el estado de la caché local de un cierto nodo para cualquier instante de tiempo.	X	
R-32	El software deberá ser capaz de realizar la búsqueda de un documento en la caché local.	X	

Tabla 4.6. Requisitos de usuario: Estudio de la caché local

Id.	DESCRIPCIÓN	RF	RNF
Requisitos del estudio de la caché de redirecciones			
R-33	La aplicación deberá ofrecer al usuario la posibilidad de consultar el estado de la caché de redirecciones de un cierto nodo para cualquier instante de tiempo.	X	
R-34	El software deberá ser capaz de realizar la búsqueda de un documento en la caché de redirecciones.	X	

Tabla 4.7. Requisitos de usuario: Estudio de la caché de redirecciones

Id.	DESCRIPCIÓN	RF	RNF
Requisitos del estudio de estadísticos			
R-35	La aplicación deberá presentar controles para mostrar el resumen de estadísticos (en la zona de la ventana principal de la aplicación habilitada para ello) siempre que se produzca un evento que modifique algún parámetro estadístico en los nodos que estén destacados o seleccionados.	X	
R-36	La aplicación deberá presentar controles para mostrar el resumen de estadísticos (en la zona de la ventana principal de la aplicación habilitada para ello) en cualquier instante de tiempo y para cualquier nodo.	X	
R-37	La aplicación deberá presentar controles para generar un informe en PDF con el resumen de estadísticos para todos los nodos del escenario en cualquier instante de tiempo.	X	

Tabla 4.8. Requisitos de usuario: Estudio de estadísticos

4.2.2.2. Descripción de los Requisitos de sistema de la aplicación

Se presenta a continuación un listado con los requisitos de sistema de la aplicación, añadiendo una breve descripción de cada uno de ellos e indicando en cada caso si se trata de un RF o RNF. Además, estos requisitos van a ir agrupados según unas determinadas funcionalidades, como se ha establecido para los requisitos de usuario.

Los requisitos de sistema, que contienen información más detallada de la que aportan los requisitos de usuario, son presentados en las tablas 4.9, 4.10, 4.11, 4.12, 4.13, 4.14, 4.15 y 4.16.

Id.	Descripción	RF	RNF
Requisitos generales			
R-38	El software deberá manejar la información cargada a partir del fichero de trazas con los mensajes debidos a la actividad en la red (*.txt*) en estructuras de información estables.		X
R-39	El software deberá manejar la información cargada a partir del fichero de trazas con los mensajes debidos a la movilidad de los nodos en la red (*.pos*) en estructuras de información estables.		X
R-40	La aplicación deberá no sólo utilizar una herramienta de desarrollo que sea software libre, sino que además será necesario que las librerías y componentes que haya que instalar para su correcto funcionamiento sean igualmente código libre.		X
R-41	La aplicación deberá poder actualizarse para responder a futuros cambios en sus requisitos (como consecuencia del uso de la misma, por ejemplo), con relativamente poca inversión en software. Es decir, el software debe presentarse mantenible.		X

R-42	La aplicación deberá ser extensible para poder adaptarse a las nuevas situaciones, como las causadas debido a la evolución de los datos que maneja la misma y/o modificaciones en las estrategias de cachés que es capaz de interpretar.		X

Tabla 4.9. Requisitos de sistema: Requisitos generales

Id.	Descripción	RF	RNF
Requisitos de la carga de ficheros			
R-43	El software deberá ser capaz de interpretar cada una de las líneas del fichero de salida de la simulación (*.txt*), desechando toda aquella información que no sea útil para la representación, y generando los parámetros de los eventos de tipo mensajes que se vayan sucediendo. Estos eventos de mensajes representan el tráfico que circula por la red.		X
R-44	El software deberá ser capaz de interpretar cada una de las líneas del fichero de movilidad de los nodos (*.pos*), desechando la información no útil para la representación, generando los parámetros de los eventos de cambio de dirección de los nodos que se vayan sucediendo.		X
R-45	El control de la apertura de los ficheros deberá filtrar los archivos por la extensión buscada (*.txt* o *.pos*), según sea el caso.		X
R-46	El software deberá generar algún tipo de indicación expresando que se está leyendo, traduciendo y almacenando la información que se encuentra contenida en el archivo de salida de la simulación (*.txt*).	X	
R-47	La aplicación deberá mostrar un control que indique que ya se ha cargado satisfactoriamente la información contenida en el archivo de salida de la simulación (*.txt*).	X	
R-48	El software deberá generar algún tipo de aviso de indicación expresando que se está leyendo, traduciendo y almacenando la información que se encuentra contenida en el archivo de movilidad de los nodos (*.pos*).	X	
R-49	La aplicación deberá mostrar un control que indique que ya se ha cargado satisfactoriamente la información contenida en el archivo de movilidad de los nodos (*.pos*).	X	
R-50	El software deberá ser capaz de restringir la creación del escenario y su posterior inserción en la zona de la ventana principal de la aplicación habilitada para ello, o bien cuando se haya producido la carga satisfactoria de los datos del archivo de movilidad de los nodos (*.pos*), si se pretende visualizar únicamente una animación con los desplazamientos de los nodos, o bien cuando se produzca la carga satisfactoria de los dos ficheros de entrada (siempre y cuando se haya leído en primer lugar el archivo de salida de simulación *.txt*), si se pretende visualizar una animación con todos los eventos que se generan en la red.		X

R-51	El software deberá ser capaz de cargar en cualquier instante de la visualización los datos del archivo de salida de simulación (*.txt*) cuando se haya creado previamente el escenario tras la carga satisfactoria del archivo con la movilidad de los nodos (*.pos*), en cuyo caso deberá llevarse la animación hacia el instante inicial.	X	
R-52	Para que se considere la carga satisfactoria de la información de los dos ficheros de entrada deberá cumplirse una integridad de los datos, de forma que ambos ficheros contengan el mismo número de nodos y la misma duración de la simulación, teniendo valores para ambos parámetros mayores a 0, y unas dimensiones del escenario también mayores a 0. Si no se cumple la integridad de la información, se deberá empezar con la carga de los dos archivos de entrada de nuevo.		X
R-53	El sistema deberá avisar cuando se intente cargar un nuevo archivo de salida de la simulación (*.txt*) cuando se haya cargado anteriormente uno a tal fin, sin que esto modifique el estado en que se encuentre la aplicación.	X	
R-54	El sistema deberá avisar cuando se intente cargar un nuevo archivo de movilidad de los nodos (*.pos*) cuando se haya cargado anteriormente uno a tal fin, sin que esto modifique el estado en que se encuentre la aplicación.	X	
R-55	El software deberá detectar que los eventos del archivo de movilidad de los nodos (*.pos*) produzcan cambios de dirección en nodos que tengan un identificador numérico correcto, teniendo en cuenta que el rango numérico válido será [0, *numNodes* – 1].	X	
R-56	En aquellos campos o atributos pertenecientes a todos los mensajes del archivo de salida de la simulación (*.txt*) donde la información representada sea relativa a la identificación numérica de un nodo, el software deberá ser capaz de detectar que se correspondan con valores correctos, teniendo en cuenta que el rango numérico válido será [0, *numNodes* – 1].	X	

Tabla 4.10. Requisitos de sistema: Carga de ficheros

Id.	Descripción	RF	RNF
Requisitos de la carga del escenario			
R-57	El software deberá ser capaz de estimar el *zoom* inicial para posicionar la cámara en la vista en planta a partir de las dimensiones de la altura y la anchura del escenario virtual cargadas del archivo de movilidad (*.pos*), de forma que quede completamente dentro del espacio de la ventana principal de la aplicación que se ha habilitado para ello.	X	
R-58	El software deberá reconocer el valor de la duración de la simulación tomado durante la carga del archivo de movilidad de los nodos (*.pos*) o tomado de ambos ficheros de entrada para poder actualizar el valor máximo de la barra de desplazamiento o *slider* que representa el avance temporal de la animación.	X	

R-59	El software deberá reconocer el valor de la duración de la simulación tomado durante la carga del archivo de movilidad de los nodos (*.pos*) o tomado de ambos ficheros de entrada para restringir el número de cifras a introducir en las cajas de texto de la ventana principal de la aplicación para el avance y/o retroceso por tiempo absoluto o por saltos temporales.		X
R-60	El software deberá reconocer el valor del número de nodos tomado durante la carga del archivo de movilidad de los nodos (*.pos*) o tomado de ambos ficheros de entrada y rellenar los controles con el listado de nodos para la selección y/o deselección de los nodos y de sus coberturas.	X	
R-61	La aplicación será capaz de gestionar todos los eventos procedentes del archivo de salida de la simulación (*.txt*) y del archivo de movilidad de los nodos (*.pos*) en una misma estructura óptima para ser recorrida, de forma que los eventos provenientes de ambos ficheros se encontrarán ordenados por el instante temporal en el que vayan teniendo lugar.	X	
R-62	La aplicación deberá habilitar todos los controles de la ventana una vez el escenario se haya creado satisfactoriamente, con excepción de los botones para el avance y/o retroceso por eventos, que por defecto estarán desactivados mientras no se haya seleccionado ningún nodo y no se haya marcado que se produzca para todos los nodos de la red.		X

Tabla 4.11. Requisitos de sistema: Carga del escenario

Id.	Descripción	RF	RNF
Requisitos de la navegación de la animación			
R-63	La aplicación deberá ser capaz de ajustar las posiciones de los nodos y obtener el siguiente evento que se debe producir cuando se presiona alguno de los controles para avanzar y/o retroceder temporalmente por la animación al desplazar un ítem sobre la barra horizontal o *slider* temporal.	X	
R-64	La aplicación deberá ser capaz de ajustar las posiciones de los nodos y obtener el siguiente evento que se debe producir cuando se presiona alguno de los controles para avanzar y/o retroceder temporalmente por la animación al introducir saltos temporales relativos al instante actual de la animación.	X	
R-65	La aplicación deberá ser capaz de ajustar las posiciones de los nodos y obtener el siguiente evento que se debe producir cuando se presiona alguno de los controles para avanzar y/o retroceder temporalmente por la animación al introducir instantes absolutos.	X	
R-66	La aplicación deberá ser capaz de ajustar las posiciones de los nodos y obtener el siguiente evento que se debe producir cuando se presiona alguno de los controles para avanzar y/o retroceder temporalmente por la animación al ir pausándose en los eventos relativos a los nodos que estén destacados o seleccionados.	X	

R-67	El software deberá ofrecer la posibilidad de reanudar la animación cuando se encuentre pausada ya sea al principio de la misma, al final, o en cualquier punto intermedio. En concreto, deberá cerciorarse de su llegada al final de la animación para poder reanudarse desde el principio, llevando los nodos a su posición inicial (coordenadas *x*, *y* y *z* iniciales) marcada por el contenido del archivo de movilidad de los nodos (*.pos*).	X	
R-68	El software deberá ofrecer la posibilidad de pausar la animación cuando se encuentre activa en cualquier instante temporal de la misma.	X	
R-69	El software deberá ofrecer la posibilidad de parar la animación cuando se encuentre activa o se encuentre en estado pausado en cualquier instante temporal de la animación, cerciorándose de que la animación permanezca pausada y llevando los nodos a su posición inicial (coordenadas *x*, *y* y *z* iniciales) marcada por el contenido del archivo de movilidad de los nodos (*.pos*).	X	
R-70	El software deberá ser capaz de modificar la relación de velocidades entre la animación de base y la que se pretenda mostrar, teniendo en cuenta el recorrido que lleva cada nodo y provocando la llegada a su destino en el instante que venga ahora marcado por la nueva velocidad.	X	
R-71	El software deberá proporcionar el cambio de velocidad marcado por el requisito anterior para cualquier estado de la animación, incluyendo la animación en estado activo o en estado pausado, y pudiéndose producir en cualquier instante de la animación.	X	
R-72	La aplicación presentará un control de forma que podrá modificar la relación de la velocidad de visualización de la animación presentada con los dos requisitos anteriores. Las posibilidades para este control, presentadas como el ratio entre la velocidad de la animación y la velocidad base de la simulación, variarán desde 0.1 (lo que implica que se está visualizando 10 veces más lento de lo que viene marcado por la velocidad de la simulación) hasta 10 (lo que implica que se está visualizando 10 veces más rápido de lo que viene marcado por la velocidad de la simulación). El valor por defecto de dicho ratio cuando se cree el escenario será de 1, es decir, se visualizará a la velocidad a la que venga marcada por los eventos de la simulación.	X	
R-73	El software deberá mostrar en todo momento el instante actual de la animación, ya sea considerando la animación normal o bien desplazándose por saltos temporales.	X	
R-74	El valor del instante temporal se actualizará al menos por cada décima de segundo cuando se esté ejecutando una visualización normal.		X
R-75	El valor del instante temporal se actualizará con el valor exacto del evento cuando se esté ejecutando la visualización en modo avance por eventos para los nodos que estén destacados o seleccionados.		X

R-76	El valor del instante temporal se actualizará con el valor exacto introducido en las cajas de texto (en milisegundos) cuando se esté ejecutando la visualización en modo avance/retroceso por saltos temporales relativos o bien en forma absoluta.		X
R-77	La aplicación deberá controlar que los valores de instantes de tiempo introducidos en las cajas de texto para realizar un avance/retroceso por saltos temporales o en forma absoluta (que irán en milisegundos) sean consistentes, es decir, que se permita únicamente introducir cifras decimales, rechazando cualquier otro carácter.		X
R-78	La aplicación deberá controlar que los valores de instantes de tiempo introducidos en la caja de texto con el fin de realizar un avance/retroceso por saltos temporales produzcan un desplazamiento temporal válido, esto es, superior o igual al instante inicial de la simulación (0 segundos) e inferior o igual al instante final o duración de la simulación.		X
R-79	El software deberá ser capaz de deshabilitar los botones de avance y/o retroceso por eventos cuando no se haya indicado que se realice para todos los nodos de la red y, además, no se tenga seleccionado ningún nodo, habilitando dichos botones para el resto de los casos.		X
R-80	El software deberá ser capaz de reconocer los nodos que están seleccionados en cada momento para avanzar y/o retroceder por eventos relacionados únicamente para esos nodos, aunque también se añadirá adicionalmente la posibilidad de avanzar y/o retroceder por eventos relacionados para todos los nodos implicados en la simulación.	X	
R-81	El software deberá ser capaz de actualizar los estadísticos cada vez que se produzca algún evento (de aquellos que sean provenientes del archivo de salida de simulación *.txt*) que desencadene un cambio en alguno de los parámetros para el caso de los nodos seleccionados, ya sea durante la animación normal de la visualización o sea por el avance/retroceso por eventos. El resultado se mostrará en la zona de la ventana principal de la aplicación habilitada para ello.	X	
R-82	El software deberá ser capaz de actualizar los estadísticos para los nodos que estén seleccionados cada vez que se produzca un avance y/o retroceso por saltos temporales o de forma absoluta. El resultado se mostrará en la zona de la ventana principal de la aplicación habilitada para ello.	X	

Tabla 4.12. Requisitos de sistema: Navegación de la animación

Id.	Descripción	RF	RNF
Requisitos de la vista del escenario			
R-83	El software deberá interpolar los puntos intermedios del movimiento de cada nodo dada la posición final de cada cambio de dirección y la velocidad con la que se dará el mismo, siendo siempre rectilíneos.	X	

R-84	El software detectará las discontinuidades en la temporización de la animación como consecuencia o bien de la animación por eventos, o bien por desplazamientos temporales, ya sean absolutos o relativos, o bien por el desplazamiento de la barra temporal o *slider* de la animación. Obtendrá las diferentes posiciones de los nodos en función de la dirección y sentido del desplazamiento actual que tenga cada nodo y del tiempo transcurrido desde el último cambio de dirección, y los situará en el escenario en las coordenadas exactas.	X	
R-85	La aplicación dibujará los bordes del escenario virtual atendiendo a las dimensiones del mismo leídas desde el archivo de movilidad de los nodos (*.pos*) y estimará el *zoom* inicial desde donde posicionar la vista en planta para asegurarse de que el escenario virtual quede completamente dentro de la zona de la ventana principal de la aplicación habilitada para ello.	X	
R-86	La aplicación presentará un control de forma que podrá seleccionar diferentes posibilidades para los radios de los nodos representados en el escenario virtual. Los nodos se representarán como esferas cuyo radio vendrá dado por el valor seleccionado por el control anteriormente mencionado. Los posibles valores presentarán un rango variado, siendo el valor por defecto de 10 metros.	X	
R-87	La aplicación permitirá modificar el radio de los nodos en cualquier instante, es decir, tanto estando la animación en estado pausado como en estado activo.	X	
R-88	El software permitirá seleccionar de un listado los nodos cuyas coberturas se deseen observar. Además, se podrán seleccionar y/o deseleccionar en cualquier instante, es decir, tanto estando la animación en estado pausado como en estado activo.	X	
R-89	El software ofrecerá la posibilidad de seleccionar y/o deseleccionar las coberturas de todos los nodos a la vez, utilizando una nueva opción en el control presentado en el requisito anterior.	X	
R-90	El software permitirá elegir de un listado los nodos que se deseen marcar como nodos seleccionados, así como desmarcarlos. Además, se podrán seleccionar y/o deseleccionar en cualquier instante, es decir, tanto estando la animación en estado pausado como en estado activo.	X	
R-91	El software ofrecerá la posibilidad de seleccionar y/o deseleccionar (para que aparezcan como nodos seleccionados) todos los nodos a la vez, utilizando una nueva opción en el control presentado en el requisito anterior.	X	
R-92	La aplicación deberá ofrecer la posibilidad de destacar de alguna manera los nodos que intervengan en el camino seguido por los mensajes que pertenezcan a una petición realizada por un nodo que se encuentre seleccionado, que será representado con el "halo" del nodo. Esta representación se realizará en cualquier momento de la animación, ya sea en visualización normal o con avance y/o retroceso por eventos.	X	

R-93	El software permitirá elegir mostrar los identificadores de: - todos los nodos, distinguiendo con colores diferentes los identificadores para los nodos que se encuentren seleccionados de los identificadores de aquellos nodos que no estén seleccionados. - sólo los identificadores de los nodos que se encuentren seleccionados, con el color elegido para tal fin. - ningún identificador de nodo. Además, se podrá cambiar entre las diferentes opciones en cualquier instante, es decir, tanto estando la animación en estado pausado como en estado activo.	X	
R-94	La aplicación deberá ofrecer la posibilidad de representar de alguna manera la circulación por la red de los paquetes que pertenezcan al camino seguido por una petición realizada por un nodo que se encuentre seleccionado, si la animación está en estado de visualización normal, o bien en avances/retrocesos por eventos (para los nodos que estén seleccionados o para todos los nodos).	X	
R-95	La aplicación presentará como animación extra una pequeña radiación indicando cuándo un nodo ha realizado la petición de un documento, independientemente que sea un nodo seleccionado o no, que se podrá visualizar cuando se esté avanzando y/o retrocediendo por eventos.	X	
R-96	La aplicación presentará la opción de selección de nodos con el fin de realizar las funciones siguientes: - realizar un estudio de los eventos que estén relacionados con los nodos seleccionados, de forma que: si se trata de un evento de cambio de dirección proveniente del archivo de movilidad de los nodos (*.pos*), se debe considerar para avanzar y/o retroceder por eventos en nodos seleccionados, mostrando la descripción del evento en la zona de la ventana principal de la aplicación habilitada para ello; si se trata de un evento proveniente del archivo de salida de la simulación (*.txt*), se deberá tomar en consideración cuando se avance y/o retroceda por eventos en nodos seleccionados y sea un mensaje que pertenezca al camino seguido por una petición de documento realizada por un nodo que se encuentre seleccionado, mostrándose su descripción como en el caso anterior en la zona de la ventana principal de la aplicación habilitada para ello. - conocer los estadísticos de los nodos seleccionados cuando se produzca un mensaje del archivo de salida de la simulación (*.txt*) que actualice alguno de sus parámetros, mostrándose un resumen en la zona de la ventana principal de la aplicación habilitada para ello, ya sea durante la visualización normal de la aplicación como por saltos temporales. - representar las animaciones de los paquetes relativos a la petición de un documento por parte de un nodo que esté seleccionado, ya sea durante la visualización normal de la aplicación como por eventos.	X	

	- representar los halos de los nodos que reciban y/o envíen paquetes relativos a la petición de un documento por parte de un nodo que esté seleccionado.		
R-97	La aplicación ofrecerá la posibilidad de seleccionar y/o deseleccionar un nodo, con todo lo que conlleva lo especificado en el requisito anterior, a través del ratón, concretamente clicando sobre el nodo en particular con el botón izquierdo.	X	
R-98	Los nodos que aparezcan como seleccionados presentarán un color distintivo, así como su identificador de nodo, cuando éste deba mostrarse.	X	
R-99	La aplicación deberá añadir la posibilidad de clicar sobre un nodo del escenario virtual con el botón derecho, de forma que se obtenga un completo menú de posibilidades: - mostrar los estadísticos acumulados hasta el instante actual de tiempo del nodo en particular en la zona de la ventana principal de la aplicación habilitada a tal fin. - mostrar el contenido de la caché local en el instante de tiempo actual del nodo en particular en una nueva ventana, cuyo contenido viene representado a modo de tabla, y con las opciones propias del estudio de la caché local que se presentará en el apartado de requisitos correspondiente. - mostrar el contenido de la caché de redirecciones en el instante de tiempo actual del nodo en particular en una nueva ventana, cuyo contenido viene representado a modo de tabla, y con las opciones propias del estudio de la caché de redirecciones que se presentará en el apartado de requisitos correspondiente. - mostrar el punto en el escenario real (indicando las coordenadas *x*, *y* y *z*) que posee el nodo en particular en el instante actual en la zona de la ventana principal de la aplicación habilitada para ello. - mostrar el identificador del nodo en particular (siempre y cuando no se encuentre ya mostrado en ese instante de tiempo) durante tiempo reducido, y con un color diferenciado en función de si se trata de un nodo que se encuentre seleccionado o no. - mostrar la cobertura del nodo en particular, debiendo quedar seleccionado a su vez el ítem del nodo dentro del listado de coberturas presentado como control de la ventana principal de la aplicación con el fin de seleccionar/deseleccionar las coberturas a visualizar.	X	
R-100	La vista ofrecida por la aplicación del escenario virtual en planta (proyección horizontal) podrá modificarse realizando desplazamientos hacia arriba, hacia abajo, hacia la derecha y/o hacia la izquierda, a través de los oportunos controles en la ventana principal de la aplicación, o a través de las flechas indicadoras en los teclados de cualquier ordenador.	X	

R-101	La vista ofrecida por la aplicación del escenario virtual en planta (proyección horizontal) podrá acercarse y/o alejarse con un *zoom*, a través de un control en la ventana principal de la aplicación a modo de barra de desplazamiento o *slider* vertical con un ítem para desplazarse, y a través de la rueda de desplazamiento o *scroll* del ratón.	X	
R-102	La vista rotada ofrecida por la aplicación del escenario virtual podrá modificarse realizando desplazamientos hacia arriba, hacia abajo, hacia la derecha y/o hacia la izquierda, a través de los oportunos controles en la ventana principal de la aplicación, a través de las flechas indicadoras en los teclados de cualquier ordenador, a través de un control en la ventana principal de la aplicación a modo de barra de desplazamiento o *slider* vertical con un ítem para desplazarse, o a través de la rueda de desplazamiento o *scroll* del ratón. Hay que destacar que el movimiento de la cámara que sigue las modificaciones solicitadas en la vista cambiarán su significado respecto a la vista normal (proyección horizontal).	X	
R-103	La vista rotada ofrecida por la aplicación del escenario virtual podrá acercarse y/o alejarse con un *zoom*, a través de los oportunos controles en la ventana principal de la aplicación, o a través de las flechas indicadoras en los teclados de cualquier ordenador.	X	
R-104	El cambio de la vista ofrecida por la aplicación, ya sea desde vista rotada hacia vista en planta (proyección horizontal) o viceversa, colocará al escenario en unas coordenadas de partida, evitando que si se produce la pérdida de la vista del escenario, se pueda llevar a una posición de referencia donde el escenario aparezca centrado en la zona de la ventana principal de la aplicación dedicada a ello.	X	

Tabla 4.13. Requisitos de sistema: Vista del escenario

Id.	Descripción	RF	RNF
Requisitos del estudio de la caché local			
R-105	El software permitirá la consulta del contenido de la caché local para un cierto nodo de cualquiera de las formas siguientes: - clicando sobre el nodo en cuestión con el botón derecho y seleccionando la opción a tal fin. - seleccionando de la barra de menús de la ventana principal de la aplicación el control que permite elegir visualizar la caché local de un nodo eligiendo en primer lugar el nodo para el que se quiere realizar la consulta.	X	
R-106	La caché local deberá representarse de forma que las entradas que no sean válidas, es decir, aquellas donde el tiempo de expiración haya cumplido, tendrán la etiqueta "*EXPIRED*" en la columna dedicada al tiempo de expiración y tendrá un color diferenciable.	X	

R-107	La caché local se deberá visualizar en una nueva ventana emergente, representando su contenido a modo de filas y columnas de una tabla. Dicho contenido variará en función de las opciones siguientes: - mostrar todos los documentos. - mostrar sólo los documentos no expirados. - mostrar el documento buscado por su identificador numérico, en el caso de que se encuentre en la caché local.	X	

Tabla 4.14. Requisitos de sistema: Estudio de la caché local

Id.	Descripción	RF	RNF
Requisitos del estudio de la caché de redirecciones			
R-108	El software mostrará, para cada uno de los documentos para el que se posee información, una entrada diferente para los registros GET y RESP de cada documento, siempre y cuando dichos registros tengan datos almacenados.	X	
R-109	El software permitirá la consulta del contenido de la caché de redirecciones para un cierto nodo de cualquiera de las formas siguientes: - clicando sobre el nodo en cuestión con el botón derecho y seleccionando la opción a tal fin. - seleccionando de la barra de menús de la ventana principal de la aplicación el control que permite elegir visualizar la caché de redirecciones de un nodo eligiendo en primer lugar el nodo para el que se quiere realizar la consulta.	X	
R-110	La caché de redirecciones se deberá visualizar en una nueva ventana emergente, representando su contenido a modo de filas y columnas de una tabla. Dicho contenido variará en función de las opciones siguientes: - mostrar todos los documentos. - mostrar sólo los documentos no expirados. - mostrar el documento buscado por su identificador numérico, en el caso de que se encuentre en la caché de redirecciones.	X	
R-111	La caché de redirecciones deberá representarse de forma que las entradas que no sean válidas, es decir, aquellas donde el tiempo de expiración haya cumplido, tendrán la etiqueta "*EXPIRED*" en la columna dedicada al tiempo de expiración y tendrán un color diferenciable.	X	
R-112	La caché de redirecciones deberá representarse de forma que las entradas que posean un tiempo de expiración nulo, tendrán en esa columna la etiqueta "*Not Available*".	X	

Tabla 4.15. Requisitos de sistema: Estudio de la caché de redirecciones

Id.	Descripción	RF	RNF
Requisitos del estudio de estadísticos			
R-113	El informe generado de tipo PDF contendrá un resumen con los valores de todos los estadísticos acumulados desde el inicio de la simulación hasta el instante en que se haya solicitado dicho informe para todos los nodos.	X	
R-114	El informe generado de tipo PDF contendrá un histograma para cada uno de los estadísticos bajo estudio con el fin de comparar el rango de valores obtenidos por los nodos.	X	
R-115	El informe generado de tipo PDF contendrá un cuadro resumen final que indique el valor medio entre todos los nodos de cada uno de los estadísticos bajo estudio.	X	
R-116	La aplicación deberá presentar un resumen con los estadísticos de un nodo en la zona de la ventana principal de la aplicación habilitada para ello siempre que: - se clique un nodo del escenario con el botón derecho del ratón y se seleccione la opción a tal fin. - se produzca un evento proveniente del archivo de salida de simulación (*.txt*) que modifique algún parámetro estadístico en los nodos que estén destacados o seleccionados, ya sea durante la visualización normal de la aplicación o cuando se esté avanzando y/o retrocediendo por eventos, o para todos los nodos si es avance/retroceso por eventos de todos ellos.	X	

Tabla 4.16. Requisitos de sistema: Estudio de estadísticos

4.3. FASE DE DESARROLLO DEL SOFTWARE

4.3.1. CARACTERÍSTICAS

La fase de desarrollo se centra en el cómo, intentando definir cómo han de diseñarse las estructuras de datos, cómo debe implementarse la función como una arquitectura del software y los detalles procedimentales, cómo han de caracterizarse las interfaces, cómo han de realizarse las pruebas, etc. Estas actividades se engloban dentro de las tres tareas principales del desarrollo de software, que son el diseño del software, la generación del código y la prueba del software. Aunque la prueba del software pertenece a esta fase de desarrollo, se verá en un capítulo aparte, indicando cada uno de los escenarios utilizados para validar y verificar cada uno de los requisitos del proyecto planteados.

4.3.2. DISEÑO DEL SOFTWARE

El diseño del software es una tarea clave que definirá la propia calidad de software, propiciando la estructuración del mismo para admitir posibles cambios. Además, deberá

estructurarse para degradarse poco a poco cuando deba enfrentarse a datos, sucesos o condiciones operativas aberrantes.

Algunos conceptos fundamentales a tener en cuenta para el diseño software:

- Abstracción y refinamiento paso a paso: la abstracción implica modelar la realidad con diferentes niveles de detalle, desde descripciones de más alto nivel, para ir procediendo a sucesivos refinamientos, hasta llegar al mínimo detalle.
- Modularidad: dividir y compartimentar datos y funciones. Evita errores o efectos laterales no deseados, además que propicia el trabajo en paralelo. El problema que surge es considerar el número óptimo de los módulos, lo que necesita de un compromiso: si el número de módulos es demasiado pequeño, el coste de desarrollo del módulo es muy alto aunque el coste de integración es muy reducido, y a medida que va creciendo el número de módulos, se reduce el coste del desarrollo de cada módulo por ser más simples, pero aumenta el coste de integración de los mismos.
- Independencia funcional: se consigue desarrollando módulos con una función única y una aversión a excesiva interacción con otros módulos. Las dos características que definen la independencia funcional son: la **cohesión**, que es una medida de la fuerza relativa funcional de un módulo, y el **acoplamiento**, que es una medida de la interdependencia relativa entre los módulos. Lo ideal es que el software sea muy cohesionado y poco acoplado, lo que implica que tenga módulos con funciones muy concretas y que se comunique lo mínimo con otros módulos.
- Ocultación de la información: se debe ocultar la información en la medida de lo posible para evitar el acoplamiento entre módulos, puesto que se reduce la probabilidad de efectos laterales, limita el impacto de decisiones de diseño locales, resalta la importancia de la comunicación a través de interfaces controladas y dirige hacia la **encapsulación**, que es un atributo de diseño de alta calidad.

Diferentes aspectos del diseño software:

- El diseño de datos transforma el modelo de la información en las estructuras de datos concretas necesarias para implementar el software.
- El diseño arquitectónico define la relación entre los principales elementos estructurales de programa.
- El diseño de interfaz describe cómo se comunica el software consigo mismo, con los sistemas que operan con él y con los operadores que lo emplean.

- El diseño procedimental transforma elementos estructurales de la arquitectura del programa en una descripción procedimental de los componentes de software.

La decisión de diseño elegida para el desarrollo de la aplicación es un diseño orientado a objetos, puesto que consigue aunar las tres características definidas anteriormente que denotan alta calidad del software (cohesión, acoplamiento y encapsulación), además que está motivada por el uso de la tecnología Java.

4.3.2.1. Características del Diseño Orientado a Objetos

Un sistema orientado a objetos está compuesto de objetos que interactúan, los cuales mantienen ellos mismos su estado local y proveen operaciones sobre su estado. Por esta característica se dicen que un objeto es autocontenido. Los objetos son abstracciones del mundo real, son independientes y encapsulan el estado y la representación de su información, puesto que la representación del estado es privada y no se puede acceder a ella directamente desde fuera del objeto. El proceso de diseño orientado a objetos comprende el diseño de clases de objetos y las relaciones entre estas clases. Cuando el diseño se implementa como un programa ejecutable, los objetos requeridos se crean dinámicamente utilizando las definiciones de las clases.

El diseño orientado a objetos es parte del desarrollo orientado a objetos en el que se utiliza una estrategia orientada a objetos:

- El análisis orientado a objetos comprende el desarrollo de un modelo orientado a objetos de dominio de la aplicación. Los objetos identificados reflejan las entidades y operaciones que se asocian con el problema a resolver.
- El diseño orientado a objetos comprende el desarrollo de un modelo orientado a objetos de un sistema software para implementar los requisitos identificados. Los objetos en un diseño orientado a objetos están relacionados con la solución del problema a resolver. Pueden existir relaciones estrechas entre algunos objetos del problema, pero inevitablemente el diseñador tiene que agregar nuevos objetos para implementar la solución.
- La programación orientada a objetos se refiere a implementar el diseño del software utilizando un lenguaje de programación orientado a objetos, como Java. Un lenguaje orientado a objetos provee los recursos para definir las clases y un sistema para crear los objetos correspondientes a las clases.

Los sistemas orientados a objetos son más fáciles de mantener que los sistemas desarrollados con otras aproximaciones, debido a que los objetos son independientes. Cambiar la implementación de un objeto o agregarle servicios no debe afectar a los otros objetos del sistema, mientras se mantenga la interfaz entre ellos. Los objetos son componentes potencialmente reutilizables porque son encapsulamientos independientes de estado y operaciones. Los diseños se pueden desarrollar utilizando objetos creados en los diseños previos. Esto reduce los costes de diseño, programación y validación. Han aparecido varias notaciones diferentes para describir desarrollos orientados a objetos, y UML es la integración de estas notaciones. UML da un conjunto de herramientas con las que se pueden realizar distintos modelos a distintos niveles de abstracción durante las fases de análisis y diseño orientado a objetos, siendo hoy día un estándar de facto. Dentro del apartado dedicado al modelado del sistema bajo estudio, se ofrecerá una visión más extensa de UML, adjuntando además los diagramas generados durante el análisis y el diseño del proyecto.

4.3.2.2. Arquitectura del Software: Modelo Vista Controlador

La arquitectura del software define la estructura global del software y las formas en que esa estructura proporciona integridad conceptual a un sistema. Representa la estructura jerárquica de los módulos, la forma de interactuar entre ellos y la estructura de datos usados por los mismos. La arquitectura, además de proporcionar cómo se distribuyen los componentes de un sistema y sus interacciones, es capaz de ofrecer propiedades extra-funcionales, dando una idea de cómo el diseño hará cumplir los requisitos de rendimiento, capacidad, fiabilidad y seguridad.

Por las características de la aplicación que se ha desarrollado, se ha optado por una arquitectura de software basada prácticamente en su totalidad en el patrón de diseño denominado MVC (Modelo Vista Controlador), muy utilizado en Ingeniería del Software. La característica fundamental de este patrón de diseño es la de separar la lógica del dominio o lógica de la aplicación, de los datos de entrada y de la presentación de los mismos, permitiendo un desarrollo, pruebas y mantenimiento independientes para cada uno.

- El modelo es la representación específica de la información sobre la cual la aplicación opera. La lógica de la aplicación añade significado a los datos cargados, y facilita sus presentaciones en la vista. El sistema también puede operar con más datos no relativos a la presentación, haciendo uso integrado de datos afines con el

sistema modelado. Cuando un modelo cambia su estado, se lo notifica a sus vistas asociadas para que puedan refrescar ese contenido. Muchas aplicaciones utilizan un mecanismo persistente de almacenamiento de datos, como una base de datos. El MVC no menciona específicamente la capa de acceso de datos porque se entiende que está encapsulado por el propio modelo. Los datos de entrada de la aplicación se cargarán de los dos ficheros ya mencionados, tomando de ellos la información necesaria y relevante para el correcto funcionamiento de la misma, y creando los objetos que contengan toda esa información.

- La vista se encarga de presentar el modelo en un formato adecuado para interactuar, normalmente a través de la interfaz de usuario. Pueden existir múltiples vistas de un mismo modelo con el fin de responder a diferentes propósitos.
- El controlador recibe eventos de entrada, generalmente acciones del usuario, y responde invocando peticiones al modelo y, probablemente, a la vista.

El MVC pretende separar la capa visual gráfica de su correspondiente programación y acceso a datos, algo que mejora el desarrollo y mantenimiento de la vista y el controlador en paralelo, ya que ambos cumplen ciclos de vida muy distintos entre sí. Aunque se pueden encontrar diferentes implementaciones del MVC, el flujo que sigue el control es generalmente el siguiente:

1) el usuario interactúa con la interfaz de usuario de alguna forma (pulsando un botón, eligiendo una opción de un menú, seleccionando de un listado desplegable, etc.);

2) el controlador recibe la notificación del evento desde los objetos de la vista (o la interfaz) indicando la acción solicitada por el usuario. El controlador gestiona el evento que le llega, frecuentemente a través de un gestor de eventos (usualmente denominado *handler* o manejador de eventos, como es el caso del *thread* de Java definido a tal fin);

3) el controlador accede al modelo, obteniendo su información e incluso puede actualizarlo, posiblemente modificándolo acorde a la acción solicitada por el usuario;

4) la vista actualiza la interfaz de usuario de forma apropiada para mostrar la información acerca de los modelos en concordancia con el cambio que éstos hayan podido sufrir. El modelo no debería tener conocimiento directo sobre la vista, si bien se podría utilizar un observador para proveer cierta relación, entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Una aproximación a esta solución ha sido la utilizada en la aplicación diseñada, de modo que existe una clase que contiene información susceptible de ser utilizada en la vista;

5) la interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo de nuevo.

4.4. MODELADO DEL SOFTWARE

Un buen software es aquel que satisface las necesidades cambiantes de sus usuarios. Y para producir un software que cumpla este propósito, hay que conocer e involucrar a los usuarios de forma disciplinada con el fin de extraer los requisitos reales del sistema. Con el fin de obtener un software de calidad duradera, hay que idear una sólida base arquitectónica que sea flexible al cambio. Esto implica el desarrollo de un software de forma eficiente y con el mínimo de desechos software y de trabajo repetido, que únicamente se consigue con las herramientas adecuadas y el enfoque apropiado. Para hacer todo esto de forma consistente y predecible, hay que disponer de un proceso de desarrollo sólido que pueda adaptarse a las necesidades cambiantes del problema en cuestión y de la tecnología. El modelado es una parte central de todas las actividades que conducen a la producción de buen software. Construimos modelos para comunicar la estructura deseada y el comportamiento de nuestro sistema, para visualizar y controlar la arquitectura del sistema, y para comprender mejor el sistema que estamos construyendo, muchas veces descubriendo oportunidades para la simplificación y reutilización. Construimos modelos para controlar el riesgo.

Un modelo es una simplificación de la realidad. Un buen modelo incluye aquellos elementos que tienen una gran influencia y omite aquellos elementos menores que no son relevantes para el nivel de abstracción dado. Todos los sistemas pueden ser descritos desde diferentes perspectivas utilizando diferentes modelos, y cada modelo es, por tanto, una abstracción semánticamente cerrada del sistema. El uso del modelado tiene una abundante historia en todas las disciplinas de ingeniería. Esa experiencia sugiere cuatro principios básicos de modelado:

1. La elección acerca de qué modelos crear tiene una profunda influencia sobre cómo se acomete un problema y cómo se da forma a una solución.
2. Todo modelo puede ser expresado con diferentes niveles de precisión.
3. Los mejores modelos están ligados a la realidad.
4. Un único modelo o vista no es suficiente. Cualquier sistema no trivial se aborda mejor a través de un pequeño conjunto de modelos casi independientes con múltiples puntos de vista.

4.4.1. MODELADO ORIENTADO A OBJETOS. UML

En el software hay varias formas de enfocar un modelo; una de las más comunes es la perspectiva orientada a objetos. En este enfoque, el principal bloque de construcción de todos los sistemas software es el objeto o clase. Actualmente, el enfoque orientado a objetos forma parte de la tendencia principal para el desarrollo de software. Más aún, la mayoría de los lenguajes actuales, sistemas operativos y herramientas son orientados a objetos de alguna manera, lo que ofrece más motivos para ver el mundo en términos de objetos. El desarrollo orientado a objetos proporciona la base fundamental para ensamblar sistemas a partir de componentes utilizando tecnologías como J2EE (*Java 2 Platform, Enterprise Edition*) y .NET. Visualizar, especificar, construir y documentar sistemas orientados a objetos es exactamente el propósito del Lenguaje Unificado de Modelado, UML.

El Lenguaje Unificado de Modelado es un lenguaje estándar para escribir planos de software. UML puede utilizarse para visualizar, especificar, construir y documentar los aspectos de un sistema que involucre una gran cantidad de software. UML es sólo un lenguaje y, por tanto, es tan sólo una parte de un método de desarrollo de software. Como se ha mencionado anteriormente, UML es una potente herramienta para especificar un sistema, lo que implica construir modelos precisos, no ambiguos y completos. En particular, UML cubre la especificación de todas las decisiones de análisis, diseño e implementación que deben realizarse al desarrollar y desplegar un sistema con gran cantidad de software. En relación a la propiedad de UML para construir, no se debe entender como que se trata de un lenguaje de programación visual, sino más bien que sus modelos pueden conectarse de forma directa a una gran cantidad de lenguajes de programación. Esta correspondencia permite la ingeniería directa, es decir, la generación de código a partir de un modelo UML en un lenguaje de programación. Lo contrario, conocido como ingeniería inversa, también es posible: se puede reconstruir un modelo UML a partir de una implementación.

Uno de los aspectos principales que han provocado el uso de UML en este proyecto es por su capacidad de documentar un sistema software. Una organización de software que funcione bien produce toda clase de artefactos[1], además de código ejecutable. Estos artefactos incluyen (aunque no se limitan a): requisitos, arquitectura, diseño, código fuente, planificación de proyectos, pruebas, prototipos, versiones y documentos.

[1] Pieza discreta de información que es utilizada o producida por un proceso de desarrollo de software o un sistema existente.

4.4.2. DIAGRAMAS EN UML

Un diagrama es la representación gráfica de un conjunto de elementos, visualizado la mayoría de las veces como un grafo de nodos (elementos) y arcos (relaciones). Los diagramas se dibujan para visualizar un sistema desde diferentes perspectivas, de forma que un diagrama es una proyección de un sistema. En teoría, un diagrama puede contener cualquier combinación de elementos y relaciones. En la práctica, sólo surge un pequeño número de combinaciones habituales, las cuales son consistentes con las vistas más útiles que comprenden la arquitectura de un sistema con gran cantidad de software. Por esta razón, UML incluye trece tipos de diagramas, de los cuales sólo un subconjunto de ellos se ha considerado oportuno de utilizar para dar una vista global de nuestro sistema.

Cuando se modelan sistemas reales, sea cual sea el dominio del problema, muchas veces se dibujan los mismos tipos de diagramas, porque representan vistas frecuentes de modelos habituales. Las partes estáticas del sistema se representarán mediante los diagramas de clases y de casos de uso, mientras que las partes dinámicas del mismo van a representarse con los diagramas de secuencia, de máquinas de estados y de actividades. A los diagramas que son capaces de visualizar, especificar, construir y documentar los aspectos estáticos de un sistema también se les denomina **diagramas estructurales**. A los diagramas que son capaces de visualizar, especificar, construir y documentar los aspectos dinámicos de un sistema también se les denomina **diagramas de comportamiento**. Hay limitaciones prácticas obvias para describir algo inherentemente dinámico (el comportamiento de un sistema) a través de diagramas. Al dibujarse sobre una pantalla de ordenador, hay posibilidades de animar los diagramas de comportamiento para que simulen un sistema ejecutable o reproduzcan el comportamiento real de un sistema en ejecución, algo que no se podría conseguir en un diagrama sobre papel.

4.4.3. MODELADO DE DIFERENTES VISTAS DE UN SISTEMA

Cuando se modela un sistema desde diferentes vistas, de hecho de está construyendo el sistema simultáneamente desde múltiples dimensiones. Eligiendo un conjunto apropiado de vistas, se establece un proceso que obliga a plantearse buenas preguntas sobre el sistema y a identificar los riesgos que hay que afrontar. Si se eligen mal las vistas o si uno se concentra en una vista a expensas de las otras, se corre el riesgo de ocultar preguntas y demorar problemas que finalmente acabarán con cualquier posibilidad de éxito. Las cinco vistas para el modelado de un sistema son: vista de casos de uso, vista de diseño, vista de

interacción, vista de implementación y vista de despliegue. Para modelar un sistema desde diferentes vistas, es necesario:

- Decidir qué vistas se necesitan para expresar mejor la arquitectura del sistema e identificar los riesgos técnicos del proyecto.
- Para cada una de estas vistas, decidir qué artefactos hay que crear para capturar los detalles esenciales. La mayoría de las veces, estos artefactos consistirán en varios diagramas UML.
- Como parte del proceso de planificación, decidir a cuáles de estos diagramas se pondrá algún tipo de control formal o semiformal. Éstos son los diagramas para los que se planificarán revisiones y se conservarán como documentación del proyecto.
- Tener en cuenta que habrá diagramas que se desecharán. Esos diagramas transitorios aún serán útiles para explorar las implicaciones de las decisiones y para experimentar con los cambios.

Si el sistema es reactivo, o si se centra en el flujo de procesos, como es el caso de la aplicación bajo estudio desarrollada en este proyecto, además de la vista de casos de uso, basada en los diagramas de casos de uso, y las vistas de diseño e implementación, basadas ambas en los diagramas de clases, se deben incluir los diagramas de estados y de actividades, para modelar el comportamiento del sistema, los cuales constituyen la vista de interacción.

4.5. MODELADO DE LA APLICACIÓN

En este apartado se van a ir comentando los diagramas UML utilizados para las fases de diseño y desarrollo de la aplicación. Antes de insertar cada tipo de diagramas, se va a presentar una breve explicación de las características más relevantes del tipo en particular.

4.5.1. MODELADO DEL ANÁLISIS

4.5.1.1. Diagramas de Casos de Uso

Un diagrama de casos de uso muestra un conjunto de casos de uso y actores (un tipo especial de clases) y sus relaciones. Los diagramas de casos de uso cubren la vista de casos de usos estática de un sistema. Estos diagramas son especialmente importantes en el modelado y organización del comportamiento de un sistema.

Un caso de uso especifica el comportamiento de un sistema o de una parte de éste, y es una descripción de un conjunto de secuencias de acciones, incluyendo variantes, que ejecuta un sistema para producir un resultado observable de valor para un actor. Los casos de uso se emplean para capturar el comportamiento deseado del sistema en desarrollo, sin tener que especificar cómo se implementa ese comportamiento. Los casos de uso proporcionan un medio para que los desarrolladores, los usuarios finales del sistema y los expertos del dominio lleguen a una comprensión común del sistema. Lo más importante es que permite que los usuarios finales y los expertos del dominio se comuniquen con los desarrolladores (quienes construyen sistemas que satisfacen sus requisitos) sin quedarse atascados en los detalles. Estos detalles llegarán, pero los casos de uso permiten centrarse en las cuestiones más importantes para el usuario final. Además, los casos de uso ayudan a validar la arquitectura y a verificar el sistema mientras evoluciona a lo largo del desarrollo. Los casos de uso bien estructurados denotan sólo comportamientos esenciales del sistema o de un subsistema, y nunca deben ser excesivamente genéricos ni demasiado específicos.

A nivel del sistema, un caso de uso describe un conjunto de secuencias, donde cada secuencia representa la interacción de los elementos externos al sistema (sus actores) con el propio sistema. En realidad, estos comportamientos son funciones a nivel de sistema que se utilizan durante la captura de requisitos y el análisis para visualizar, especificar y documentar el comportamiento esperado del sistema. Un caso de uso representa un requisito funcional del sistema global. Un caso de uso involucra la interacción de actores y el sistema u otros sujetos. Un actor representa un conjunto coherente de roles que juegan los usuarios de los casos de uso al interactuar con éstos.

El nombre de un caso de uso puede constar de texto con cualquier número de letras, números y la mayoría de los signos de puntuación y puede extenderse a lo largo de varias líneas. En la práctica, los nombres de los casos de uso son expresiones verbales que describen algún comportamiento del vocabulario del sistema que se está modelando.

Los casos de uso también pueden organizarse especificando relaciones de generalización, inclusión y extensión entre ellos, que se utilizan para factorizar el comportamiento común (extrayendo ese comportamiento de los casos de uso en los que se incluye) y para factorizar variantes (poniendo ese comportamiento en otros casos de uso que lo extienden):

- La relación de generalización entre casos de uso es como la generalización entre clases. Aquí significa que el caso de uso hijo hereda el comportamiento y el significado del

caso de uso padre; el hijo puede añadir o redefinir el comportamiento del padre; el hijo puede ser colocado en cualquier lugar donde aparezca el padre. Gráficamente, se utiliza una línea dirigida continua, con una gran punta de flecha vacía, apuntando al caso de uso padre.

- La relación de inclusión entre casos de uso significa que un caso de uso base incorpora explícitamente el comportamiento de otro caso de uso en el lugar especificado por el caso base. La relación de inclusión se usa para evitar describir el mismo flujo de eventos repetidas veces, poniendo el comportamiento común en un caso de uso aparte. El caso de uso que incluye no puede ser entendido por completo en ausencia del caso incluido, puesto que forma parte de la funcionalidad del mismo. Una relación de inclusión se representa como una dependencia, utiliza uno de los estereotipos[2] más extendidos, como es el caso de <<*include*>> y se denota como una relación que apunta del caso que incluye al caso incluido utilizando una línea discontinua.
- La relación de extensión entre casos de uso significa que un caso de uso base incorpora implícitamente el comportamiento de otro caso de uso en el lugar especificado indirectamente por el caso de uso que extiende al base. Este caso de uso base puede extenderse sólo en ciertos puntos, llamados, como era de esperar, puntos de extensión. La extensión se puede ver como que el caso de uso que extiende incorpora su comportamiento en el caso de uso base. Una relación de extensión se utiliza para modelar una parte de un caso de uso que el usuario puede ver como el comportamiento opcional del sistema, es decir, agrega funcionalidad al caso base pero sin alterar su funcionalidad. De esta forma, se separa el comportamiento opcional del obligatorio. Una relación de extensión se representa como una dependencia, estereotipada con <<*extend*>>, y se denota como una relación que apunta del caso extendido al caso base utilizando una línea discontinua.

A continuación se van a ir explicando los seis diagramas de casos de uso que plasman una primera aproximación a la funcionalidad principal que debe ofrecer la aplicación desarrollada en el proyecto, sin especificar detalles de la misma. Cada diagrama va a contener todos aquellos casos de uso que estén directamente relacionados por representar un mismo rasgo o característica generalizada, en concordancia con los diferentes grupos de requisitos planteados según la funcionalidad.

[2] Extensión del vocabulario de UML que permite crear nuevos bloques de construcción derivados a partir de los existentes pero específicos a un problema concreto.

4.5.1.1.1. Carga de datos de la simulación

En el diagrama de casos de uso de la figura 4.1 se engloban los requisitos de usuario y de sistema pertenecientes al grupo "Requisitos de la carga de ficheros". Se pueden apreciar, en concreto, cuatro casos de uso, tres de los cuales son incluidos explícitamente en el caso de uso denominado "Cargar archivos de simulación". Esto se representa así porque la funcionalidad asociada a los casos de uso "Comprobar integridad datos", "Cargar datos desde *.txt*" y "Cargar datos desde *.pos*" se enmarca en la funcionalidad del caso de uso "Cargar archivos de simulación", de forma que este último no se podría entender sin los anteriores. El uso de este tipo de estereotipos ayuda a simplificar una funcionalidad en otras más simples y que tienen significado por sí mismas. Las acciones que se suceden dentro del caso de uso "Cargar archivos de simulación" son las siguientes: apertura de ventana emergente para la selección de archivo de salida de simulación (*.txt*) o de movilidad de los nodos (*.pos*), apertura y carga de datos del archivo seleccionado, lo que vendría representado por los casos de uso "Cargar datos desde *.txt*" y "Cargar datos desde *.pos*" y, finalmente, la verificación de los datos leídos desde los archivos, funcionalidad definida en el caso de uso "Comprobar integridad datos".

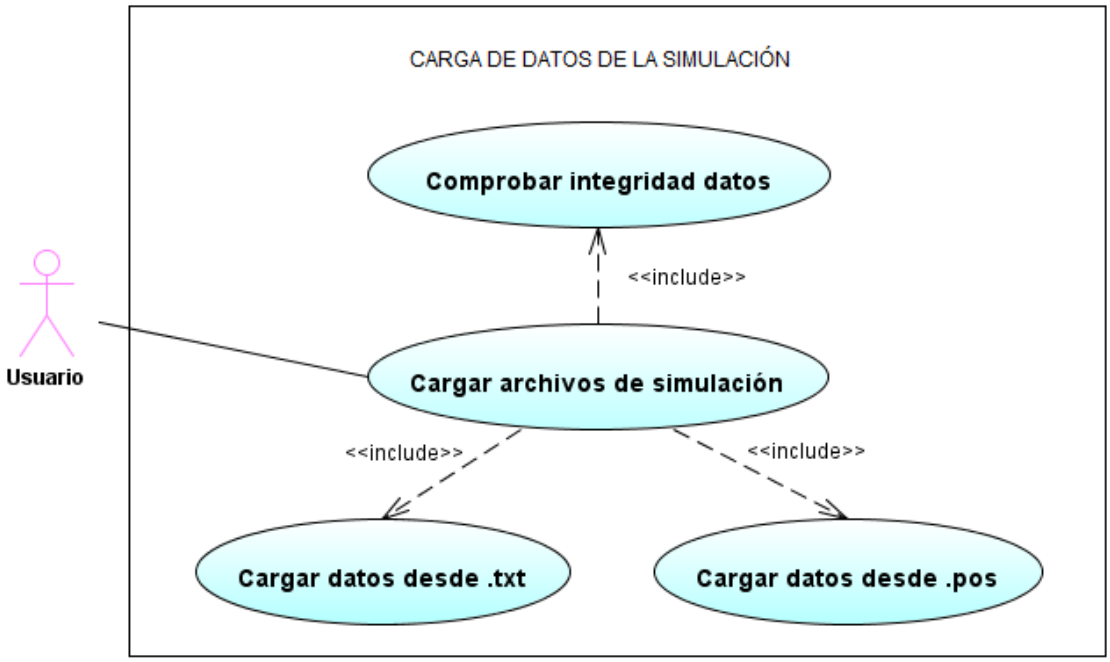


Figura 4.1. Diagrama de casos de uso - Carga de datos de la simulación

En este diagrama de casos de uso, así como en los que se presentarán en los siguientes apartados, se considerará un único actor del sistema, que será el usuario final de la aplicación. Éste será el único encargado de interactuar con el software.

4.5.1.1.2. Creación del escenario

Los requisitos de usuario y de sistema relacionados con la carga del escenario se pueden agrupar en los tres casos de uso que aparecen en la figura 4.2, dos de los cuales se encuentran explícitamente formando parte de otro caso de uso. Así, el caso de uso "Habilitar controles" incluye la funcionalidad de forzar a que los controles presentados en la interfaz de la aplicación se encuentren disponibles, actualizando los extremos y los valores intermedios de la barra horizontal del avance de la animación o *slider* temporal a partir de la duración de la simulación. Este valor será tomado del archivo de movilidad de los nodos (*.pos*) cuando sea el único archivo cargado o bien de ambos archivos de entrada si se comienza cargando el archivo de salida de la simulación (*.txt*). A su vez, este caso de uso formará parte explícitamente del caso de uso "Crear escenario", que además incluirá las acciones relativas a la creación de los nodos del escenario, partiendo del valor del número de nodos que se tomará de la misma forma que la duración de la simulación, y al posicionamiento inicial de los mismos en las coordenadas virtuales. Finalmente, este último caso de uso será parte del caso de uso "Cargar datos de la simulación", englobando a todas las funciones mencionadas en este apartado, y añadiendo la mezcla de los eventos que se hayan obtenido de los archivos de entrada de la aplicación.

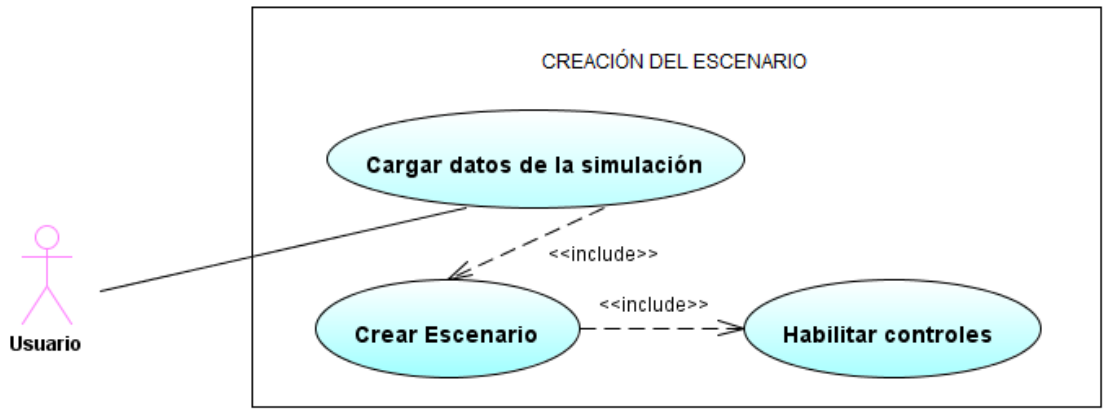


Figura 4.2. Diagrama de casos de uso - Creación del escenario

4.5.1.1.3. Opciones de visualización de la aplicación

En el diagrama de casos de uso de la figura 4.3 se representan algunas de las opciones de visualización del escenario gráfico, en concordancia con los requisitos planteados en el grupo "Requisitos de la navegación de la animación". Además, se utiliza el estereotipo *<<extend>>* indicando que las funcionalidades de los casos de uso que extienden podrían, opcionalmente, incorporar su funcionalidad en el caso base. Así, el caso de uso base "Ir hacia adelante", representa el avance temporal hacia un instante superior al instante actual

en que se encuentre la animación y, opcionalmente, se podría realizar con los casos de uso que extienden de él: “Ir hacia adelante por pasos temporales”, que representa el avance incremental a partir de un valor introducido en la aplicación; “Ir hacia delante de forma absoluta”, que establece un avance hacia el instante absoluto introducido en la aplicación; o “Desplazar animación con el eje temporal”, que implica el avance en el tiempo por el desplazamiento de la barra horizontal o *slider* hacia un instante posterior al actual.

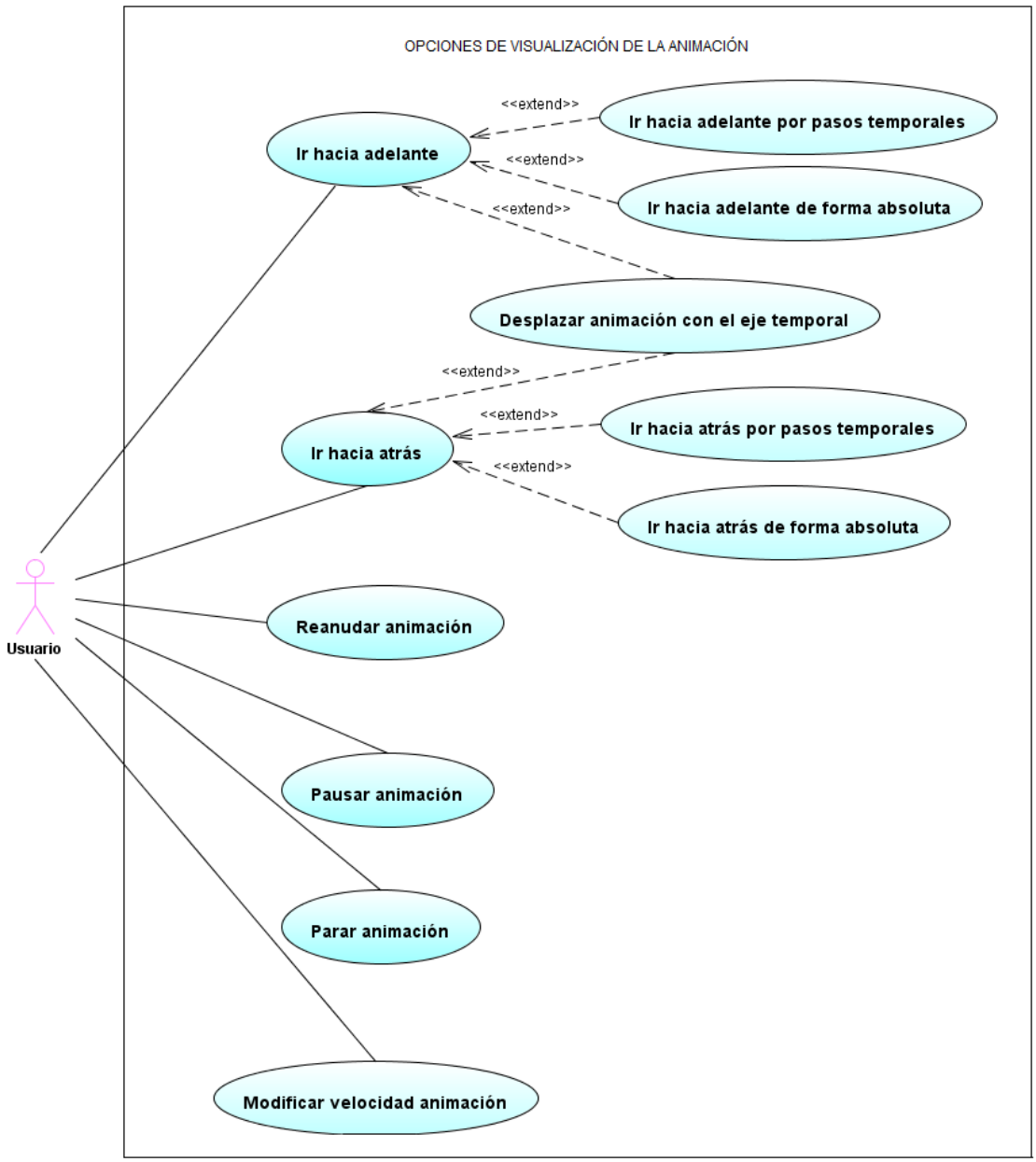


Figura 4.3. Diagrama de casos de uso - Opciones de visualización de la aplicación

De la misma forma, se establece el caso de uso base “Ir hacia atrás”, a partir del cual se podrían extender otros tres casos de usos similares a los del caso de uso “Ir hacia

adelante", aportando las opciones que se podrían tener para cumplir su funcionalidad. El caso de uso "Reanudar animación" especifica el requisito de arrancar la animación una vez que se ha pausado, siguiendo a partir del mismo instante de la pausa. El caso de uso "Pausar animación" implica que la animación se pueda retener indefinidamente en un determinado instante de tiempo, hasta que se produzca otro determinado evento en la aplicación, como la reanudación de la animación. El caso de uso "Parar animación" genera aquel escenario en el que el usuario decide parar la animación y llevarla a su instante inicial. Y por último, el caso de uso "Modificar velocidad animación" establece el escenario para verificar el cumplimiento del cambio de la velocidad de la animación por una acción del usuario, ya sea teniendo la animación en estado pausado como en estado activo.

4.5.1.1.4. Opciones de vista del escenario

En el diagrama de la figura 4.4 se presentan los casos de uso generados para establecer la funcionalidad de los requisitos del grupo "Requisitos de la vista del escenario". En este diagrama vuelven a aparecer algunos casos de uso que deben incluirse explícitamente como parte del comportamiento de los casos de uso base, indicando que el flujo de acciones del caso de uso base contiene al flujo de acciones del caso de uso incluido.

El caso de uso "Marcar nodo seleccionado para estudio" implica que el usuario ha elegido un nodo para que sea tenido en cuenta a la hora de mostrar sus estadísticos o de moverse por la animación atendiendo a sus eventos. Las acciones contenidas en este caso de uso se deben realizar siempre que el usuario seleccione un nodo de alguna de las maneras posibles, lo que viene representado en el caso de uso base "Seleccionar nodo". El caso de uso "Cambiar apariencia del Escenario" contiene una serie de funciones que se deben ejecutar al darse el escenario "Seleccionar nodo" o "Seleccionar Cobertura", e implica que se modifiquen atributos de apariencia de los nodos y de sus coberturas. El caso de uso "Cambiar el radio de los nodos" reproduce el escenario generado cuando el usuario decide modificar el radio de las esferas que representan los nodos en la aplicación. Finalmente, el resto de casos de uso que aparecen en el diagrama de la figura 4.4 implican el movimiento del punto de vista, o más concretamente, de la situación de la cámara a través de la cual se observa el escenario, realizando movimientos hacia arriba, abajo, derecha e izquierda, así como la aproximación/alejamiento a través del *zoom*.

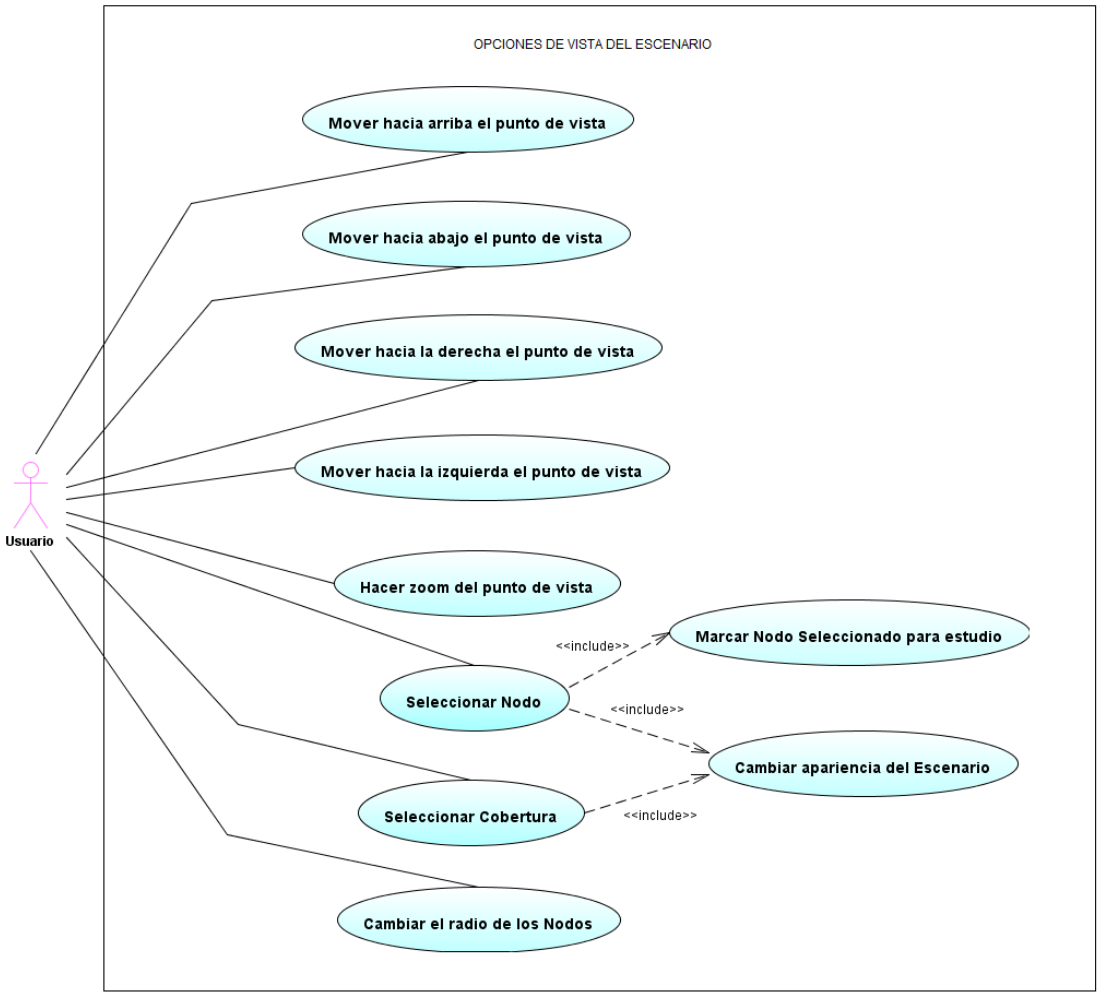


Figura 4.4. Diagrama de casos de uso - Opciones de vista del escenario

4.5.1.1.5. Consulta de la caché local y de la caché de redirecciones

Los requisitos de los grupos de "Requisitos del estudio de la caché local" y "Requisitos del estudio de la caché de redirecciones" se convierten, en la primera fase de análisis del proyecto software, en los casos de uso que aparecen en el diagrama de la figura 4.5. Con respecto al estudio de la caché local, la aplicación ofrecerá la posibilidad de mostrar la caché para un determinado nodo en cualquier instante de tiempo creando el escenario definido en el caso de uso "Solicitar Caché para un Nodo en un instante de tiempo". Esta caché local aparecerá representada a modo de tabla ordenada por filas según las posiciones de las entradas. Además, el escenario del caso de uso "Buscar información de un documento dentro de una caché" podrá ofrecer el encontrar una determinada entrada de caché a partir del identificador del documento. Para el estudio de la caché de redirecciones ocurre algo parecido, presentando dos casos de uso para la consulta y la búsqueda de información, con la única diferencia de que el contenido de ésta vendrá ordenado según el

identificador numérico, puesto que no se ha definido aún una política de reemplazo para ella que le dé importancia a la posición ocupada.

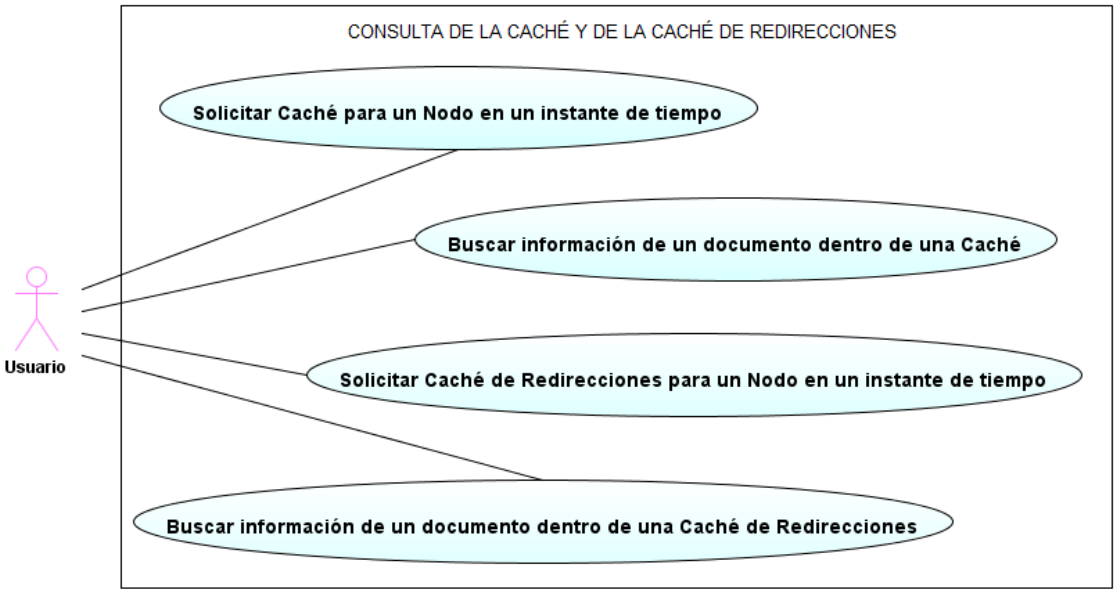


Figura 4.5. Diagrama de casos de uso - Consulta de la caché local y de la caché de redirecciones

4.5.1.1.6. Cálculo de estadísticos

Como último grupo de funcionalidades diferentes ofrecidas por la aplicación se presenta el diagrama de casos de uso de la figura 4.6, que contiene aquellos casos de uso acordes a los requisitos de usuario y de sistema del grupo "Requisitos del estudio de estadísticos".

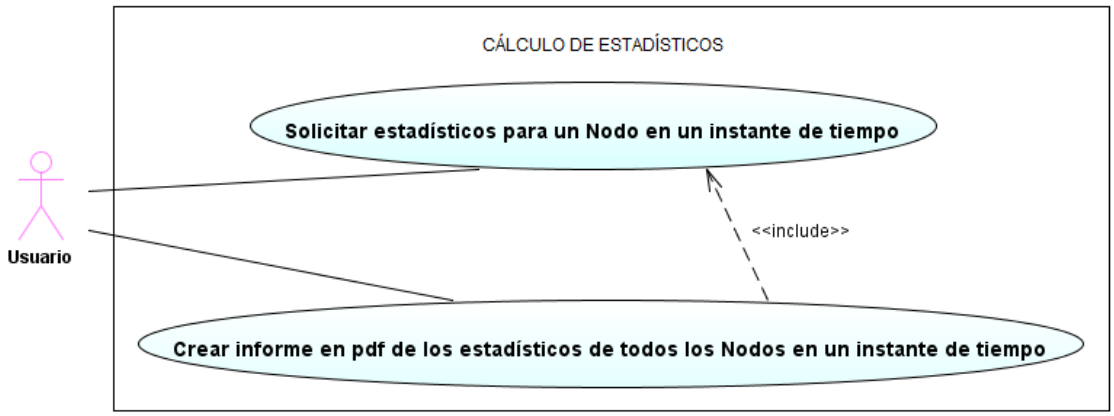


Figura 4.6. Diagrama de casos de uso - Cálculo de estadísticos

En primer lugar, el caso de uso "Solicitar estadísticos para un Nodo en un instante de tiempo" implica el cálculo de los valores estadísticos de un nodo válidos hasta el instante

de tiempo actual que tenga la animación. En segundo lugar, el caso de uso "Crear informe en PDF de los estadísticos de todos los Nodos en un instante de tiempo" incluye al anterior porque, durante la creación del informe, debe recopilar la información de todos los valores estadísticos válidos para todos los nodos del escenario. Además, se encargará de tomar valores medios y representar gráficamente esta información con el uso de histogramas.

4.5.1.2. Diagramas de Estados

Un diagrama de estados muestra una máquina de estados, que consta de estados, transiciones, eventos y actividades. Da una idea, por tanto, de la vista dinámica de un objeto o conjunto de objetos. Son especialmente importantes en el modelado del comportamiento de una interfaz, una clase o una colaboración y resaltan el comportamiento dirigido por eventos de un objeto, lo cual es especialmente útil en el modelado de sistemas reactivos.

Un estado es una condición o situación en la vida de un objeto durante la cual satisface alguna condición, realiza alguna actividad o espera algún evento. Un objeto permanece en un estado durante una cantidad de tiempo finita. El nombre de un estado puede ser texto formado por cualquier número de letras, dígitos y ciertos signos de puntuación y puede extenderse a lo largo de varias líneas. En la práctica, los nombres de estados son nombres cortos o expresiones nominales extraídos del vocabulario del sistema que se está modelando.

Una transición es una relación entre dos estados que indica que un objeto que esté en el primer estado realizará ciertas acciones y entrará en el segundo estado cuando ocurra un evento especificado y se satisfagan unas condiciones especificadas. Una transición tiene cinco partes: el estado origen, que es el estado afectado por la transición; el evento de disparo, el evento cuya recepción por el objeto que está en el estado origen provoca el disparo de la transición si se satisface su condición de guarda; la condición de guarda, una expresión booleana que se evalúa cuando la transición se activa por la recepción del evento de disparo, indicando por tanto si se puede o no realizar el efecto; efecto, un comportamiento ejecutable, como una acción; y el estado destino, estado activo tras completarse la transición. En el diagrama de estados que se ha adjuntado explicitando los distintos eventos de la visualización de la animación, no se han indicado los eventos de disparo porque las acciones por sí solas dan una idea del evento producido. Tampoco se han añadido condiciones de guarda porque el comportamiento asociado a dichos eventos se detalla con los diagramas de actividad de la aplicación. Además no se ha estimado

necesario realizar ningún otro diagrama de estados respecto a otras actividades de la aplicación porque no constituyen un comportamiento dinámico. Los detalles del diagrama de estados con los eventos de la animación se van a mostrar a continuación.

En esta fase de análisis de la aplicación del proyecto, sin tener claramente establecidos cada uno de los objetos que se tendrán tras la fase de desarrollo del software, se ha pretendido explicar el comportamiento dinámico que tendrá la aplicación en sí. De esta forma, se indica que la aplicación o, mejor dicho, el software asociado a ella representando el conjunto de objetos que finalmente se encargarán de la animación, estará en estado ocioso o *idle* mientras no ocurra alguno de los eventos mostrados en el diagrama de la figura 4.7.

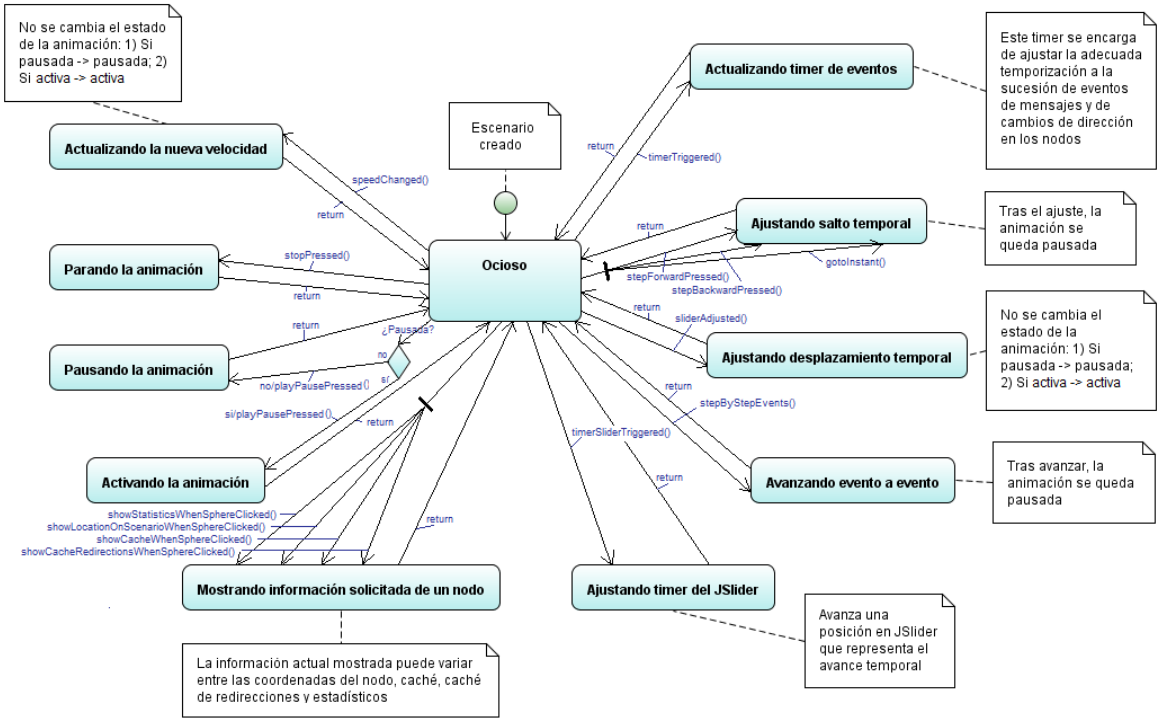


Figura 4.7. Diagrama de estados - Eventos de la animación

El diagrama parte del estado inicial de creación satisfactoria del escenario gráfico, y el manejador de eventos, asociado a la animación gráfica, va a responder ante cada uno de los eventos presentes en el diagrama y actuando de forma parecida, ya que cuando se finalice el código a ejecutar tras la detección de un evento, va a volver al estado ocioso. Es más, para cada una de las transiciones que aparecen en el diagrama de estados, el estado origen siempre va a coincidir con el estado destino, que a su vez será el mismo estado ocioso. De esta forma se puede representar de forma gráfica y sencilla una primera aproximación a la solución que será implementada durante la fase de diseño. El software se planteará de

forma que estos eventos no interrumpan a otros eventos, finalizando el código asociado a uno antes de atender al otro. Como se comentó anteriormente, debido a los nombres asignados a las acciones que se realizan con cada evento, no se ha estimado necesario añadir el evento de disparo ni la condición de guarda para simplificar el problema. Así, los diferentes eventos de disparo que se han recogido en el diagrama son los siguientes:

- Evento del temporizador global encargado de gestionar los diferentes eventos ordenados temporalmente procedentes del archivo de movilidad de los nodos y, en su caso, del archivo de salida de la simulación. Con este temporizador se puede ir avanzando en la animación a la velocidad de visualización que se haya elegido de forma que cada vez que exista un nuevo evento en un nodo, ya sea un cambio de dirección o el envío o recepción de un mensaje, se interprete correctamente y realice las funciones oportunas, como mostrar los estadísticos de ese nodo en la zona de la aplicación destinada a ello. Se logra adecuar a la correcta temporización la sucesión de los diferentes eventos que ocurran en los nodos.
- Evento de cambio del instante temporal, ya sea de alguna de las tres maneras presentadas (por acciones en los controles de avance por saltos temporales, de retroceso por saltos temporales, o de avance/retroceso por instantes absolutos). El estado al que se dirige tras el evento se ocupa de ajustar los valores de la animación al nuevo valor temporal, desplazando los nodos a las posiciones que deberían ocupar. Hay que indicar que tras este tipo de eventos, la animación permanece pausada.
- Evento de desplazamiento por el usuario de la barra horizontal o *slider* temporal, de forma que se dirige hacia un estado en el que se ajustan los valores de la animación y las posiciones de los nodos al instante marcado por el ítem temporal. Hay que indicar que tras volver al estado ocioso, la animación presentará la misma característica que antes de producirse el evento de disparo, esto es, pausada si estaba pausada o activa si se encontraba activa.
- Evento de avance o retroceso "por eventos", es decir, avanzando o retrocediendo en la animación por saltos temporales que vienen dados por la diferencia de tiempo entre dos eventos sucesivos (ya sean cambios de dirección o de mensajes) de nodos, para aquellos nodos que se encuentren seleccionados o simplemente, para cualquier nodo. Se trata de ajustar el instante de tiempo en el que se debe llevar la animación, interpretando cada uno de los mensajes que vayan sucediendo y parándose únicamente en aquellos que pertenezcan a los nodos especificados. En este caso también se queda la animación pausada tras un evento de este tipo.

- Evento del temporizador asociado a la barra horizontal de desplazamiento o *slider* temporal, puesto que la velocidad a la que avanza el propio *slider* indicando el instante en el que se encuentra la animación en cada momento, se debe variar atendiendo a la velocidad de visualización que se tenga seleccionada.
- Evento de muestra de información que puede darse debido a diversos eventos de disparo: solicitud de información de estadísticos de un nodo, solicitud de información acerca de las coordenadas físicas del escenario real que ocupa un nodo, solicitud de información acerca del contenido de la caché local, y solicitud de información acerca del contenido de la caché de redirecciones.
- Evento tras accionar el control para la activación o pausa de la animación, ya que se trata de un evento de disparo común para ambas acciones. En función de la condición de guarda acerca de si la animación se encuentra pausada o no, se dirigirá hacia un estado de reanudación o de pausa de la animación, según sea el caso.
- Evento del control asociado a la parada de la animación, dirigiéndose hacia el estado encargado de pausar la animación y llevar todos sus parámetros al inicio de la animación, incluyendo el posicionamiento de los nodos en sus correspondientes puntos iniciales.
- Evento de cambio del ratio entre las velocidades de la animación y de la simulación de base. En este caso, el estado se encarga de actualizar la información de interés para la animación y la deja en la misma forma en la que estaba antes de producirse el evento de disparo, es decir, pausada o activa.

4.5.2.MODELADO DEL DISEÑO

4.5.2.1. Diagramas de Secuencia

Los diagramas de secuencia forman parte de los llamados diagramas de interacción, donde se muestra consecuentemente una interacción, que consta de un conjunto de objetos o roles, incluyendo los mensajes que pueden ser enviados entre ellos. Los diagramas de interacción ofrecen información de la vista dinámica de un sistema. Un diagrama de secuencia es un diagrama de interacción que resalta la ordenación temporal de los mensajes. Un diagrama de secuencia se forma colocando en primer lugar los objetos o roles que participan en la interacción en la parte superior del diagrama, a lo largo del eje horizontal. Normalmente, se coloca a la izquierda el objeto o rol que inicia la interacción, y los objetos o roles subordinados a la derecha. A continuación, se colocan los mensajes que estos objetos envían y reciben a lo largo del eje vertical, en orden de sucesión en el tiempo,

de arriba abajo. Así, se puede distinguir claramente el flujo de control a lo largo del tiempo.

Los diagramas de secuencia tienen dos características que los distinguen de otros diagramas de comunicación, como son la línea de vida y el foco de control del objeto. La línea de vida de un objeto es la línea discontinua vertical que representa la existencia de un objeto a lo largo de un período de tiempo. El foco de control es un rectángulo estrecho que representa el período de tiempo durante el cual un objeto ejecuta una acción, bien sea directamente o bien sea a través de un procedimiento. En la parte inferior del rectángulo se puede marcar adicionalmente con un aspa el fin de la vida de los objetos, como se ha representado en uno de los diagramas de secuencia realizados para la aplicación.

El contenido principal de los diagramas de secuencia son los mensajes. Un mensaje se representa con una flecha que va de una línea de vida hasta la otra. La flecha apunta al receptor. Si el mensaje es asíncrono, la flecha es abierta; si es síncrono (una llamada), la flecha es un triángulo relleno, como todos los mensajes enviados en los diagramas de secuencia adjuntos de la aplicación. Una respuesta a un mensaje síncrono (el retorno de una llamada, que podría omitirse) se representa como una línea discontinua.

De los aspectos dinámicos del comportamiento del software que se pueden representar con diagramas de secuencia y que resultan de mayor interés para el entendimiento del diseño realizado, se han tomado las fases de arranque de la aplicación y de creación de los controladores (atendiendo al MVC) por la ordenación de las acciones que se llevan a cabo. De hecho, una vez que la ventana principal de la aplicación se ha creado, el resto de acciones que se pueden producir no siguen un orden estricto. Todas estas acciones no presentadas aquí se explicarán con más detalle en los diagramas de actividad en el siguiente epígrafe.

4.5.2.1.1. Arranque de la aplicación

En la figura 4.8 se observan las primeras acciones que se llevan a cabo tras el arranque de la aplicación.

El usuario, el primer objeto o rol que aparece a la izquierda del diagrama, abre la aplicación y crea el primer objeto de la misma, una instancia de la clase *Main*. Este objeto se va a encargar de mostrar la imagen inicial con el logo de la aplicación antes de cargar los componentes de la ventana, imagen que va sobre una ventana emergente o *Splash Window*. Además, va a crear un temporizador con un retardo de pocos segundos y su código asociado (vía el controlador *InitialTimerListener*) para que el *Splash Window*

permanezca visible unos instantes más. Cuando este temporizador se agote, se activará dicho código asociado a través del mensaje *actionPerformed*, que es el que finalmente llamará al constructor de la ventana principal de la aplicación. A partir de este momento, todos los objetos excepto la ventana principal, que es la instancia de la clase *ApplicationWindow*, se destruirán, quedando únicamente activo este último. El intercambio de mensajes que se produce en el constructor de la ventana principal se muestra en el siguiente diagrama de secuencia.

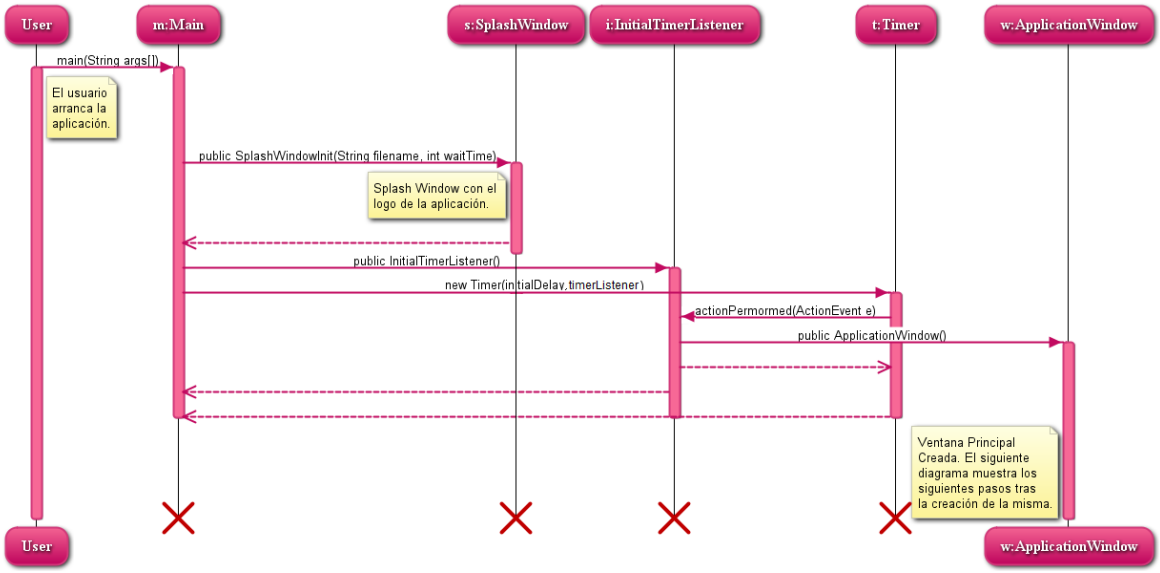


Figura 4.8. Diagrama de secuencia - Arranque de la aplicación

4.5.2.1.2. Creación de la ventana principal de la aplicación y de su controlador genérico

Como se acaba de introducir, en el diagrama de secuencia de la figura 4.9 se muestra la sucesión de acciones llevadas a cabo para la creación de la ventana principal de la aplicación, de sus componentes y del controlador genérico, que no contendrá directamente el código asociado a las acciones que se deban realizar en respuesta a los eventos que ocurran. Este controlador genérico se encarga de redirigir hacia otro controlador apropiado el evento en cuestión. Esos otros controladores sí que contienen el código con las acciones a llevar a cabo en respuesta a los eventos de la aplicación en concordancia con el tipo y funcionalidad de los mismos.

A la izquierda del diagrama se muestra como primer objeto una instancia de la clase *ApplicationWindow*, como continuación del contenido del constructor presentado en el diagrama de secuencia 4.8, y sólo se tendrá una instancia de este objeto durante el funcionamiento de la aplicación, como pasa como las instancias de los diversos

controladores. Los cinco primeros mensajes corresponden a llamadas de métodos propios del objeto, por eso tienen esa forma apuntada hacia él mismo. Estas llamadas añaden el título de la ventana y el icono de la aplicación, fijan el tamaño mínimo de la ventana, y crean todos los componentes que deben aparecer en la misma. A continuación se realiza la llamada al constructor de la clase *ApplicationInformation*, creando una instancia de él, y también se creará únicamente un objeto de este tipo. Este constructor continuará como el inicio del siguiente diagrama de secuencia. Esta clase es, si cabe, la más importante, puesto que contiene toda la información que maneja la aplicación, desde parámetros de la animación, hasta colecciones con los datos de los nodos relativos a sus estadísticos, su representación gráfica y la de su cobertura, los movimientos por la red, etc. Además contiene las instancias de los demás controladores como atributos de la propia clase, como se mostrará en el diagrama de secuencia 4.10.

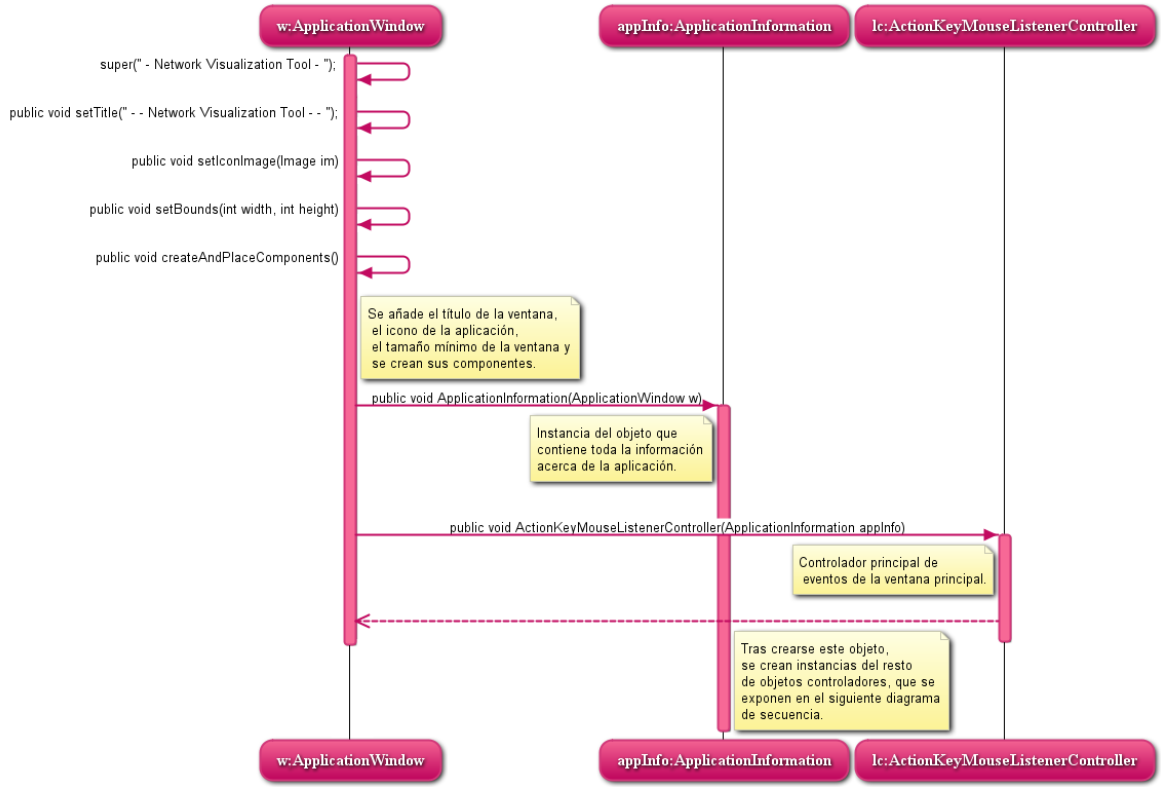


Figura 4.9. Diagrama de secuencia - Creación de la ventana principal de la aplicación y de su controlador genérico

La última acción llevada a cabo en el constructor de la ventana principal de la aplicación es la creación del manejador de eventos o controlador genérico que se encargará de redirigir los eventos que se creen debido a los componentes de la ventana hacia el controlador más adecuado. Esta clase controladora genérica se denomina

ActionKeyMouseListenerController, debido a la multitud de interfaces definidas para Java que implementa.

Es importante destacar que la instancia de la clase *ApplicationInformation* se crea desde el constructor de la ventana principal. Para ello, en la llamada del constructor se le pasa como parámetro la propia ventana para que lo almacene como atributo. Se realiza de esta manera porque, como se ha introducido anteriormente, la clase *ApplicationInformation* contiene toda la información de interés para la animación, además de que contiene como atributos a los controladores por funcionalidades (controladores que se detallarán en los diagramas de actividad y de clases). Estos controladores podrán acceder a cualquier dato necesario para la animación y la visualización del escenario gráfico, y podrán leer el estado de los componentes de la ventana principal y modificarlos únicamente a través de la instancia de *ApplicationInformation*, disminuyendo así el acoplamiento entre las clases.

4.5.2.1.3. Creación de los controladores auxiliares

Como ya se ha introducido en el apartado anterior, en el diagrama de secuencia presentado en la figura 4.10 aparecen las acciones llevadas a cabo dentro del constructor de la clase *ApplicationInformation* relativas a la creación de los controladores auxiliares de la aplicación y del escenario gráfico. Estas llamadas, que corresponden a los constructores, tendrán como parámetro a la propia instancia de *ApplicationInformation* por las razones expuestas anteriormente de accesibilidad a los datos.

En primer lugar crea dicho escenario, realizando la llamada a su constructor, que posiciona el *canvas* en el centro de la ventana principal de la aplicación, pero que aún no tiene información suficiente para la creación del árbol que representa el escenario gráfico. A continuación realiza la llamada al constructor de *ScenarioController*, que representa el controlador de los eventos que ocurran relativos a la representación gráfica del escenario. En tercer lugar se crea el objeto controlador para el manejo de los eventos de vayan ocurriendo durante la animación relativos a los temporizadores, saltos temporales, desplazamiento de la barra horizontal o *slider* temporal, cambio de velocidad de animación, etc. Esta clase controladora se denomina *VisualizationController*. El siguiente controlador que crea, denominado *NodeStatisticsController*, es el que se dedicará a la creación de los estadísticos de los nodos cuando sea necesario. Por último, se crea la instancia de la clase controladora dedicada a la apertura y carga de datos de los archivos de entrada de la aplicación, esto es, el archivo de movilidad de los nodos y el archivo de salida de la simulación con el intercambio de mensajes producido. Esta clase controladora se denomina *FileController*.

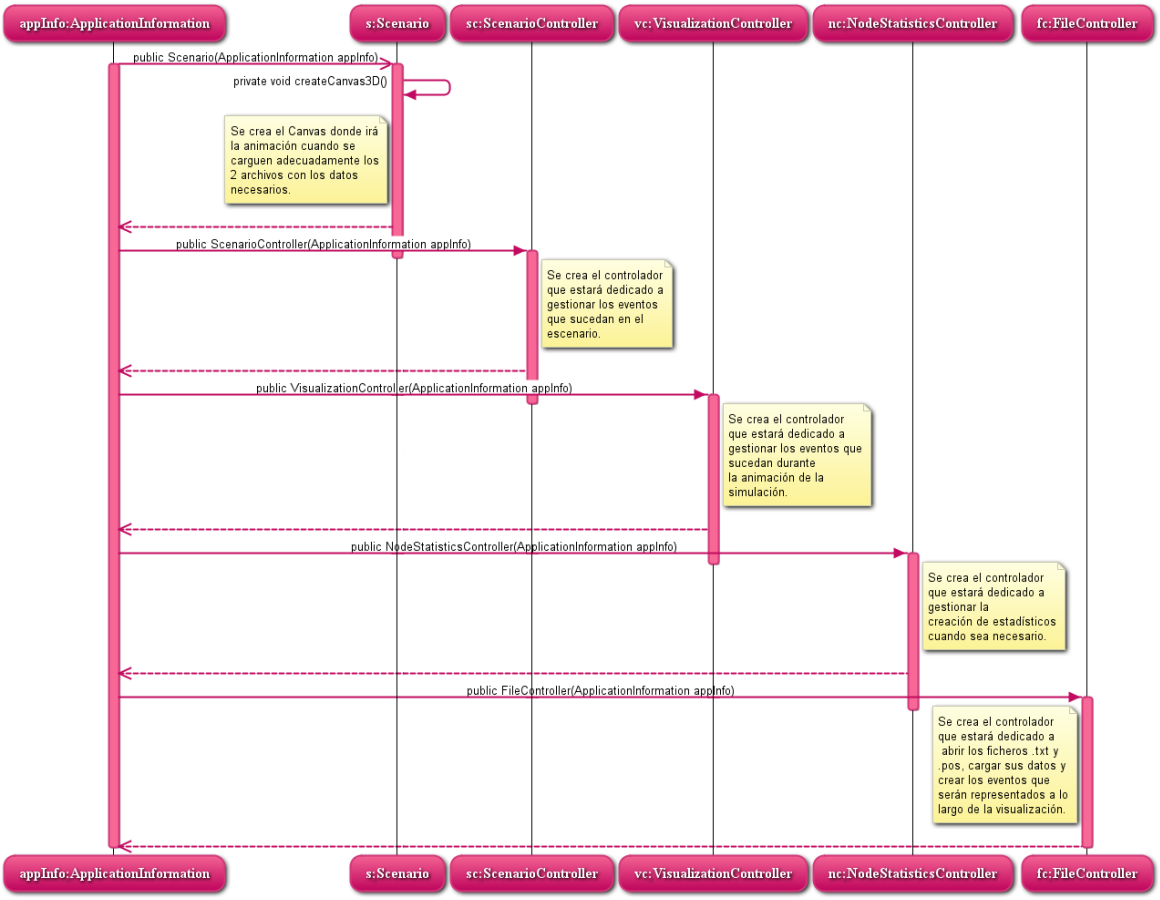


Figura 4.10. Diagrama de secuencia - Creación de los controladores auxiliares

4.5.2.2. Diagramas de Actividades

Un diagrama de actividades muestra la estructura de un proceso como el flujo de control y de datos paso a paso en la computación. Los diagramas de actividades ofrecen una vista dinámica de un sistema. Son especialmente importantes al modelar el funcionamiento de un sistema y resaltan el flujo de control entre objetos. Las actividades producen la ejecución de acciones individuales, cada una de las cuales puede producir un cambio en el estado del sistema o la comunicación de mensajes. Las acciones incluyen llamadas a otras operaciones, envío de señales, creación o destrucción de objetos o simples cálculos, como la evaluación de una expresión. Normalmente, los diagramas de actividad contienen acciones, nodos de actividad, flujos y objetos valor. Un nodo de actividad es una unidad organizativa dentro de una actividad. En general, los nodos de actividad son agrupaciones anidadas de acciones o de otros nodos de actividad. Una actividad es un nodo de actividad que no se puede descomponer más. Si se entra en los detalles de un nodo de actividad se encontrará otro diagrama de actividades. El flujo de control se especifica con flechas que muestran el camino de control de un nodo de actividad o acción al siguiente. En realidad, un flujo de control tiene que empezar y parar en algún sitio. Por lo tanto, se puede

especificar el inicio como un círculo relleno y el final como un círculo relleno dentro de una circunferencia. Los flujos secuenciales son frecuentes, pero no son el único tipo de camino que se necesita para modelar el flujo de control. Como en los diagramas de flujo, se puede incluir una bifurcación, que especifica los caminos alternativos elegidos en función de una expresión booleana.

Una cosa especialmente útil cuando se modelan flujos de trabajo es dividir en grupos a los nodos de actividad de un diagrama de actividad, donde cada uno representa la parte de organización responsable de esas actividades o incluso las instancias de clases que contienen los datos que se están consultando y/o modificando. En los diagramas de actividad de la aplicación que se han realizado, estas particiones corresponden a las instancias de clases que se encargan de realizar dichas actividades, dando una idea de entre qué objetos se reparte el flujo de control. Además, en algunos casos se presentan instancias de las clases que contienen algunos miembros de clase que se están consultando o modificando, para dar más información de las actividades utilizando objetos valor. Las divisiones se han realizado verticalmente, señalando en la parte superior el nombre de la clase cuya instancia realiza las actividades que pertenecen a esa columna. Además, los nodos de actividad van a estar representados con estructuras rectangulares con esquinas redondeadas, a diferencia de los objetos valor, que son completamente rectangulares. Los nodos de actividad tendrán el nombre de la acción que llevan a cabo, y los objetos valor representarán los valores que tomarán algunas de las variables importantes que maneja la aplicación relacionadas con la actividad que se está realizando. Por último, las estructuras en forma de rombo representan las bifurcaciones posibles que podría seguir el flujo de control atendiendo a una serie de condiciones.

Los diagramas de actividad que se han realizado pretenden explicar la vista dinámica del sistema atendiendo a los diferentes eventos que pueden producirse durante la animación y para la carga de archivos, donde se ejecutan continuamente las acciones de los controladores definidos. En todos ellos, el evento que desencadena el flujo de actividades del diagrama es realizado por el usuario, que es el que interactuará con el sistema. Los métodos que contienen el código asociado a las actividades que se llevan a cabo en los diagramas que se presentarán a continuación forman parte del controlador de la visualización de la animación *VisualizationController*, lo que no quita que desde esos métodos se delegue parte de las actividades más concretas a otros controladores más apropiados. No se han considerado de interés para la representación a través de este tipo de

diagramas las acciones realizadas como consecuencia de los eventos relacionados con la vista del escenario.

4.5.2.2.1. Carga de archivos de entrada y creación del escenario

En este apartado se van a explicar las acciones que van a llevar a cabo las instancias de las clases principales de la aplicación y de algunos de los controladores durante la apertura y la carga de información de los archivos de entrada de la aplicación. Estas acciones se representan en el diagrama de actividades de la figura 4.11.

Hay que recordar que la aplicación cargará el escenario representando la movilidad de los nodos si se produce la carga del archivo de movilidad de los nodos en primer lugar, pudiendo cargar o no después el archivo de salida de la simulación. El escenario creado sólo presentará, durante la animación, los eventos relacionados con los cambios de dirección de los nodos, no realizándose ningún intercambio de mensajes, como era de esperar. En el caso de realizar la carga de datos del archivo de salida de la simulación en primer lugar, no se realizará la carga de ningún escenario porque, en este caso, es indispensable la carga del archivo con la movilidad de los nodos. El escenario creado ahora sí representará tanto los eventos relacionados con los cambios de dirección de los nodos como los eventos relacionados con el intercambio de mensajes entre ellos.

Las actividades mostradas en el diagrama van a ser realizadas por las instancias de las clases *ApplicationWindow* y *ApplicationInformation*, clases principales de la aplicación, y *ActionKeyMouseListenerController* y *FileController*, que son los controladores involucrados.

La parte inicial del diagrama se centra en el arranque de la aplicación y la creación de los objetos que se han explicado con los tres diagramas de secuencia presentados anteriormente, es decir, las instancias de *ApplicationWindow*, *ApplicationInformation* y de todos los controladores. Una vez que la aplicación se encuentra en este estado, que es el estado de partida real de este diagrama y que se nombrará más adelante, se espera a la acción del usuario a través del ratón para la carga de algún fichero, o bien para el cierre de la aplicación. El cierre de la aplicación se podría dar en cualquier instante, por eso se ha obviado en el resto de los diagramas de actividades. Una vez que el usuario decide realizar la carga de datos de alguno de los dos archivos, se invoca a la instancia creada de *FileController* desde la instancia de *ActionKeyMouseListenerController* (puesto que este controlador genérico lo único que hace es redirigir hacia uno de los otros controladores más concretos la acción requerida por el evento).

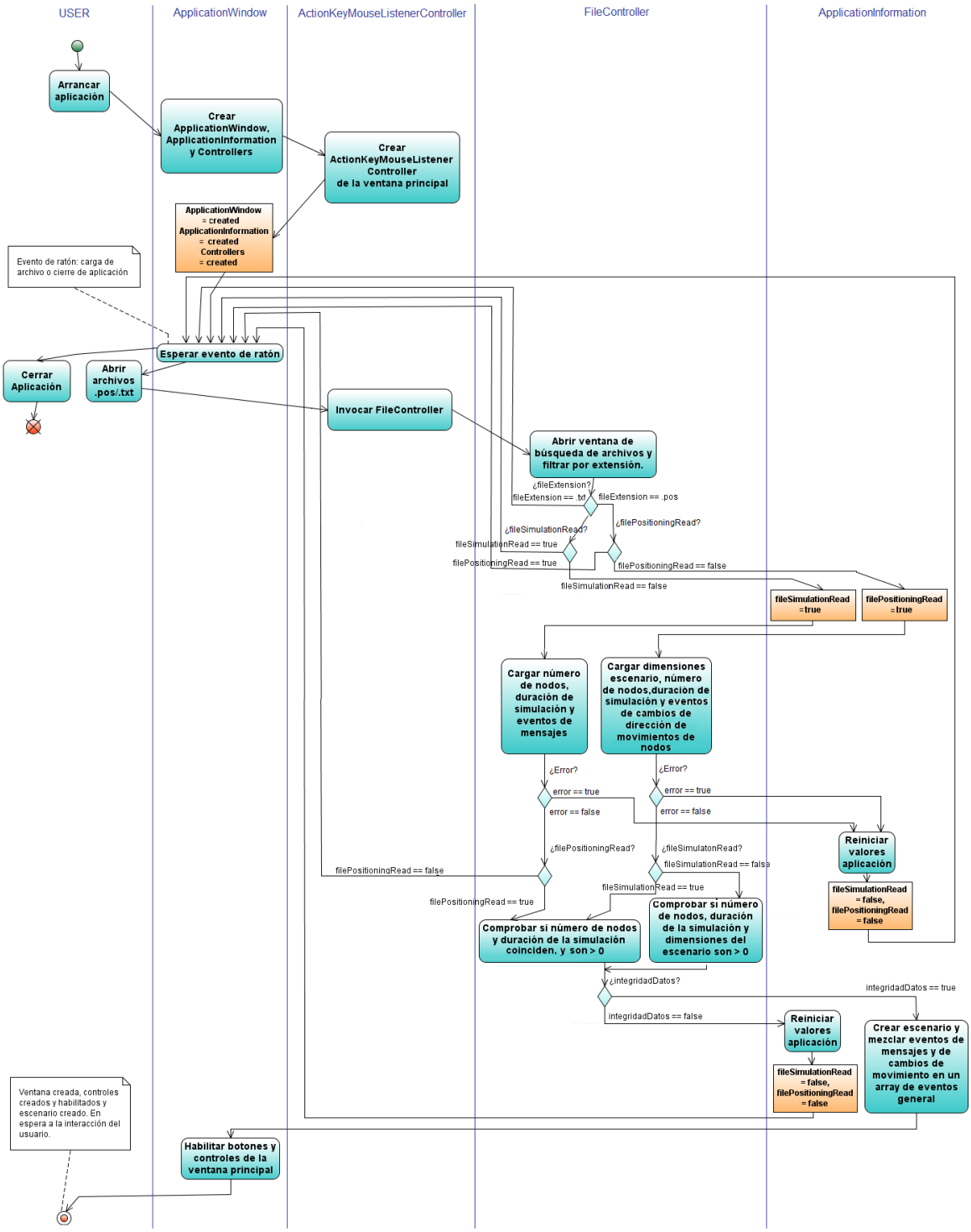


Figura 4.11. Diagrama de actividades - Carga de archivos de entrada y creación del escenario

La instancia de *FileController* es la que tendrá el papel principal en este diagrama, puesto que va a comprobar que se cumplan los requisitos de este grupo. Así, tras abrir la ventana para la búsqueda de archivos y el filtrado de los mismos por el tipo de extensión, procede a la apertura del archivo seleccionado. Se deberá verificar que no se ha cargado anteriormente un archivo con la misma extensión; si es así, se deberá volver al estado de

partida, y en otro caso, se cambiará el valor de la variable de *ApplicationInformation* que controla si se ha cargado anteriormente el archivo en cuestión. Si se ha abierto el archivo de salida de la simulación, no se ha producido ningún error y se ha cargado el archivo de movilidad de los nodos, se pasa a la verificación de la integridad de los datos de ambos archivos, como que coincidan en el número de nodos y en la duración de la simulación y que sean valores mayores a cero; en otro caso, se vuelve al estado de partida, reiniciando los valores de la aplicación en el caso de producirse error. Si se ha abierto el archivo de movilidad de los nodos y no se ha producido ningún error, se haya cargado el archivo de salida de la simulación o no, se pasa a la verificación de la integridad de los datos que contiene o contienen, como que coincidan en el número de nodos y en la duración de la simulación y que sean valores mayores a cero, y que las dimensiones del escenario sean también mayores a cero; en otro caso, se vuelve al estado de partida, reiniciando los valores de la aplicación en el caso de producirse error. Si se verifica esta integridad de los datos, se pasa a la creación del escenario y se habilitan los controles de la ventana principal, mezclando los eventos que provienen de los dos archivos de entrada si se han cargado ya ambos en una estructura de tipo *array* global ordenada temporalmente; si no se verifica, se reinicia la aplicación dejándola como recién abierta, con ninguno de los archivos cargados.

4.5.2.2.2. Avance por saltos temporales: *Step forward*

El avance por saltos temporales o *step forward* implica progresar por la animación en forma de saltos temporales relativos al instante actual, saltos que vienen impuestos por la cantidad (en milisegundos) almacenada en la caja de texto de la ventana principal a tal efecto. Las acciones llevadas a cabo para dejar la animación en el estado marcado por el nuevo instante de tiempo se presentan en el diagrama de actividades de la figura 4.12, donde el usuario procede a activar el control (botón) de avance hacia adelante. Las instancias de clases que realizan estas actividades son el controlador genérico *ActionKeyMouseListenerController*, las clases principales de la aplicación *ApplicationWindow* y *ApplicationInformation*, y los controladores *VisualizationController*, *ScenarioController* y *StatisticsController*.

Cuando el usuario acciona el botón, el evento producido es redirigido desde *ActionKeyMouseListenerController* hacia el controlador *VisualizationController*, que es el encargado de las actividades relacionadas con las opciones de visualización de la animación. Éste comprobará si se ha alcanzado el final de la animación, o en el caso de que no se haya alcanzado, verificará si la caja de texto con la cantidad de tiempo que debe

desplazarse la animación no está vacía; en esos casos se indicaría con una ventana emergente y terminando el flujo de control. Si no se ha alcanzado el final y se ha introducido un valor de tiempo, se pausa la animación si ésta se encontraba activa. A continuación, se debe verificar si el instante actual más el salto temporal que se debe producir genera un nuevo instante de tiempo inferior o igual a la duración total de la animación. Si no es así, se avisaría con una nueva ventana emergente informativa. Si se cumple la condición, el controlador de la visualización va a actualizar el valor de la barra horizontal o *slider* temporal de la ventana principal y el nuevo valor del instante de la animación, el controlador del escenario debe recalcular las nuevas posiciones de los nodos y el próximo evento de los nodos que se debe reproducir y, finalmente, debe ajustar el aspecto de los halos de los nodos, indicando si son nodos que están interviniendo en la petición de un documento por parte de un nodo que se encuentra seleccionado en ese nuevo instante de tiempo. Por último, el controlador de estadísticos deberá calcular los estadísticos válidos en el nuevo instante de tiempo para aquellos nodos que se encuentren seleccionados y mostrarlos en la ventana principal.

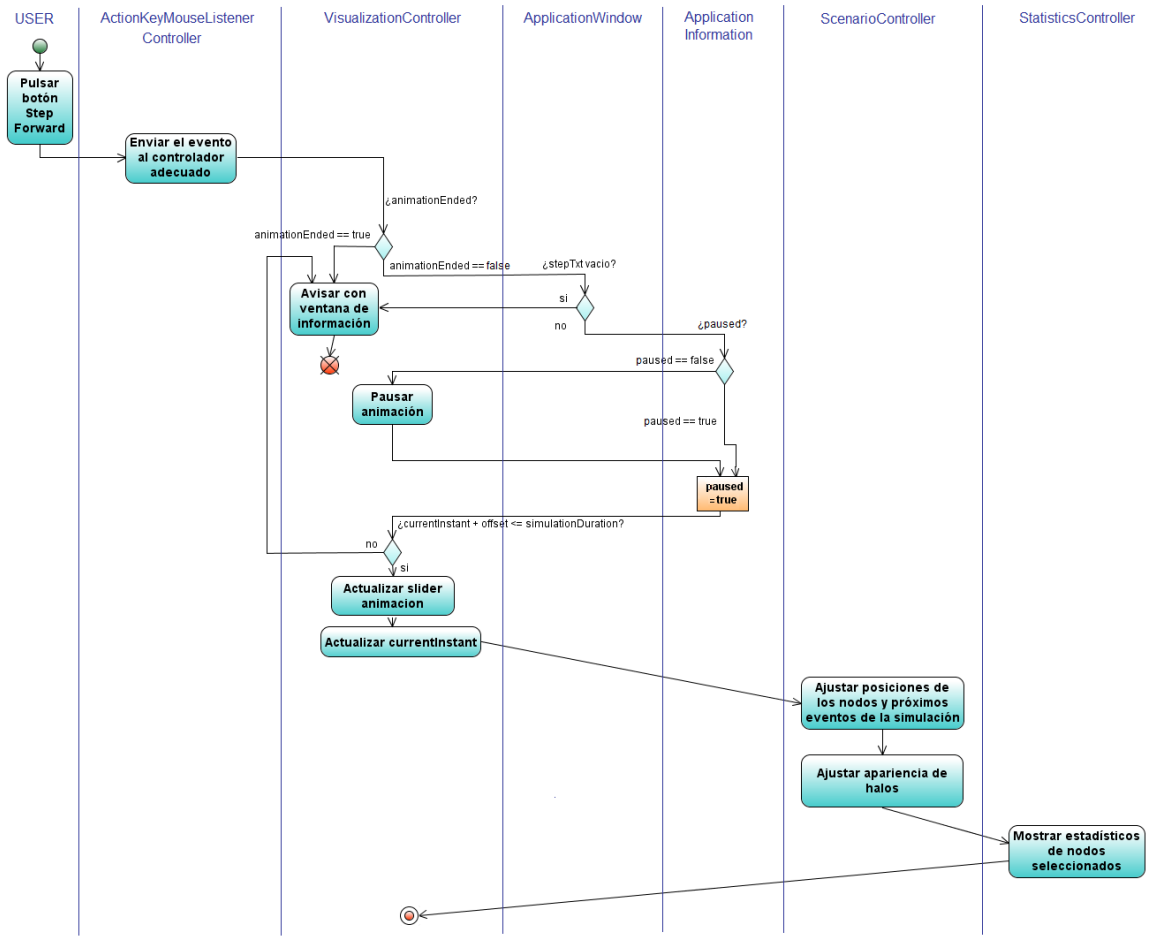


Figura 4.12. Diagrama de actividades - Avance por saltos temporales: *Step forward*

4.5.2.2.3. Retroceso por saltos temporales: *Step backward*

Las actividades a realizar para proceder a un salto temporal hacia un instante anterior del actual se presentan en el diagrama de actividades de la figura 4.13. En dicho diagrama se puede observar que las instancias de las clases involucradas son las mismas que para el caso del salto temporal hacia adelante expuesto en el apartado anterior. De hecho, las acciones principales que se llevan a cabo son idénticas, difiriendo en las condiciones a cumplir para la realización correcta de la actualización del instante temporal.

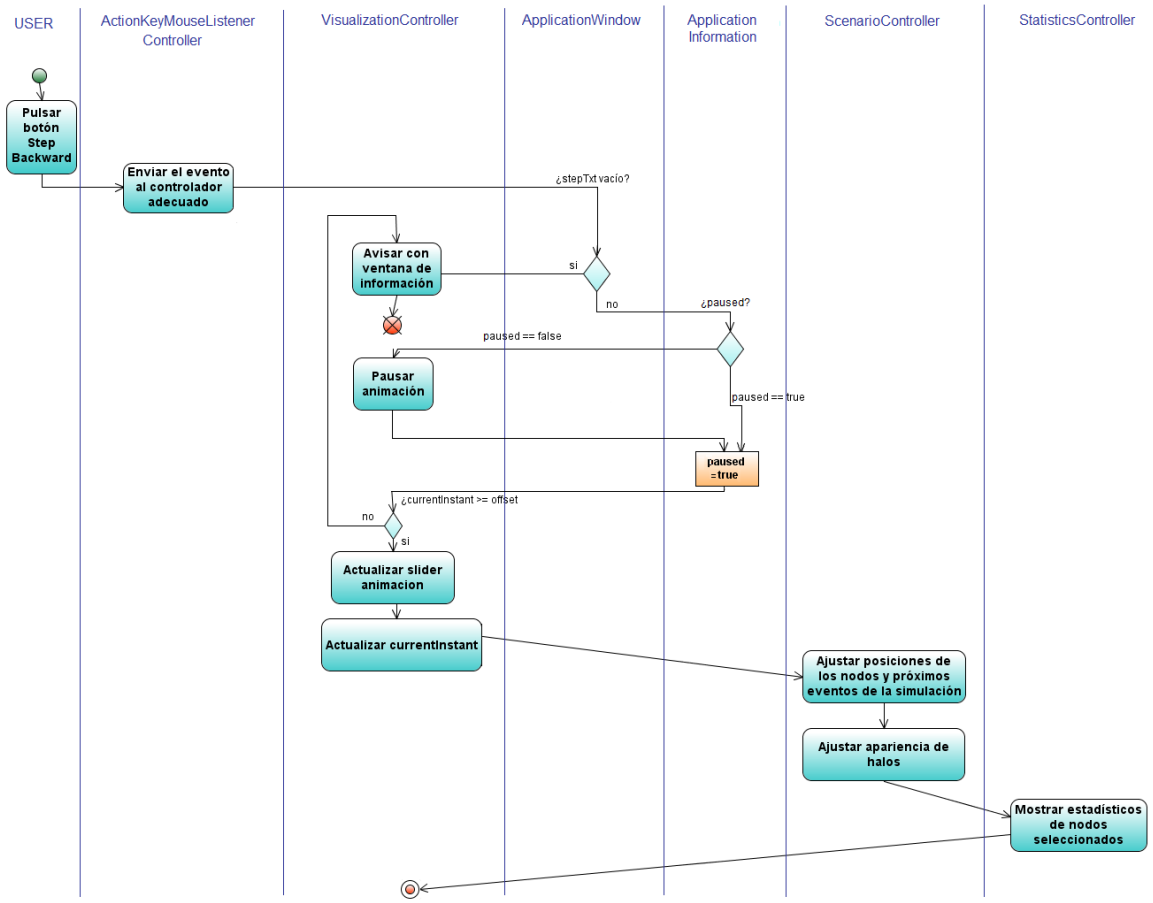


Figura 4.13. Diagrama de actividades - Retroceso por saltos temporales: *Step backward*

En este caso no se produce la primera comprobación acerca de si la animación ha alcanza el final o no, puesto que se podría retroceder en el tiempo. La segunda condición que se debe verificar es que el instante actual sea superior o igual que el salto que se debe producir, porque no se podrían tener instantes anteriores al inicio de la simulación, establecido en cero. Si ésta no se verifica, se indicará a través de una ventana emergente que no se puede retroceder esa cantidad en el tiempo, finalizando el flujo de actividades en ese instante. Como en el caso de avance temporal, la animación quedará pausada si no lo estaba tras activar el control de retroceso temporal (o botón *step backward*), y actualizará

los halos para aquellos nodos que en el nuevo instante de tiempo formen parte del camino que está siguiendo una determinada petición por la red realizada por un nodo seleccionado, mostrando además un resumen de los estadísticos de dichos nodos seleccionados en la zona de la ventana principal habilitada para ello.

Tras la explicación de los diagramas de actividades para los casos de saltos temporales hacia adelante y hacia atrás, se ha decidido no mostrar el diagrama para la opción de avance y/o retroceso por instantes absolutos puesto que las actividades a realizar serían las mismas que en los casos anteriores, uniendo las condiciones mostradas en ambos diagramas para no ir más allá de final de la simulación ni hacia instantes negativos inexistentes.

4.5.2.2.4. Reanudación de la animación: *Play*

A continuación se van a explicar a grandes rasgos las actividades llevadas a cabo para realizar la reanudación de la animación cuando se encuentra pausada, mostradas en el diagrama de actividades de la figura 4.14.

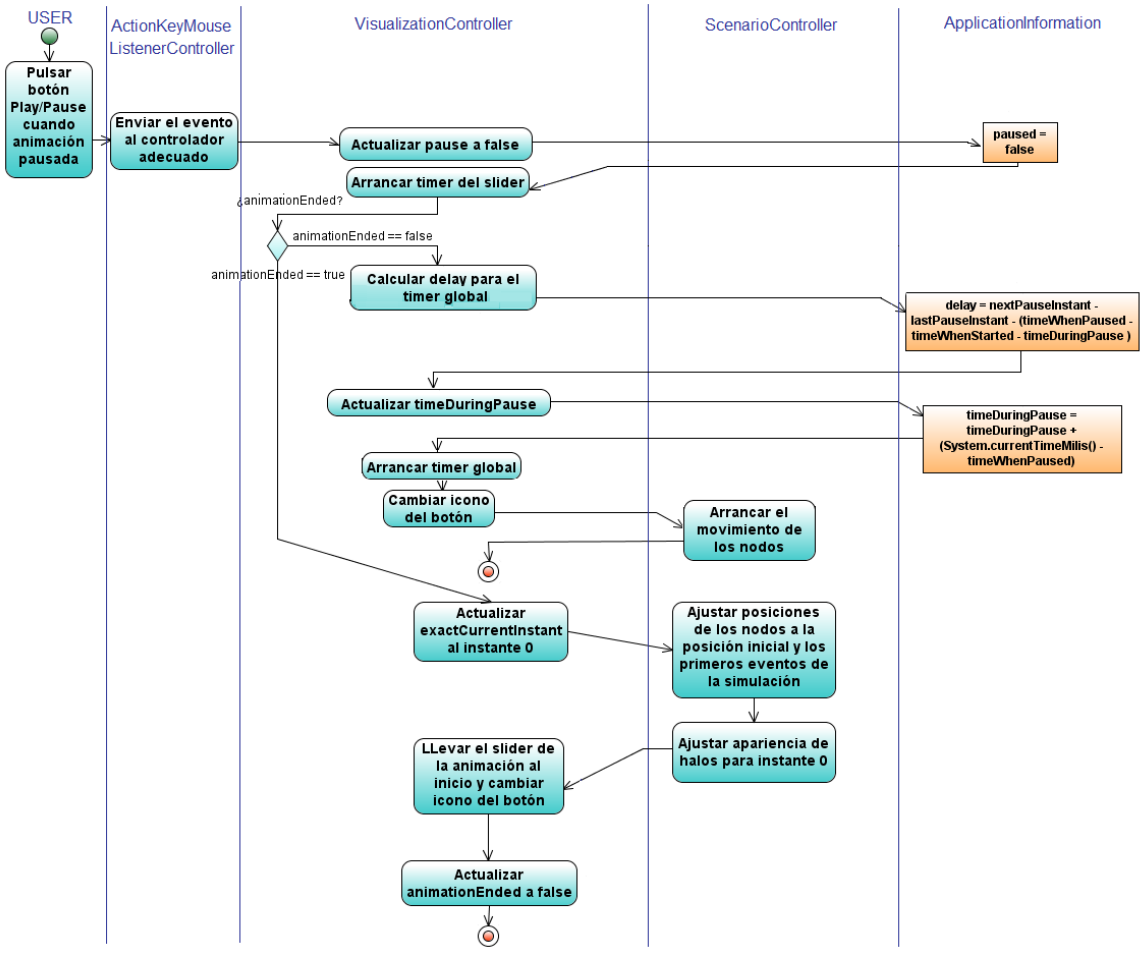


Figura 4.14. Diagrama de actividades - Reanudación de la animación: *Play*

En el diagrama se muestran las instancias de las clases que intervienen, que son el controlador genérico *ActionKeyMouseListener Controller* que, como ya se ha indicado, dirige el flujo de control hacia el controlador adecuado, el objeto principal de la aplicación *ApplicationInformation* indicando el valor que tienen algunos de sus atributos, y los controladores *VisualizationController* (que gestiona los parámetros de la visualización de la animación), y *ScenarioController* (que ajusta las posiciones de los nodos y cambia el aspecto visual de los mismos).

El controlador *VisualizationController* fija el valor del atributo *pause* de *ApplicationInformation* a falso, arranca el temporizador que actualiza el valor del *slider* y pasa a seguir un flujo de control con actividades diversas en función de su atributo *animationEnded*. Si se ha alcanzado el final de la animación, al reanudar la misma hay que comenzarla desde el principio actualizando el instante actual a cero, el controlador gráfico del escenario recalculará las posiciones de los nodos, activando sus movimientos y primeros eventos de la simulación y ajustará la apariencia de los halos, finalizando con la modificación de la posición del *slider* de la ventana principal de la aplicación. En otro caso, cuando no se ha alcanzado el final de la animación, la reanudación se debe producir desde el instante actual en que se encuentre la animación, calculando el retardo inicial y activando el temporizador global que gestiona los eventos en los nodos durante la animación, actualizando además el valor del tiempo que ha permanecido la animación pausada. Finalmente, *ScenarioController* activará el movimiento de los nodos.

4.5.2.2.5. Pausado de la animación: *Pause*

Cuando el usuario interacciona con la aplicación para pausar la animación, se suceden una serie de actividades que, a grandes rasgos, se presentan en el diagrama de actividades de la figura 4.15.

De la misma forma que ocurre al reanudar la animación, las instancias de clases que intervienen en el pausado de la misma son *ActionKeyMouseListenerController*, *ApplicationInformation*, *VisualizationController* y *ScenarioController*. Una vez que se dirige el flujo de control hacia el controlador de la visualización, se actualiza el instante actual en el que se ha producido la pausa, se paran los dos temporizadores utilizados y se actualiza la imagen o icono del botón de la ventana principal de la aplicación. El controlador del aspecto gráfico del escenario pausará los nodos y mostrará sus halos en el

caso en el que corresponda. Finalmente, se actualiza la variable *timeWhenPaused* al instante actual para que se pueda obtener el tiempo que la animación permanece pausada.

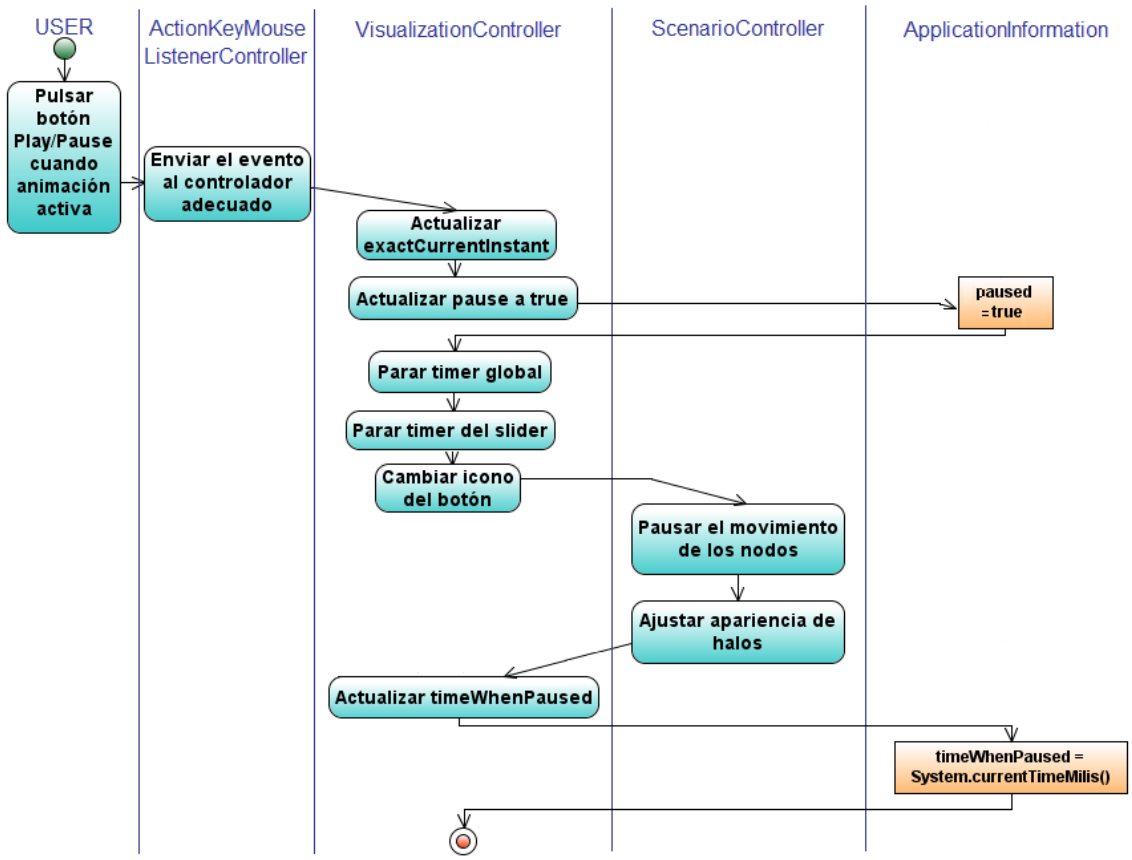


Figura 4.15. Diagrama de actividades - Pausado de la animación: *Pause*

4.5.2.2.6. Parada de la animación: *Stop*

La parada de la animación se produce activando el control *stop*. Las acciones que se deben llevar a cabo tras este evento se muestran en el diagrama de actividades de la figura 4.16, donde vuelven a aparecer involucradas las instancias de las mismas clases que para la reanudación y pausa de la animación, es decir, *ActionKeyMouseListenerController*, *VisualizationController*, *ScenarioController* y *ApplicationInformation*. Con el fin de parar la animación, ya sea estando pausada o activa, se deben situar los nodos en su posición inicial y reiniciar las variables que intervienen en la representación temporal de la animación para que se empiecen a dar los primeros eventos de los nodos, actualizando el aspecto gráfico de los halos de los nodos. Además, se deberá dejar la animación en estado pausado, parando por tanto el movimiento de los nodos, cambiar la imagen asociada al botón pulsado, parar los dos temporizadores utilizados en la aplicación (es decir, el temporizador global para la secuenciación de los eventos de los nodos y el temporizador para el avance del *slider*) y llevar el *slider* al extremo inicial.

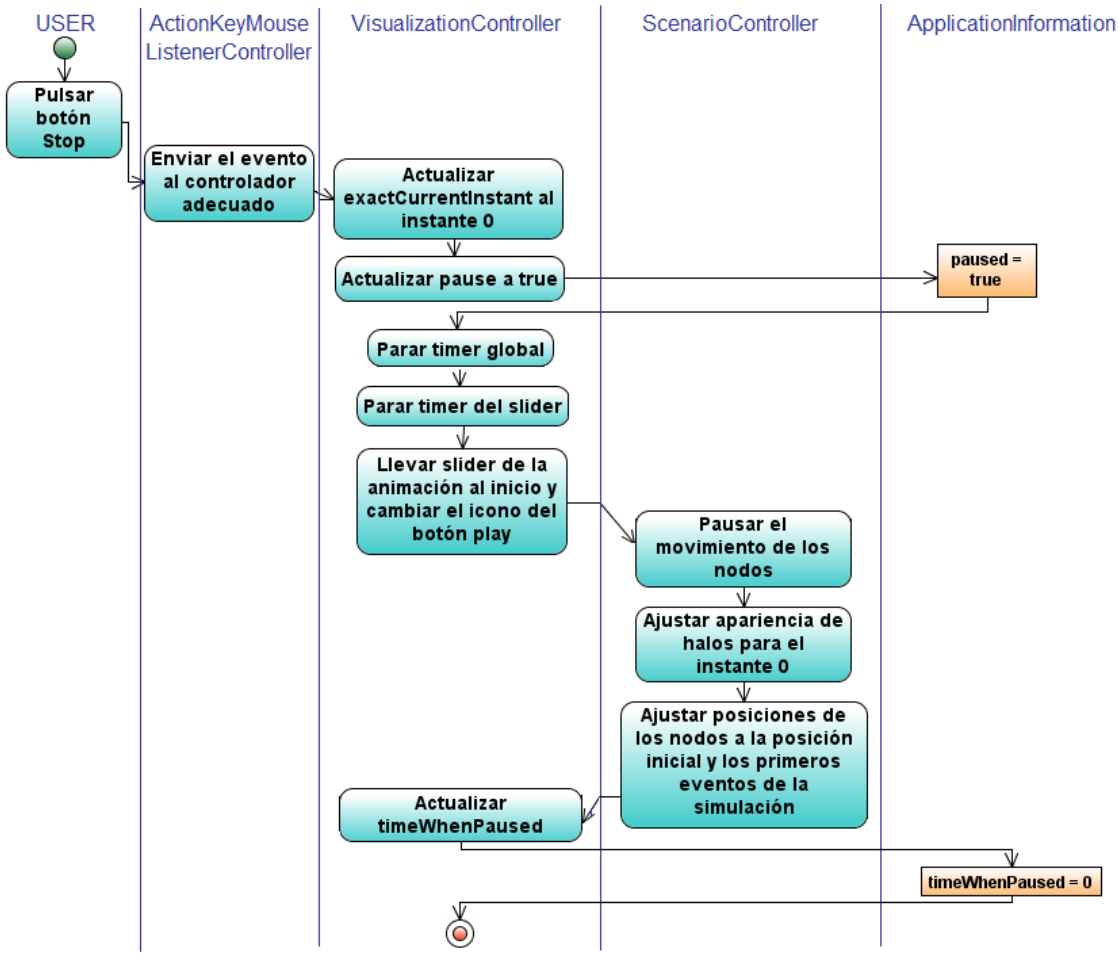


Figura 4.16. Diagrama de actividades - Parada de la animación: *Stop*

4.5.2.2.7. Evento del temporizador global: *Timer global*

El temporizador o *timer* global es el encargado de secuenciar los diferentes eventos que deben ir generándose durante la animación. Así, el retardo que se irá fijando para el temporizador tendrá un valor igual a la diferencia de tiempos entre dos eventos de nodos (ya sean eventos provenientes del archivo de movilidad de los nodos o bien eventos del archivo de la salida de la simulación con el intercambio de mensajes por la red) sucesivos, aplicando un factor dependiente del ratio entre las velocidades de la animación y de la simulación de base. Cada vez que los temporizadores expiren (agoten el retardo introducido) generarán un evento que deberá ser atendido. Las actividades que se suceden al producirse este evento se pueden resumir en el diagrama de actividades de la figura 4.17. En ella se pueden observar las instancias de las clases involucradas, además del controlador de la visualización *VisualizationController*, que es el controlador que contiene el código que se debe ejecutar al producirse este evento, al igual que todos los eventos de la visualización explicados con los diagramas de actividades utilizados. Estas otras clases involucradas son *ActionKeyMouseListenerController*, *ScenarioController*,

StatisticsController, *ApplicationInformation*, y *NodeInformation*, que contiene los parámetros de los movimientos actuales que siguen los nodos.

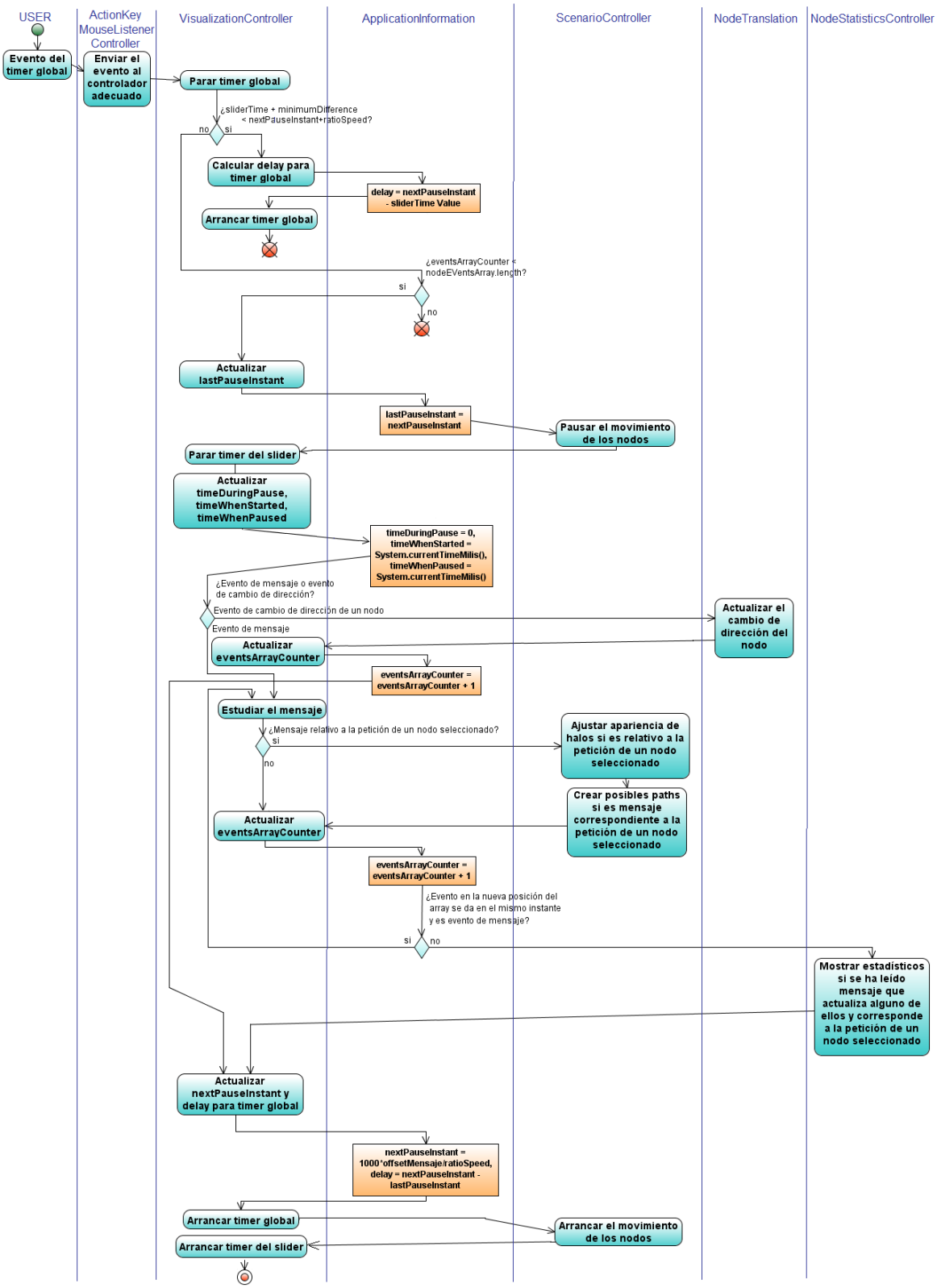


Figura 4.17. Diagrama de actividades - Evento del temporizador global: *Timer global*

La primera condición que se verifica es si ambos temporizadores están avanzando con una diferencia mínima (fijada) entre ellos, comprobando que ninguno esté avanzado respecto al otro. Para ello se toma el valor del instante actual que marca el temporizador global, que se obtiene a partir de la variable *nextPauseInstant*, y el valor que marca el *slider*. Si el temporizador que avanza el *slider* ha provocado que el propio *slider* contenga un valor inferior al valor marcado por el temporizador global, se deberá retardar el siguiente evento relativo a un nodo a reproducir en la animación, tomando la diferencia entre ambos temporizadores y finalizando el flujo de control. Si por el contrario no existe una diferencia mínima entre ambos valores se procederá a tomar el nuevo valor del retardo a fijar en el temporizador global de eventos de nodos, no sin antes verificar que no se ha alcanzado ya el último evento almacenado en el *array* global de eventos de nodos. Se parará el temporizador del *slider* para evitar que avance mientras el procesador está ejecutando este código asociado, y se pausarán los nodos. Se actualizarán los valores de *lastPauseInstant*, *timeDuringPause*, t*imeWhenPaused* y *timeWhenStarted*, que se explicarán con más detalle cuando se presenten los diagramas de clases.

Si el evento relacionado con un nodo del escenario proviene del archivo de movilidad de los nodos, entonces únicamente se actualiza el cambio de dirección del nodo y el valor de *eventsArrayCounter*, que indica la posición en el *array* global de eventos del siguiente evento que debe reproducirse. Si por el contrario se trata de un evento que proviene del archivo de salida de la simulación, se debe estudiar dicho mensaje para deducir si está directamente relacionado con un nodo que se encuentre seleccionado, en cuyo caso *ScenarioController* ajustará la apariencia de los halos de los nodos y dibujará a través de pequeñas animaciones el envío de mensajes por la red, finalizando con la actualización igualmente de *eventsArrayCounter*. Si el evento siguiente que se debe producir es de nuevo un mensaje y se genera en el mismo instante de tiempo, se repetirán estas últimas acciones, mostrando los estadísticos que se hayan modificado debido a esos mensajes para los nodos que estén seleccionados. La última parte, común a cualquier tipo de evento de nodo, actualiza el valor de *nextPauseInstant* con el ratio de velocidades oportuno, calcula el nuevo retardo del temporizador global y arranca ambos temporizadores, iniciando además los movimientos de los nodos.

4.5.2.2.8. Evento del temporizador de la barra horizontal del avance temporal: *Timer del slider*

El temporizador cuyo evento es presentado a continuación se encarga de hacer avanzar el ítem que se desplaza por la barra horizontal o *slider*. Las actividades que se llevan a

cabo para gestionar el evento producido por este temporizador al agotarse el retardo introducido se muestran en el diagrama de actividades de la figura 4.18.

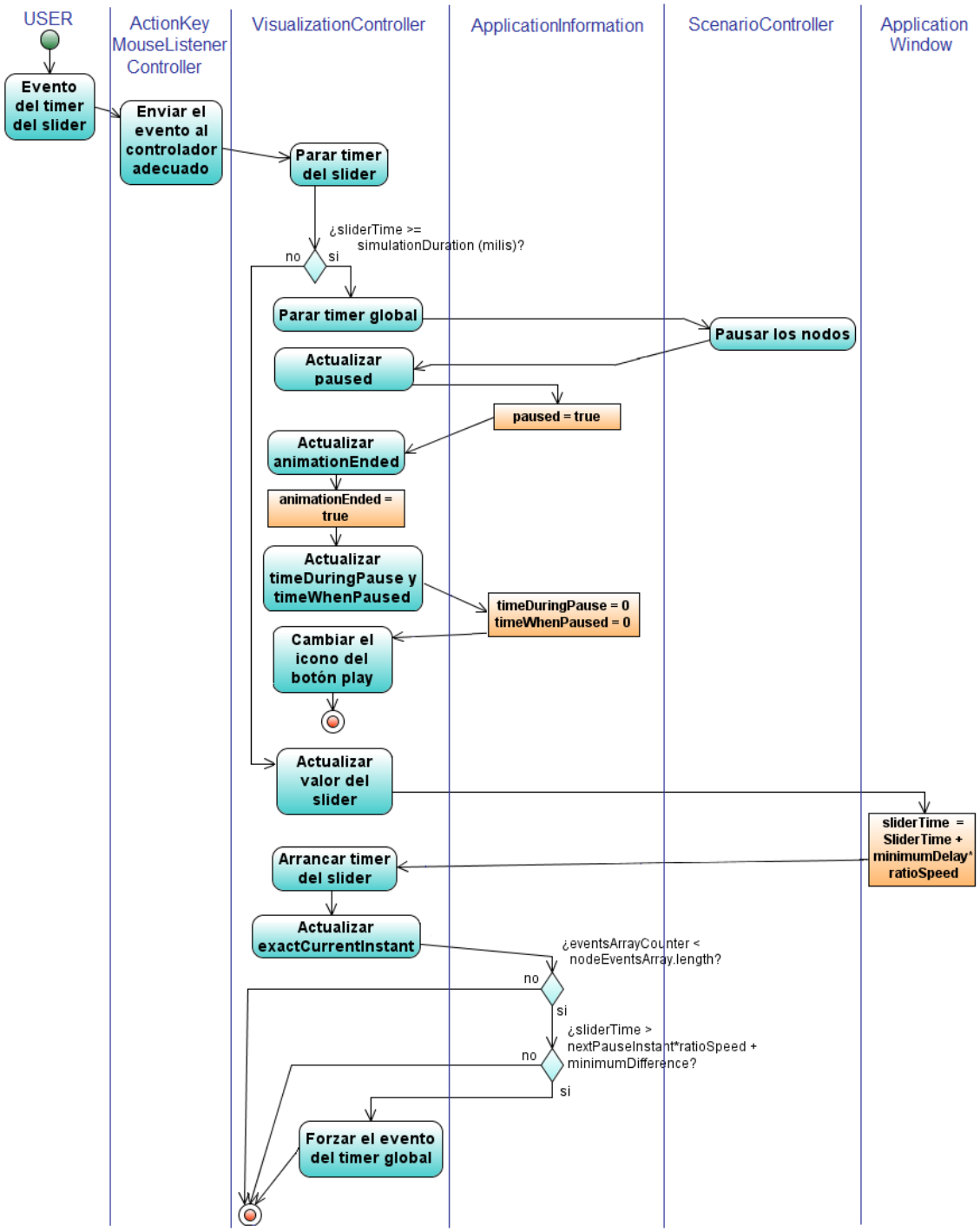


Figura 4.18. Diagrama de actividades - Evento del temporizador de la barra horizontal del avance temporal: *Timer slider*

Este temporizador tendrá un retardo fijo que dependerá exclusivamente del ratio entre las velocidades de visualización de la animación y la simulación de base, de forma que cuanto mayor sea este ratio, la animación avanzará más deprisa, aumentando el valor del

ítem sobre el *slider* cada menos tiempo, o lo que es lo mismo, fijando un retardo menor. La lógica asociada al evento producido tras agotar el retardo de este temporizador consume menos tiempo respecto de la lógica asociada al evento del temporizador global presentado en el apartado anterior, de forma que los ajustes entre ambos temporizadores se irán realizando a medida que presenten una diferencia de valor mínima.

Las instancias de clases que aparecen son, además de *VisualizationController*, que contiene el código a ejecutar para gestionar el evento, *ActionKeyMouseListenerController*, *ScenarioController*, *ApplicationInformation* y *ApplicationWindow*.

Existen dos flujos de control que se pueden seguir en función del valor del *slider*. Así, si se ha alcanzado el final de la simulación, se debe finalizar el temporizador global, pausar la animación y fijar los valores utilizados por la animación para indicar que no se puede avanzar más, de forma que la próxima vez que se active el control de reanudación de la animación, el software reconozca que debe comenzar la animación desde el principio. Si aún no se ha alcanzado el final, se actualiza el valor del *slider*, simulando el avance temporal, se arranca de nuevo el temporizador con el retardo que venga fijado según el ratio de velocidades ya nombrado en anteriores ocasiones y se verifica que ambos temporizadores estén avanzando con una diferencia de tiempos mínima, porque si no es así y el temporizador global se encuentra retrasado, se forzará a que ocurra el siguiente evento de nodo que se deba reproducir.

4.5.2.2.9. Cambio de velocidad

Como requisito de la aplicación, el software debe ofrecer la posibilidad de cambiar la velocidad de representación de la animación en cualquier momento, ya sea estando pausada como activa, dejándola en el mismo estado. Las actividades que debe llevar a cabo *VisualizationController*, con la ayuda de otros controladores, se exponen en el diagrama de actividades de la figura 4.19. En ella se observan las otras instancias de clases involucradas por realizar una pequeña parte de las actividades, como *ScenarioController*, o bien instancias de clases que forman parte del modelo y experimentan cambios en algunos de sus atributos, como *ApplicationInformation* y *NodeTranslation*. Como en los casos anteriores, la instancia del controlador genérico *ActionKeyMouseListenerController* es quien detecta el evento y lo redirige hacia el controlador de la visualización de la animación para que lo gestione.

La primera actividad que debe realizar *VisualizationController* es detener los dos temporizadores utilizados en la aplicación, obteniendo el nuevo valor del retardo que

tendrán debido al cambio de velocidad de la animación. Así, a mayor velocidad de representación de la animación, menores serán dichos retardos, además de las diferencias mínimas que deberán tener ambos temporizadores para ajustar sus posibles diferencias. Además, se actualizarán los valores de las variables *lastPauseInstant* y *nextPauseInstant* que corresponden al instante en que se produjo el último evento de nodo y el instante en que se producirá el siguiente, respectivamente. Estas variables dependen directamente de la velocidad de la animación elegida, como se verá con más detalle en el diagrama de clases de *ApplicationInformation*, donde se explicará el significado de cada uno de sus atributos y miembros de clase. La clase *NodeTranslation*, que contiene los datos relativos al movimiento actual que sigue cada nodo, generará los parámetros para los movimientos modificados con la nueva velocidad. Finalmente, si la animación no se encontraba pausada, se arrancarán los temporizadores de la aplicación y el controlador del escenario gráfico iniciará la movilidad de los nodos.

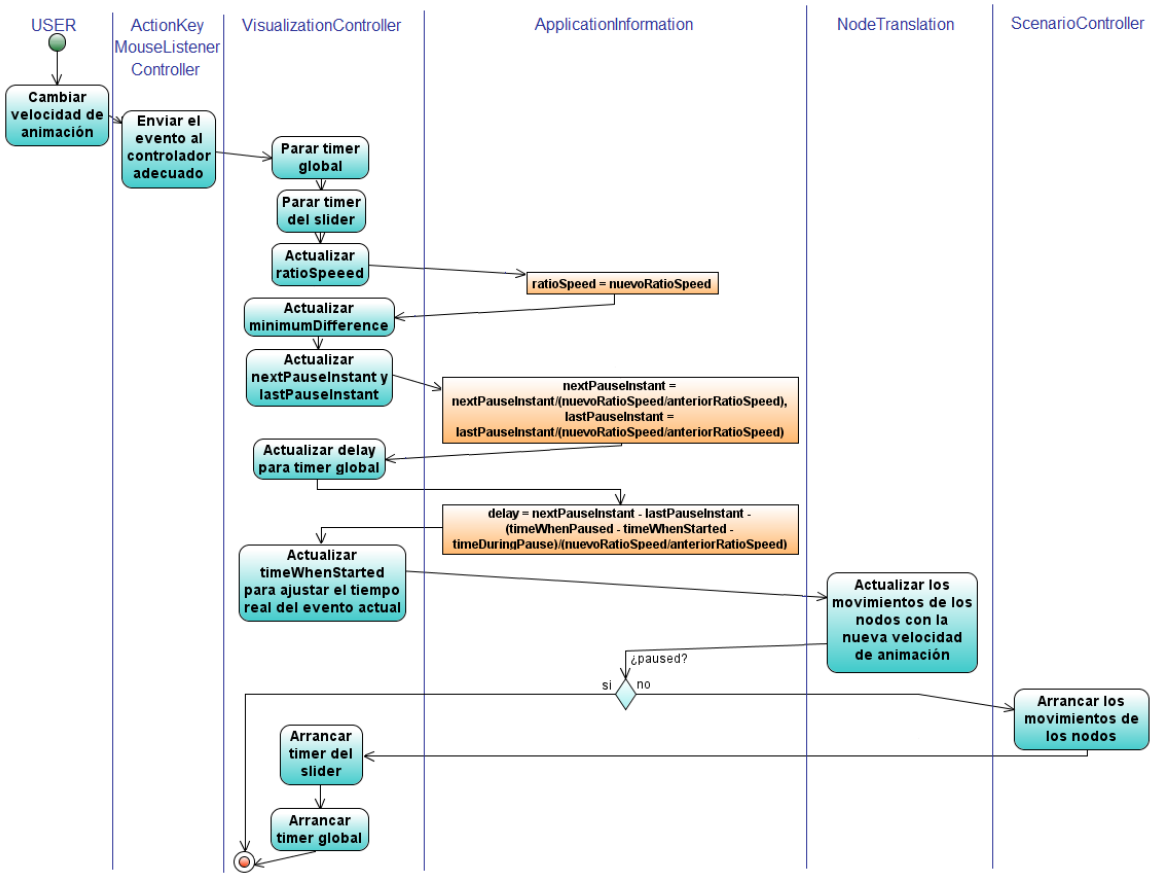


Figura 4.19. Diagrama de actividades - Cambio de velocidad

4.5.2.2.10. Avance o retroceso evento a evento

El último diagrama de actividades realizado, presentado en la figura 4.20, contiene las actividades que se deben llevar a cabo para realizar el avance y/o retroceso por los eventos

que estén directamente relacionados con nodos que se encuentren seleccionados en un instante concreto.

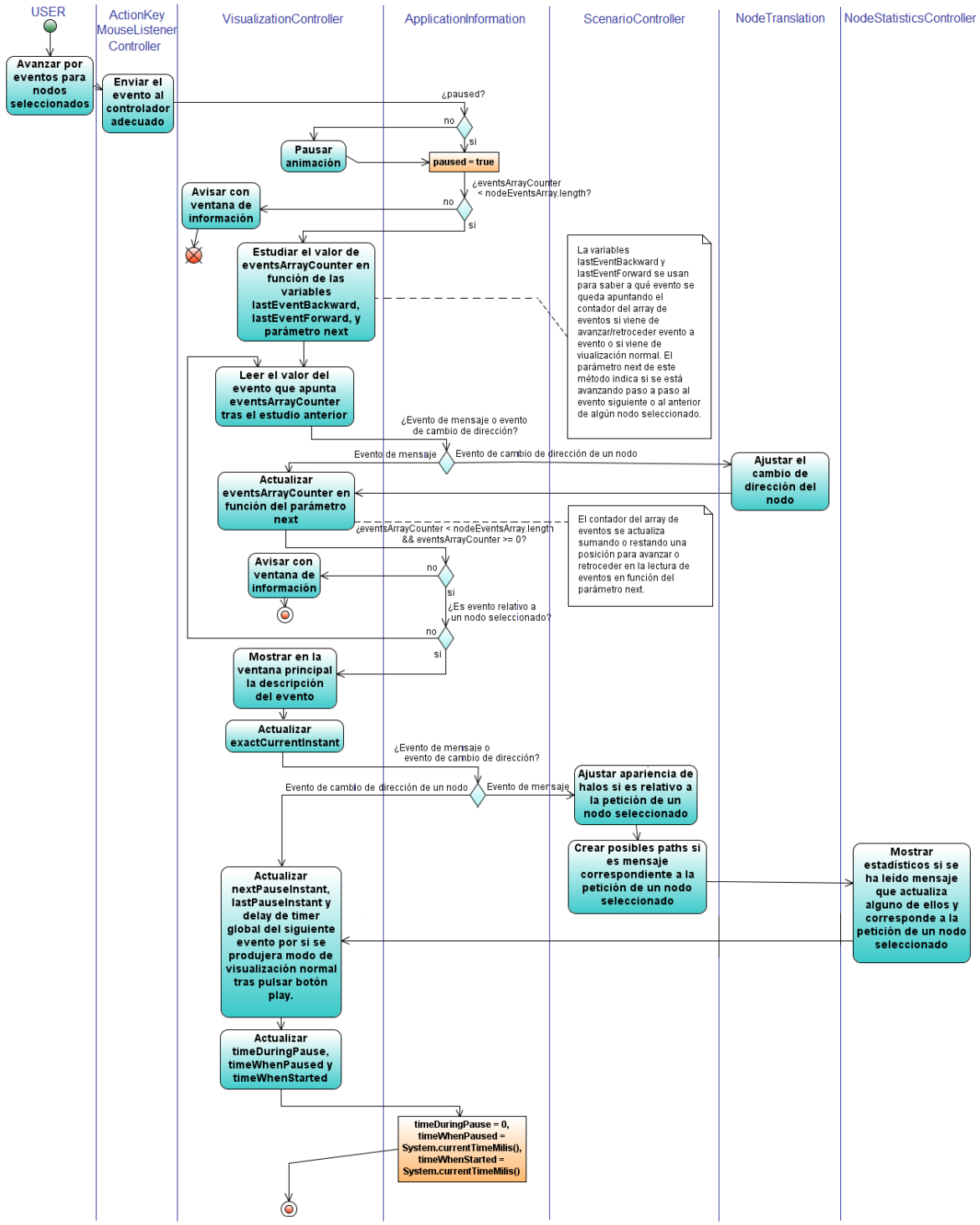


Figura 4.20. Diagrama de actividades - Avance o retroceso evento a evento

La forma de definir cómo se considera que un evento está directamente relacionado con un nodo pertenece al diagrama de clases de *VisualizationController*, en el cual aparece el método que se encarga de ello. Para el caso del avance y/o retroceso para los eventos de

cualquier nodo de la red se tendrían las mismas actividades que las aquí presentadas pero descartando la búsqueda en bucle de un evento que case con un nodo seleccionado.

Las actividades que se presentan en el diagrama son realizadas, además de por *VisualizationController*, por las instancias de las clases controladoras *ActionKeyMouseListenerController*, *ScenarioController*, *NodeStatisticsController* y por las clases pertenecientes al modelo de datos *ApplicationInformation* y *NodeTranslation*. Cuando el usuario cambia la velocidad en la representación de la animación, el controlador *ActionKeyMouseListenerController* redirige el evento hacia el controlador de la visualización *VisualizationController*, que a su vez pausa la animación si no se encontraba pausada y comprueba que no se ha alcanzado el final de la animación, porque si se ha producido se aborta el flujo de actividades previo aviso con una ventana de información. En el caso de que no se haya alcanzado el final de la simulación se deberá estudiar la posición en el *array* global de eventos de nodos del siguiente evento de la animación que se debe reproducir dependiendo de si el último evento que se dio fue hacia adelante o hacia atrás en la animación, y de si ahora se debe avanzar o retroceder.

Las siguientes actividades se enmarcan dentro de un bucle temporal, de forma que se irá avanzando o retrocediendo (según sea el caso) por los diferentes eventos de nodos hasta que se encuentre un evento relacionado directamente con un nodo, actualizando en cada iteración el valor del atributo *eventsArrayCounter* y el movimiento de los nodos si se encuentra un evento de movilidad de nodos aunque no sea de un nodo relacionado, ya que los cambios de dirección se deben insertar en el escenario. Se saldrá del bucle cuando se haya detectado un evento relacionado con un nodo seleccionado o bien cuando se haya alcanzado el primer evento o el último de la animación, en cuyo caso finalizará el flujo de control previo aviso de una ventana de información. Una vez que se obtiene el evento de nodo buscado, se muestra su descripción en la ventana principal de la aplicación y se modifica la variable que controla el instante actual de la animación en la que se ha posicionado. Además, se deben actualizar todas las variables que se utilizan para la reproducción de la animación, como *timeWhenPaused*, *timeWhenStarted*, *timeDuringPause* y el retardo del temporizador global de la animación, definido por la diferencia de tiempos entre el instante del evento actual de nodo y el evento siguiente a reproducir, con el factor del ratio de velocidad de animación utilizado. Como sucede en otros diagramas de actividades realizados, el controlador del escenario gráfico se encargará de mostrar los halos de los nodos que se encuentren afectados por el evento actual, si los hubiera, además de generar una animación de envío de mensaje por la red si se diera el

caso. Finalmente, el controlador de los estadísticos mostrará un resumen de estadísticos si el evento del nodo encontrado actualiza alguno de ellos.

Aunque no venga especificado explícitamente en el diagrama, cuando no se encuentra ningún evento relacionado con un nodo seleccionado, se recupera el instante en el que se encontraba la animación y el siguiente evento a reproducir.

4.5.2.3. Diagramas de Clases

Un diagrama de clases muestra un conjunto de clases, interfaces y colaboraciones, así como sus relaciones. Estos diagramas son los más comunes en el modelado de sistemas orientados a objetos. Los diagramas de clases abarcan la vista de diseño estática de un sistema. Esto incluye, principalmente, modelar el vocabulario del sistema, modelar las colaboraciones o modelar esquemas.

Los diagramas de clases contienen normalmente los siguientes elementos: clases, interfaces y relaciones de dependencia, generalización y asociación:

- Una dependencia es una relación de uso que declara que un elemento utiliza la información y los servicios de otro elemento, pero no necesariamente a la inversa. La mayoría de las veces, las dependencias se utilizarán en el contexto de las clases, para indicar que una clase utiliza las operaciones de otra o que utiliza variables o parámetros cuyo tipo viene dado por otra clase. Esto es claramente una relación de uso: si la clase utilizada cambia, la operación de la otra clase puede verse también afectada, porque la clase utilizada puede presentar ahora una interfaz o un comportamiento diferentes. Una dependencia puede contener un nombre, aunque es raro que se necesiten los nombres, a menos que se tenga un modelo con muchas dependencias y haya que referirse a ellas o distinguirlas. En el caso de nuestro diseño y como norma general, las relaciones de uso utilizadas en los diagramas de clases quedan representadas por el estereotipo <<*usage*>>, tal y como se presenta en la figura 4.21, en la cual se muestra cómo la Clase1 está utilizando a la Clase2.

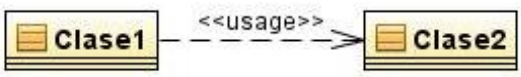


Figura 4.21. Relación de Dependencia

- Una generalización es una relación entre un elemento general (llamado superclase o padre) y un caso más específico de ese elemento (llamado subclase o hijo). La

generalización se llama a veces relación de "es-un-tipo-de": un elemento es-un-tipo-de otro elemento más general. Un objeto de la clase hija se puede asociar a una variable o un parámetro cuyo tipo venga dado por el padre, pero no a la inversa. En otras palabras, la generalización significa que el hijo puede sustituir a una declaración del padre. Un hijo hereda las propiedades de sus padres, especialmente sus atributos y operaciones; además, éste puede añadir atributos y operaciones propias de él. Este comportamiento corresponde a la **herencia** en programación orientada a objetos. Gracias a la herencia se establece el mecanismo del **polimorfismo**, que implica la posibilidad de acceder a un rango variado de funciones distintas a través del mismo interfaz, lo cual representa una característica fundamental de los lenguajes orientados a objetos. Una implementación de una operación en un hijo que redefine la implementación de la misma operación en el padre se conoce en programación orientada a objetos como **sobreescritura** de métodos, donde además se debe cumplir que ambas operaciones deben tener la misma signatura, es decir, mismo nombre y parámetros. Si por el contrario tienen únicamente igual nombre pero distinta funcionalidad representada con diferentes parámetros en número y/o tipo, define una segunda forma de polimorfismo denominada **sobrecarga** de métodos. En UML, una clase puede tener ninguno, uno o más padres, pero en el lenguaje de programación escogido (Java) la herencia múltiple no está permitida como norma general (realmente existe algo similar a tener herencia múltiple en Java que es a través de la implementación de interfaces, pero esto queda fuera del interés de este epígrafe). Gráficamente, la generalización se representa como una línea dirigida continua, con una gran punta de flecha vacía, apuntando al padre, como se muestra en la figura 4.22.

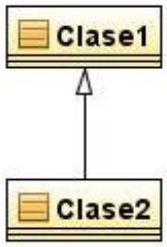


Figura 4.22. Relación de Generalización

- Una asociación es una relación estructural que especifica que los objetos de un elemento están conectados con los objetos de otro. Dada una asociación entre dos clases, se puede establecer una relación desde un objeto de una clase (Clase1) hasta

ninguno o más objetos de otra clase (Clase2), lo que se denomina **multiplicidad** en los extremos de la asociación, como se muestra en la figura 4.23.

Figura 4.23. Relación de Asociación con multiplicidad

A la anterior asociación se le denomina binaria, porque conecta exactamente dos clases, pero podría darse el caso de conectar *n* clases, dando lugar a las asociaciones *n-arias*. Las asociaciones pueden contener, además de la multiplicidad, un nombre, que se utiliza para describir la naturaleza de la misma. En los diagramas de clases creados, este nombre indica el atributo que realmente está conectando ambas clases.

Una asociación normal entre dos clases representa una relación estructural entre iguales, es decir, ambas clases están conceptualmente en el mismo nivel, sin ser ninguna más importante que la otra. A veces, se desea modelar una relación todo/parte en la cual una clase representa "el todo", que a su vez consta de elementos más pequeños, que son "las partes". Este tipo de relación se denomina agregación, la cual representa una relación del tipo "tiene un", es decir, un objeto del todo tiene objetos de la parte. Gráficamente, se expresa añadiendo a la asociación un rombo vacío en la parte del todo, como se observa en la figura 4.24.

Figura 4.24. Relaciones de Agregación

En la representación gráfica de las asociaciones (y agregaciones) se puede incorporar una flecha para indicar el flujo de la relación, denominado navegación de la asociación. Si no incorpora adorno en ninguno de los extremos de la asociación indica que el flujo de la asociación es bidireccional, que es el caso general. Pero si se quiere dejar claro la dirección de la navegación, entonces se añade una flecha apuntando a la dirección del recorrido.

A continuación se van a ir explicando los diferentes diagramas de clases que se han obtenido tras la finalización del desarrollo de la aplicación donde se muestra el contenido de cada una de las clases. Estos diagramas se van a presentar bajo diferentes epígrafes,

agrupados por el paquete donde se encuentran atendiendo al MVC anteriormente introducido, incluyendo como primer apartado el diagrama de clases de la clase principal, además de un último paquete donde se van a incluir las clases adicionales necesarias para el correcto funcionamiento de la aplicación bajo el nombre de utilidades. En los diagramas de clases van a aparecer una serie de estereotipos, como <<*boundary*>> que indica que se trata de una interfaz de comunicación con el usuario, <<*control*>> que representa las clases que se han definido como controladoras según MVC, <<*entity*>> que expresa una clase utilizada para modelar los datos que maneja la lógica, <<*interface*>> que indica cuándo se está aludiendo a una interfaz de Java en lugar de a una clase como tal, y <<*datatype*>> cuando se trata de una clase ya definida por Java. Las clases que aparecen en los diagramas se han organizado por grupos de clases de forma que se ha intentado que aparezcan juntas aquellas que tengan algún tipo de relación. Además, las relaciones que existen entre clases van a quedar bien explicitadas, de forma que si una clase tiene algún tipo de relación con otra clase que se presente gráficamente en otro diagrama, se va a introducir al menos el nombre de la clase con su paquete o carpeta correspondiente, aunque esta clase se encuentre bien detallada en el otro diagrama.

Es de gran importancia destacar que, a través de los diagramas de clases que se van a presentar, se puede observar cómo se ha intentado reducir el acoplamiento entre las clases, de forma que las relaciones entre clases que existen van a ser principalmente entre las clases controladoras y las clases del modelo. Esto indica que el acceso a los datos o modificación de los mismos en las clases que representan el modelado de los datos va a venir gobernada por las clases controladoras, ya que ese es su cometido. La minoría restante de relaciones que se observan entre clases que pertenecen al modelo no son más que relaciones de asociación que indican que una clase forma parte de otra como atributo o miembro de clase, o bien que necesita información de otra para su funcionamiento, puesto que manejan a veces gran cantidad de datos. De esta forma se maximiza la cohesión y minimiza el acoplamiento entre clases, logrando un software orientado a objetos de calidad.

Para todas las clases que se han descrito en Java, se ha definido un constructor de clase vacío, el denominado constructor por defecto, además de un constructor con parámetros, aprovechando las propiedades de la sobrecarga de métodos de los lenguajes orientados a objetos. Para reducir el tamaño de los diagramas, se han obviado los métodos denominados *setters* y *getters* para la accesibilidad de los atributos de clase, que se definen de tipo privado para cumplir con el encapsulamiento u ocultación de la información. Por último,

con la herramienta de desarrollo de UML que se ha empleado, además de los atributos que aparecen dentro de la parte gráfica de la clase a tal fin, son también atributos aquellas asociaciones que parten de la clase en estudio.

4.5.2.3.1. Paquete del Inicio: *Main Package*

En este paquete únicamente se encuentra la clase inicial del arranque de la aplicación, como se observa en la figura 4.25, y debido a que no presenta ningún tipo de lógica de control ni manejo de datos importantes para la aplicación, se ha decidido que perteneciera a un paquete o carpeta diferente de las que se presentarán más adelante.

Figura 4.25. Contenido del paquete de Inicio: *Main Package*

La clase *Main* pertenece al paquete *mainPackage*, como se muestra en la figura 4.26, presenta dos relaciones de agregación con las clases *InitialTimerListener*, del paquete *utilitiesPackage*, y *SplashWindowInit*, del paquete *viewPackage*. Estas relaciones de asociación indican que estas clases forman parte de ella como atributos de clase. Esto se debe a que la clase *Main* utiliza una ventana emergente con el logo de la aplicación durante un tiempo dado por el retardo del temporizador de tipo *Timer* de Java, que también es su atributo, y cuando se agote el tiempo fijado para el temporizador se ejecutará el código asociado de la clase *InitialTimerListener* que, concretamente, creará la ventana principal de la aplicación.

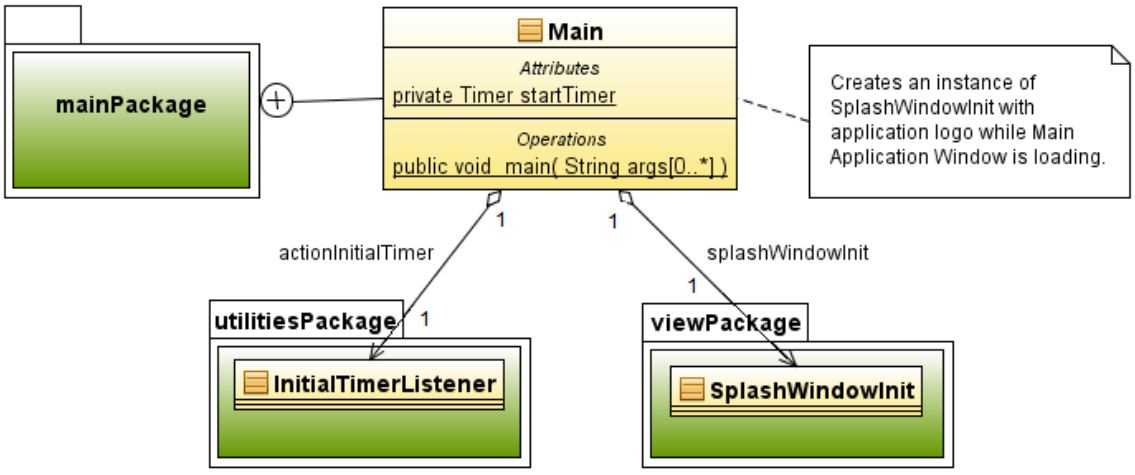


Figura 4.26. Diagrama de clases – *Main*

4.5.2.3.2. Paquete del Modelo: *Model Package*

En este apartado se van a ir presentando todos aquellos diagramas que contengan las clases que se encargan del modelado de los datos que maneja la aplicación. Como se puede apreciar en la figura 4.27, el paquete del modelo o *modelPackage* contiene cuarenta clases que van a contener toda la información que se necesita para realizar la animación y mostrar por pantalla los estadísticos, los eventos de mensajes en los nodos, los eventos de cambios de dirección de los nodos, y el contenido de las cachés locales y las cachés utilizadas para la realización de las redirecciones.

Los diagramas que contienen las clases que pertenecen al modelado de los datos se han ido realizando mediante agrupaciones de clases que pueden estar más relacionadas por la funcionalidad que representan o por la información que manejan.

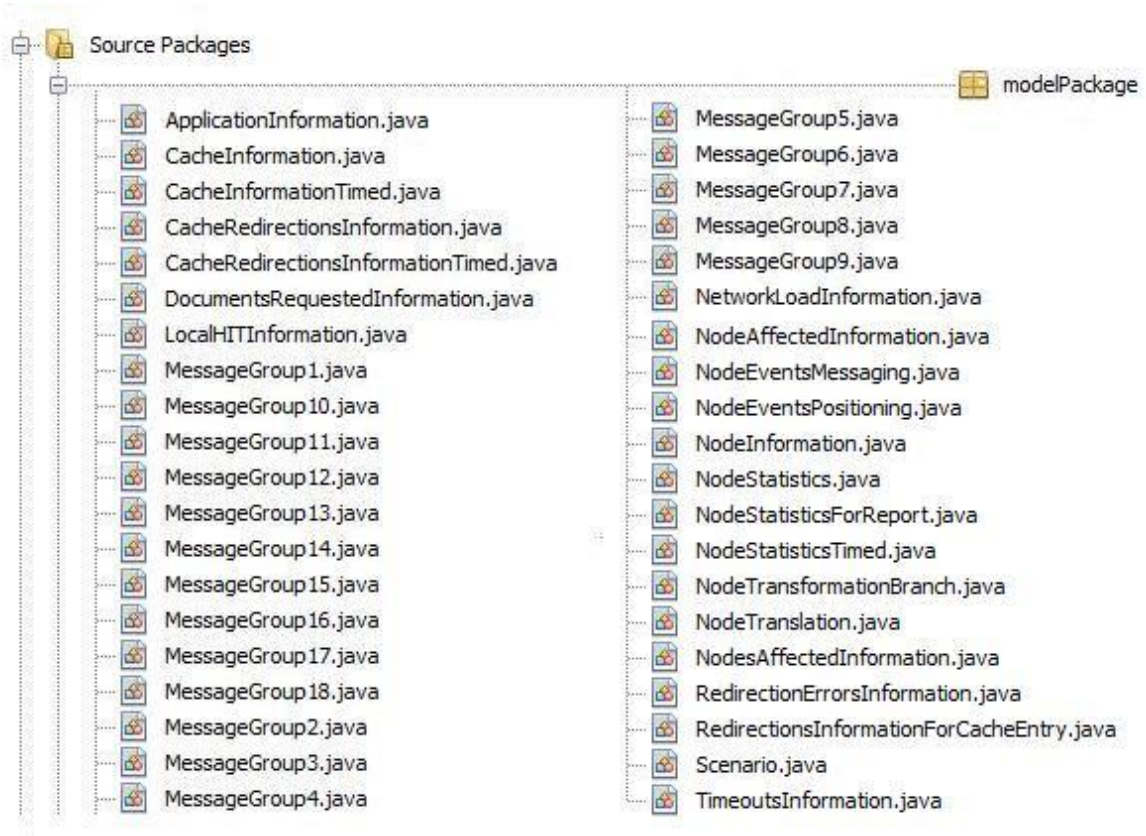


Figura 4.27. Contenido del paquete del Modelo: *Model Package*

Formato de mensajes base y formato de cambios de dirección

El diagrama de clases presentado en la figura 4.28 contiene la información con el formato de los eventos de los nodos que se van a cargar de los archivos de entrada de la aplicación. Así, los eventos que provengan del archivo de movilidad de los nodos *.pos* serán registrados con objetos de la clase *NodeEventsPositioning* y los eventos de mensajes

que provengan del archivo de salida de la simulación *.txt* serán registrados con objetos de clases hijas de la clase *NodeEventsMessaging*, la cual contiene la información común a todos esos mensajes. Para ambas clases se pueden observar los diferentes constructores de clase implementados, así como el método *toString*, método utilizado por defecto por Java para representar el contenido de un objeto, que es sobreescrito para que muestre dicha información de forma organizada. Este método será sobreescrito para la gran mayoría de clases del modelado de datos, puesto que Java por defecto devuelve un código con la referencia al objeto, sin presentar ningún tipo de relación con la información que contiene.

Otro de los métodos definidos en Java que se utiliza para las comparaciones de objetos y que se ha sobreescrito para una gran cantidad de clases es el método *equals*, ya que las comparaciones de objetos que se realicen deberán atender a un atributo o conjunto de atributos en particular, y no comprobará las referencias de los objetos, como hace este método de Java por defecto. Esto será de gran utilidad para comparar y extraer objetos de las colecciones de elementos.

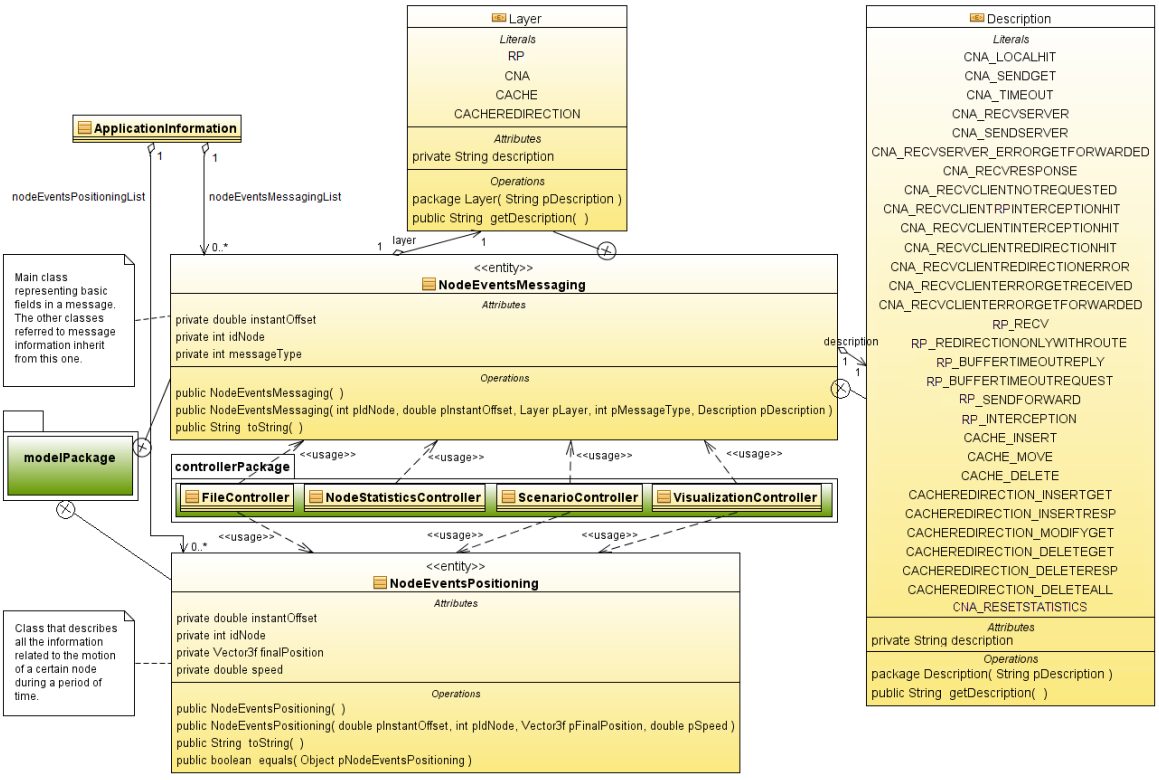


Figura 4.28. Diagrama de clases - Formato de mensajes base y formato de cambios de dirección

En el mismo diagrama se puede observar cómo se han definido dos clases internas dentro de *NodeEventsMessaging*, necesarias porque ésta posee dos atributos que son objetos de dichas clases. Estas dos clases internas, que son de tipo enumerado, se

denominan *Description* y *Layer*. Así, *Layer* indicará la capa o, más concretamente, la subcapa de la pila de protocolos donde se generan dichos mensajes, o bien el tipo de caché que se está actualizando. Los posibles valores que tomará son: RP (*Routing Protocol*), indicando que se trata de un mensaje de la subcapa dedicada al encaminamiento; CNA (*CacheNode Application*), indicando que el mensaje se ha producido en la capa de aplicación de la pila de protocolos; CACHE, indicando que se refiere a un mensaje de actualización de la caché local de un nodo; y CACHEREDIRECTION, indicando que se refiere a un mensaje de actualización de la caché dedicada a la información para la realización de redirecciones. El abanico de valores que puede presentar el tipo enumerado *Description* corresponde a los diferentes tipos de mensajes que se han adjuntado en el Apéndice A. Además de los atributos de tipo *Layer* y *Description*, *NodeEventsMessaging* contiene otros: *idNode*, identificador del nodo donde se produce el evento; *instantOffset*, instante en el que se produce el evento desde el inicio de la simulación; y *messageType*, que contiene un número identificativo del tipo de mensaje. Los atributos de la clase *NodeEventsPositioning* son: *idNode*, identificador del nodo donde se produce el evento; *instantOffset*, instante en el que se produce el evento desde el inicio de la simulación; *finalPosition*, punto final del movimiento rectilíneo que debe seguir el nodo; y *speed*, velocidad con la que debe desplazarse el nodo en ese movimiento rectilíneo.

Los mensajes utilizados en la aplicación que vienen definidos en el apartado A.2.3.9 (inicio de estadísticos – fin del calentamiento de las cachés locales) del Apéndice A, son del tipo base *NodeEventsMessaging*.

Finalmente, queda por añadir que las clases controladoras *FileController*, *NodeStatisticsController*, *ScenarioController* y *VisualizationController* establecen relaciones de uso de estas clases del modelo aquí presentadas, puesto que manejan sus datos, además de la clase *ApplicationInformation*, que establece relaciones de agregación con estas clases puesto que se utilizan en atributos que son colecciones de datos.

Formato de mensajes (I)

Los diferentes tipos de mensajes que proceden del archivo de salida de la simulación (*.txt*) se han agrupado de forma que aquellos tipos que contengan los mismos campos se podrán representar con instancias de la misma clase. Estas clases tendrán el nombre de *MessageGroupX*, y todas ellas heredarán del tipo base *NodeEventsMessaging* o bien de sus clases hijas, añadiendo además otros atributos.

Las clases con grupos de mensajes que aparecen en la figura 4.29 son clases hijas de *NodeEventsMessaging*. Todas ellas contienen como métodos el constructor por defecto (constructor vacío), el constructor con parámetros, y la sobreescritura de *toString*. En el diagrama además se aprecia cuáles de las clases controladoras manejan sus datos a través de relaciones de uso, que son *VisualizationController*, *NodeStatisticsController* y *FileController*.

MessageGroup1 es la clase utilizada para representar los mensajes de los tipos presentados en los apartados A.2.2.1 (recepción de una petición GET en el lado del servidor), A.2.3.3 (intercepción de una petición GET de forma remota), A.2.3.4 (Intercepción de una petición GET con recursos de encaminamiento), A.2.3.5 (redirección de una petición GET), A.2.3.6 (error de redirección) y A.2.3.7 (recepción de un error de redirección en el nodo originario de la petición) del Apéndice A. Los atributos que añade son los identificadores del nodo origen, del nodo destino, y del documento.

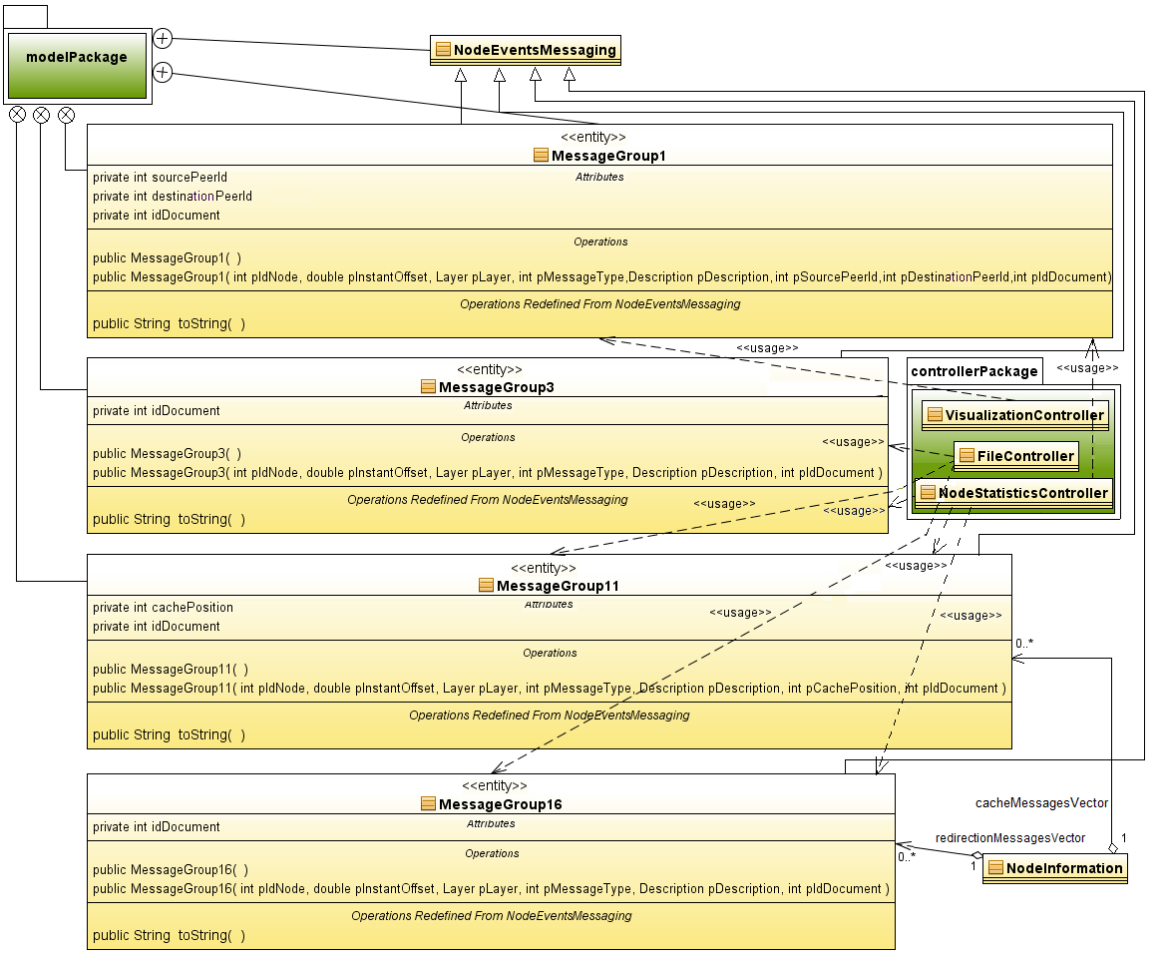


Figura 4.29. Diagrama de clases - Formato de mensajes (I)

MessageGroup3, que es la clase que representa los mensajes del tipo presentado en el apartado A.2.1.3 (*timeout* de recepción) del Apéndice A, añade un nuevo atributo para la identificación del documento al que se refiere este mensaje.

Los mensajes que se utilizan para la actualización de los datos de la caché local de los nodos van a ser del tipo *MessageGroup11* o bien de tipos definidos en sus clases hijas. Concretamente, el mensaje del tipo explicado en el apartado A.2.5.3 que se trata del borrado de un documento de la caché local queda definido con *MessageGroup11*. Además de los atributos heredados de *NodeEventsMessaging*, este tipo añade los atributos para la identificación de la posición en la memoria y del documento. Esta clase además forma parte de la clase *NodeInformation*, como se observa con la relación de agregación, puesto que esta última posee un atributo que alberga una colección de mensajes de actualización de caché local para obtener el contenido de la misma en un instante determinado.

Por último, los mensajes que indican la actualización de la caché de redirecciones en un nodo van a ser del tipo definido por *MessageGroup1*6 o por sus clases hijas. Los mensajes que son del tipo *MessageGroup16* son los encargados del borrado de entradas o parte de entradas de la caché de redirecciones, que se presentan en el Apéndice A en los apartados A.2.6.4 (borrado del registro GET de una entrada de la caché de redirecciones), A.2.6.5 (borrado del registro RESP de una entrada de la caché de redirecciones) y A.2.6.6 (borrado de una entrada de la caché de redirecciones). Se añade un nuevo atributo respecto a los heredados de la clase padre *NodeEventsMessaging*, que es el identificador del documento. Como ocurre con la clase *MessageGroup11*, esta clase forma parte a través de una relación de agregación de la clase *NodeInformation*, puesto que ésta alberga estos mensajes de actualización de la caché de redirecciones para obtener su contenido en un instante dado.

Formato de mensajes (II)

Como ha ocurrido en el diagrama de clases presentado en la figura 4.29, en el diagrama de la figura 4.30 se observan distintas clases que representan grupos de mensajes que contienen los mismos campos o información. Todos ellas son clases hijas de *MessageGroup1*, definida en el diagrama de clases de la figura 4.29. Además, en todas ellas se aprecian los atributos particulares que agregan, además de los que heredan, y los métodos de clase utilizados, que son el constructor por defecto (constructor vacío o sin parámetros), el constructor con parámetros, y la sobreescritura del método *toString*, como en la gran mayoría de clases que aparecen en estos diagramas. Se puede apreciar también cómo se establecen relaciones de uso desde algunas de las clases controladoras del paquete

controllerPackage, como son *VisualizationControlle*r, *NodeStatisticsController* y *FileController*.

MessageGroup2 define los mensajes del tipo presentado en el apartado A.2.1.2 (petición GET) del Apéndice A. Además de los atributos heredados de *MessageGroup1*, añade el atributo que almacena el identificador del número de petición realizada.

MessageGroup6 define los mensajes del tipo presentado en el apartado A.2.2.2 (envío de la respuesta a una petición GET en el lado del servidor de datos) del Apéndice A. En este caso, añade los atributos que se encargarán de manejar la información acerca del tamaño del documento y su tiempo de vida.

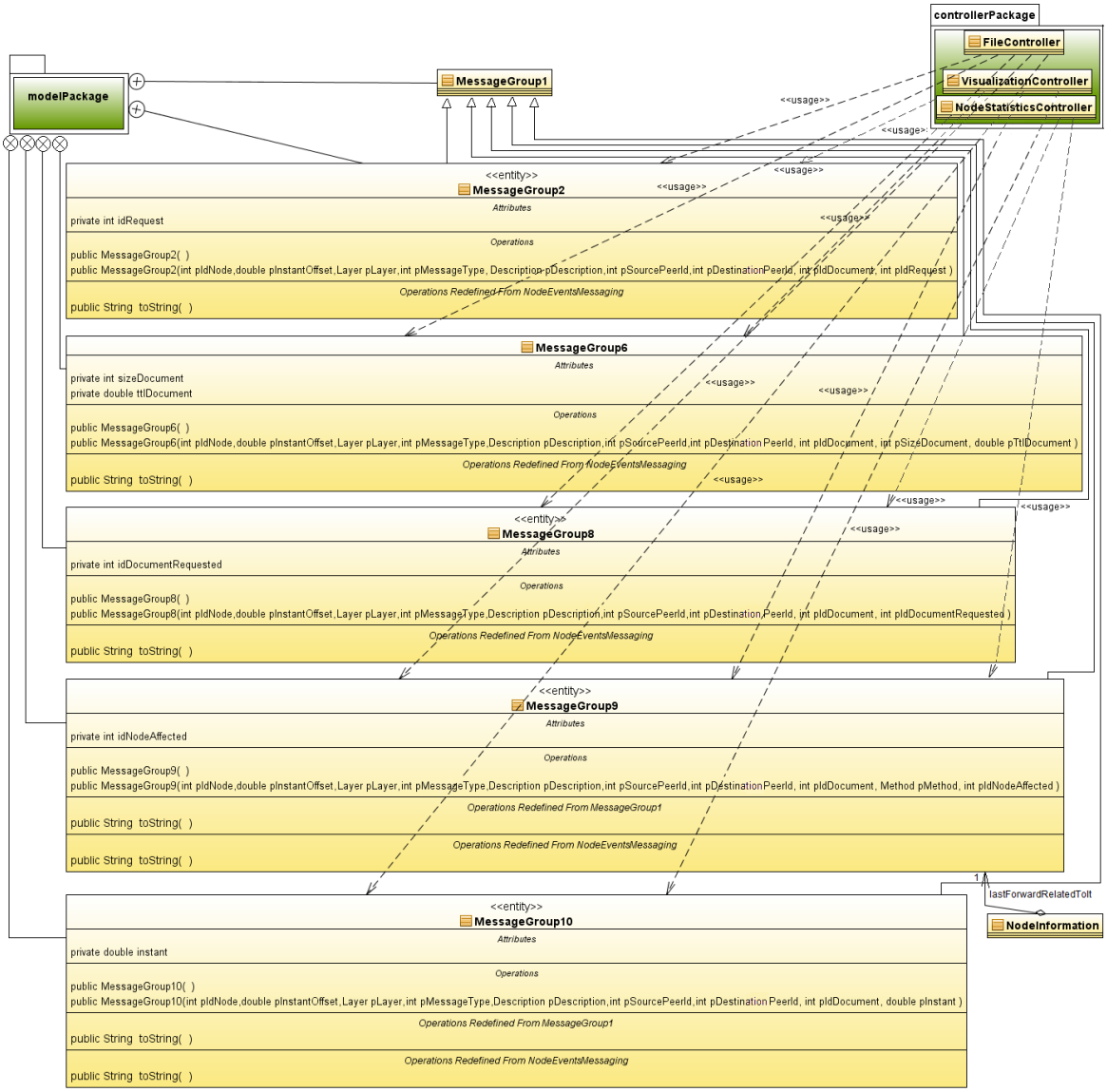


Figura 4.30. Diagrama de clases - Formato de mensajes (II)

MessageGroup8 establece la información que llevan los mensajes del tipo presentado en el apartado A.2.3.2 (recepción de una respuesta con un documento que no se ha solicitado) del Apéndice A. Necesita además de un nuevo atributo que acoja la información del identificador del documento.

MessageGroup9 contiene los datos necesarios para almacenar la información que necesitan los mensajes de los tipos presentados en los apartados A.2.4.1 (recepción de un mensaje) y A.2.4.5 (retransmisión de un mensaje). Además de los atributos que hereda, define el identificador de nodo que generó la petición y que causa estos mensajes y el método utilizado, es decir, si se trata de un mensaje de transmisión de una petición, de una respuesta, o de un error de redirección. Hay que destacar que la clase *NodeInformation* posee un atributo que es instancia de esta clase, y que alberga el último mensaje del tipo *forward* que se ha transmitido y que está relacionado con la petición de un documento realizada por el nodo en particular.

Por último, *MessageGroup10* establece la información que llevan los mensajes de los tipos presentados en los apartados A.2.4.3 (eliminación de mensajes del *buffer* debido a una respuesta de ruta) y A.2.4.4 (eliminación de mensajes del *buffer* debido a una petición de ruta). En este caso se añade un nuevo atributo que acogerá la información acerca del instante en que se ha producido la petición del documento que causa estos mensajes.

Formato de mensajes (III)

El diagrama de clases mostrados en la figura 4.31 contiene el tercer agrupamiento de aquellas clases que representan la información cargada desde el archivo de salida de la simulación con los mensajes intercambiados por la red. En él se aprecian clases que heredan de otras clases, como ha ocurrido en los dos diagramas anteriores. Además, se vuelven a tener los mismos constructores (constructor por defecto y constructor con parámetros) y el método *toString* sobreescrito con la información del mensaje a mostrar.

Los controladores que establecen relaciones de uso con estas clases vuelven a ser *NodeStatisticsController*, *VisualizationController* y *FileController*, del paquete *controllerPackage*.

MessageGroup14 define los mensajes del tipo presentado en los apartados A.2.2.3 (retransmisión de un error de redirección en el lado de un servidor) y A.2.3.8 (retransmisión de un error de redirección en el lado de un cliente) del Apéndice A, hereda

de la clase *MessageGroup1* y además añade un nuevo atributo para almacenar la información acerca del identificador del nodo que originó la petición del documento.

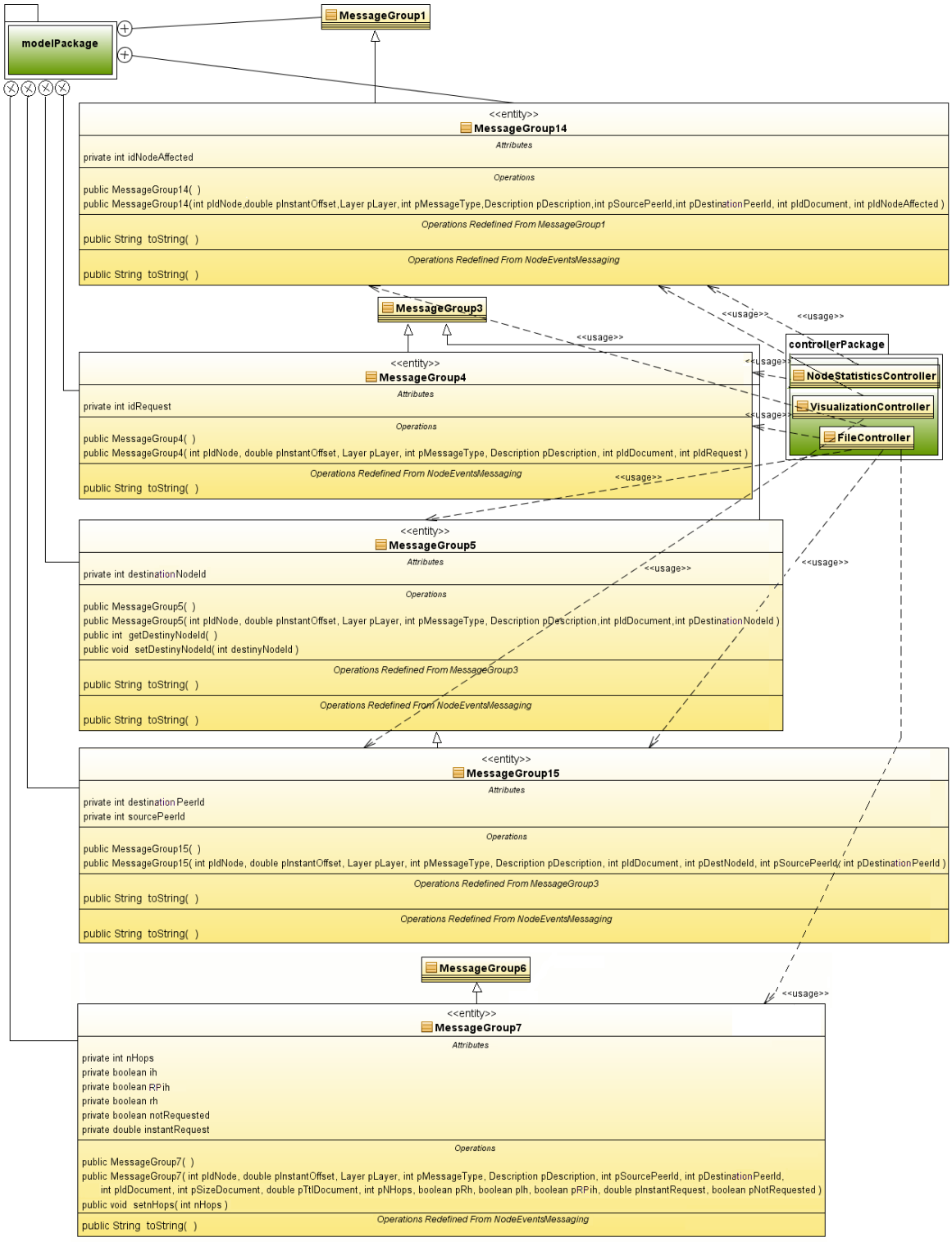


Figura 4.31. Diagrama de clases - Formato de mensajes (III)

MessageGroup4 y *MessageGroup5* heredan de la clase *MessageGroup3*, anteriormente presentada en la figura 4.29. *MessageGroup4* establece la información que lleva el mensaje del tipo presentado en el apartado A.2.1.1 (acierto en caché local) del Apéndice A, al igual

que ocurre con *MessageGroup5* y el tipo de mensaje del apartado A.2.4.6 (intercepción con recursos de encaminamiento de una petición a nivel RP). Ambos necesitan además de un nuevo atributo; así, en *MessageGroup4* se define el identificador de la petición realizada por un nodo y en *MessageGroup5* se crea un atributo de tipo entero que recoja la información del identificador del nodo que representa el nuevo destino de una petición.

MessageGroup15 define el mensaje del tipo presentado en el apartado A.2.4.2 (redirección de una petición a nivel RP) del Apéndice A, hereda de la clase *MessageGroup5* que se acaba de explicar y además añade dos nuevos atributos para almacenar la información acerca de los identificadores del nodo que originó la petición del documento y del nodo al que se lo había solicitado.

Finalmente, *MessageGroup7* contiene los datos necesarios para almacenar la información que necesita el mensaje del tipo presentado en el apartado A.2.3.1 (recepción de una respuesta RESP) del Apéndice A y es hija de la clase *MessageGroup6*. Además de los atributos que hereda define algunos nuevos, como el número de saltos por los que ha pasado la petición hasta ser servida, el instante de tiempo en que se produjo la petición a la que corresponde este mensaje, y cuatro variables booleanas que indican si se ha producido una intercepción remota, una intercepción con recursos de encaminamiento, una redirección o si se trata de un documento que no se solicitó.

Formato de mensajes (IV)

El diagrama de clases mostrados en la figura 4.32 contiene el cuarto agrupamiento de clases que representan la información cargada desde el archivo de salida de la simulación con los mensajes intercambiados por la red. En él se pueden apreciar cuatro clases diferentes y que para ellas se vuelven a tener los mismos constructores (constructor por defecto y constructor con parámetros) y el método *toString* sobreescrito con la información del mensaje a mostrar. Los controladores que establecen relaciones de uso con estas clases son *NodeStatisticsController* y *FileController*, del paquete *controllerPackage*.

MessageGroup12 define el mensaje del tipo presentado en el apartado A.2.5.1 (inserción de un documento en la caché local) del Apéndice A, hereda de la clase *MessageGroup11* que se ha explicado anteriormente y además añade varios atributos nuevos, como el tamaño del documento, el número de veces que ha sido solicitado, el instante en el que expira éste y la evaluación realizada por la política de reemplazo.

MessageGroup13 define los mensajes del tipo presentado en el apartado A.2.5.2 (cambio de posición de un documento en la caché local) del Apéndice A, hereda de la clase *MessageGroup12* y además añade un nuevo atributo para almacenar la información acerca de la posición antigua que tenía el documento antes de ser desplazado.

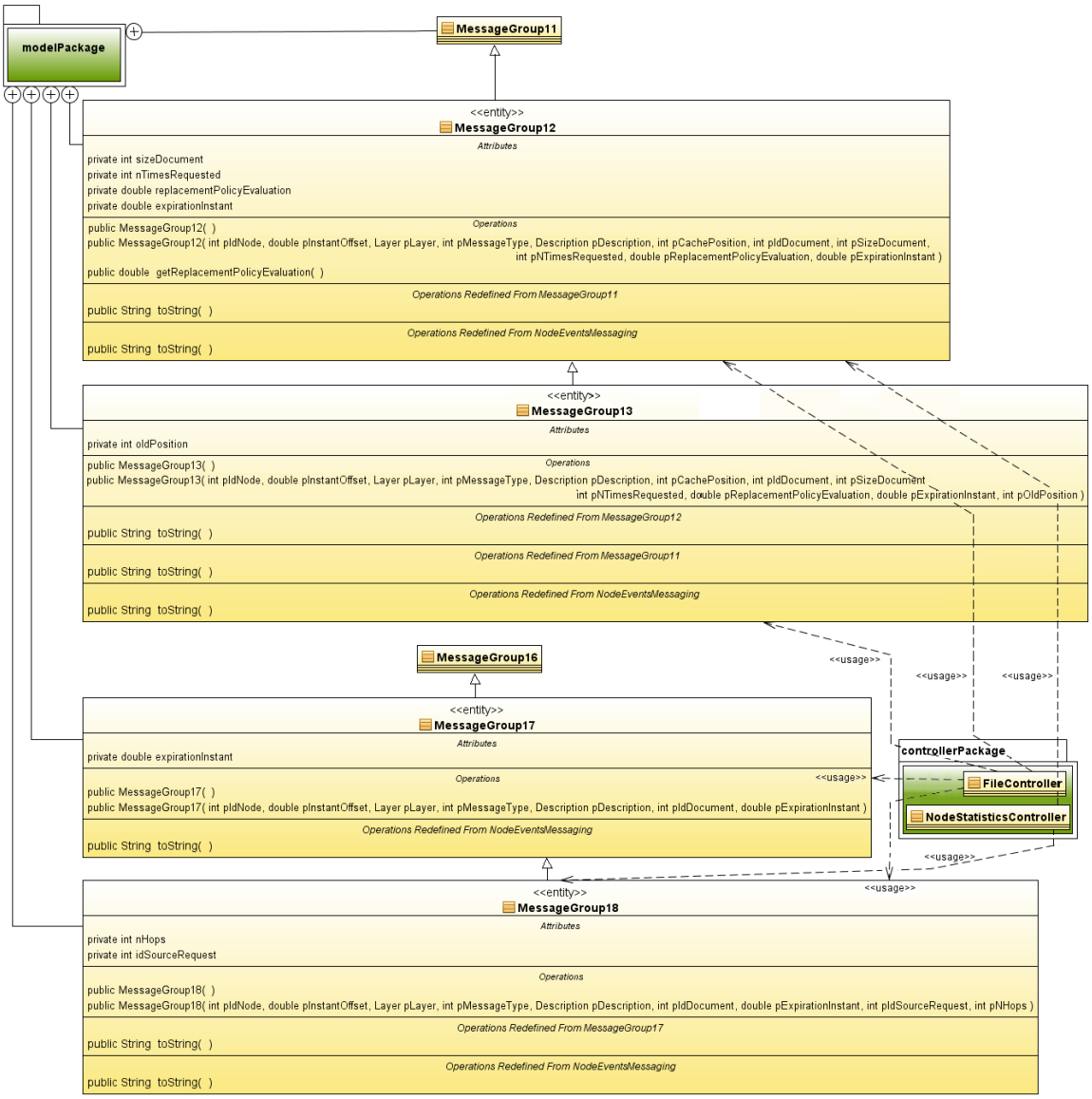


Figura 4.32. Diagrama de clases - Formato de mensajes (IV)

MessageGroup17 contiene los datos necesarios para almacenar la información que necesita el mensaje del tipo presentado en el apartado A.2.6.3 (modificación de la información de expiración de un documento en su registro GET de la caché de redirecciones) del Apéndice A y es hija de la clase *MessageGroup16*. Además de los atributos que hereda, define uno nuevo para acoger la información acerca del tiempo de expiración del documento que debe ser modificado.

Por último, *MessageGroup18* establece la información que llevan los mensajes de los tipos presentados en los apartados A.2.6.1 (inserción de la información de un documento en el registro GET de la caché de redirecciones) y A.2.6.2 (inserción de la información de un documento en el registro RESP de la caché de redirecciones). En este caso se añaden dos nuevos atributos que acogerán la información acerca de la distancia en número de saltos a la que se encuentra el documento y del identificador del nodo que posee o poseerá el documento.

Manejo de estadísticos

En el diagrama de clases de la figura 4.33 se pueden observar las diferentes clases que son clave para el manejo de la evolución de los valores estadísticos que poseen los nodos a lo largo del tiempo que dura la simulación. Además, contiene la clase utilizada para la muestra de la información de estadísticos en los informes PDF generados.

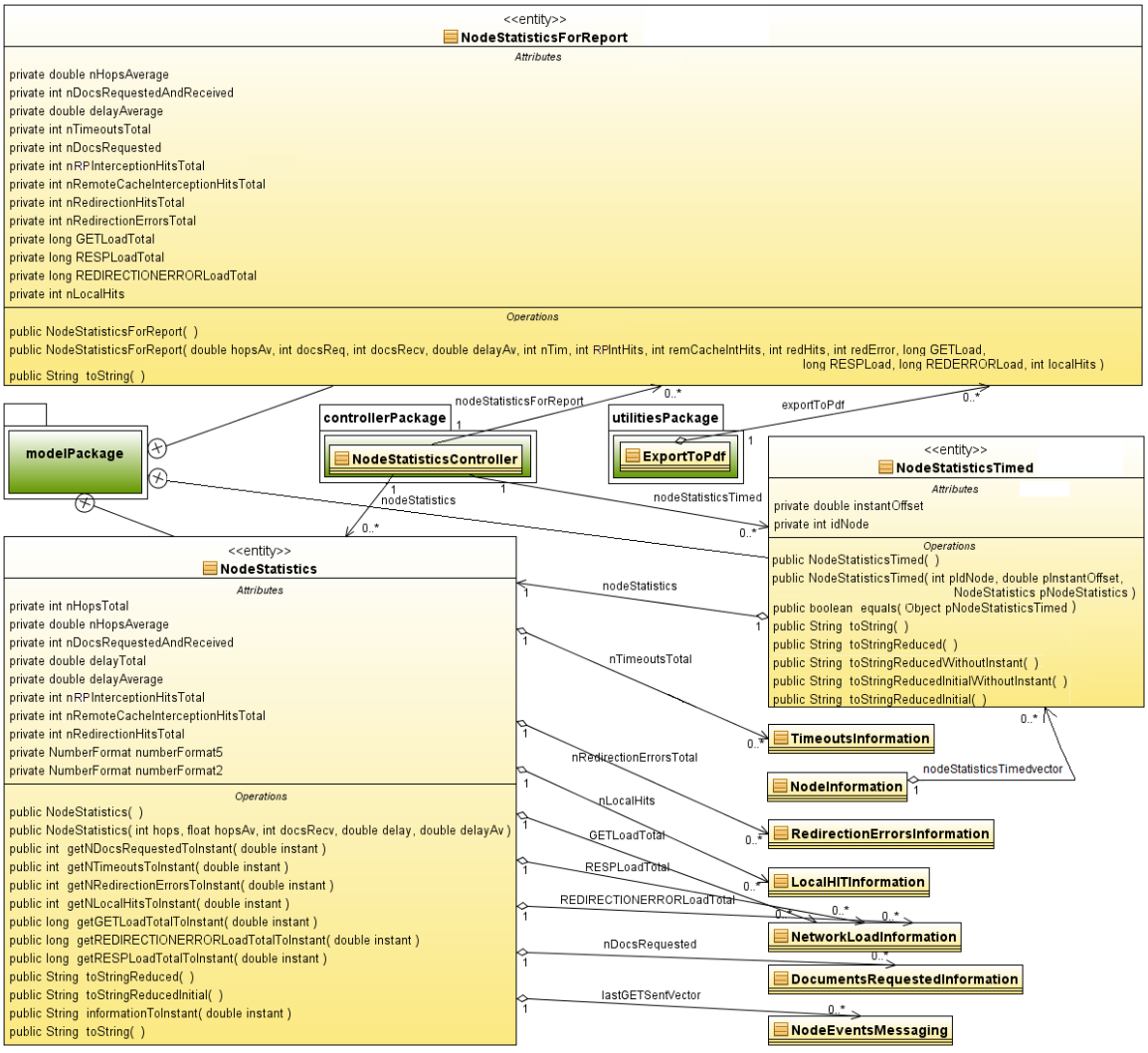


Figura 4.33. Diagrama de clases - Manejo de estadísticos

En primer lugar, la clase *NodeStatistics* contiene explícitamente algunos de los atributos necesarios para representar los valores estadísticos en estudio en este proyecto para un nodo en particular de la red, como son el número de saltos medio que realiza una petición hasta que se sirve el documento, número total de documentos recibidos, retardo medio que sufren las peticiones hasta que son cursadas, número total de intercepciones remotas que experimentaron los documentos recibidos, número total de intercepciones con recursos de encaminamiento que experimentaron los documentos recibidos, y el número total de redirecciones que experimentaron los documentos recibidos. Además de estos atributos anteriores que aparecen dentro de la parte gráfica de la clase a tal fin, como se ha comentado anteriormente en la introducción a los diagramas de clases, son atributos aquellas asociaciones de tipo agregación que parten de la clase en estudio. Así, el número total de *timeouts* en las peticiones realizadas, el número total de errores de redirección que se han producido en las peticiones de un nodo, el número total de aciertos en caché local, la carga total que introduce un nodo en la red debido a peticiones, respuestas o errores de redirección, y el número total de documentos solicitados, suponen atributos que están expresados con las relaciones de asociación que existen con la clase *NodeStatistics*. Como estos últimos atributos consisten en colecciones de elementos, existen métodos para cada uno de ellos que devuelven un valor buscado en función del instante de tiempo pasado como parámetro. La clase *NodeStatisticsTimed* contiene la información de los estadísticos comentada anteriormente puesto que posee un atributo de tipo *NodeStatistics*, además del identificador del nodo y del instante de tiempo para el cual esos estadísticos son válidos. Así, la clase *NodeInformation*, que contiene la información acerca de un nodo en particular, poseerá una colección de elementos del tipo *NodeStatisticsTimed* para poder almacenar una evolución de los valores estadísticos válidos en diferentes instantes de tiempo, de forma que cuando se soliciten en un instante determinado, no se deba comenzar a calcular desde el instante inicial de la simulación. Esta decisión de diseño ha sido motivada por la necesidad de obtener esos estadísticos con retardos mínimos.

NodeStatisticsForReport es la clase encargada de proporcionar, durante la creación del PDF, los datos que se van a mostrar en el informe, a modo de interfaz entre la clase del paquete de utilidades que se encarga de la creación del archivo PDF y las clases que manejan los estadísticos, proporcionando únicamente los valores numéricos válidos.

En el mismo diagrama aparecen las clases controladoras que manejan esos datos, así como otras clases del modelo o de utilidades que las utilizan para algún fin. Así, el controlador de estadísticos *NodeStatisticsController* posee relaciones de asociación

unidireccionales con las tres clases presentadas, ya que posee un atributo de cada uno de esos tipos. Además, la clase *ExportToPDF*, como se acaba de comentar, posee un atributo del tipo *NodeStatisticsForReport* del cual toma la información para escribirla en el archivo de salida.

Información de estadísticos

En el diagrama de clases de la figura 4.34 se presentan las clases utilizadas para la gestión de la información de estadísticos que se ha introducido en el diagrama de clases de la figura 4.33, además de las clases empleadas para determinar en cada instante de tiempo qué nodos están relacionados con otros nodos de la animación. Respecto al tema de los estadísticos, en el diagrama se puede observar la definición de las clases *RedirectionErrorsInformation*, *TimeoutsInformation*, *NetworkLoadInformation*, *DocumentsRequestedInformation* y *LocalHITInformation*. En ellas siempre se presenta un atributo indicando el valor del estadístico en cuestión y otro atributo que indica el instante de tiempo para el cual dicho valor es válido. Todas esas clases además presentan relaciones de asociación con el controlador *NodeStatisticsController*, debido a que para el cálculo de estadísticos se crean atributos que alberguen valores temporales para los datos que se le van solicitando a este controlador.

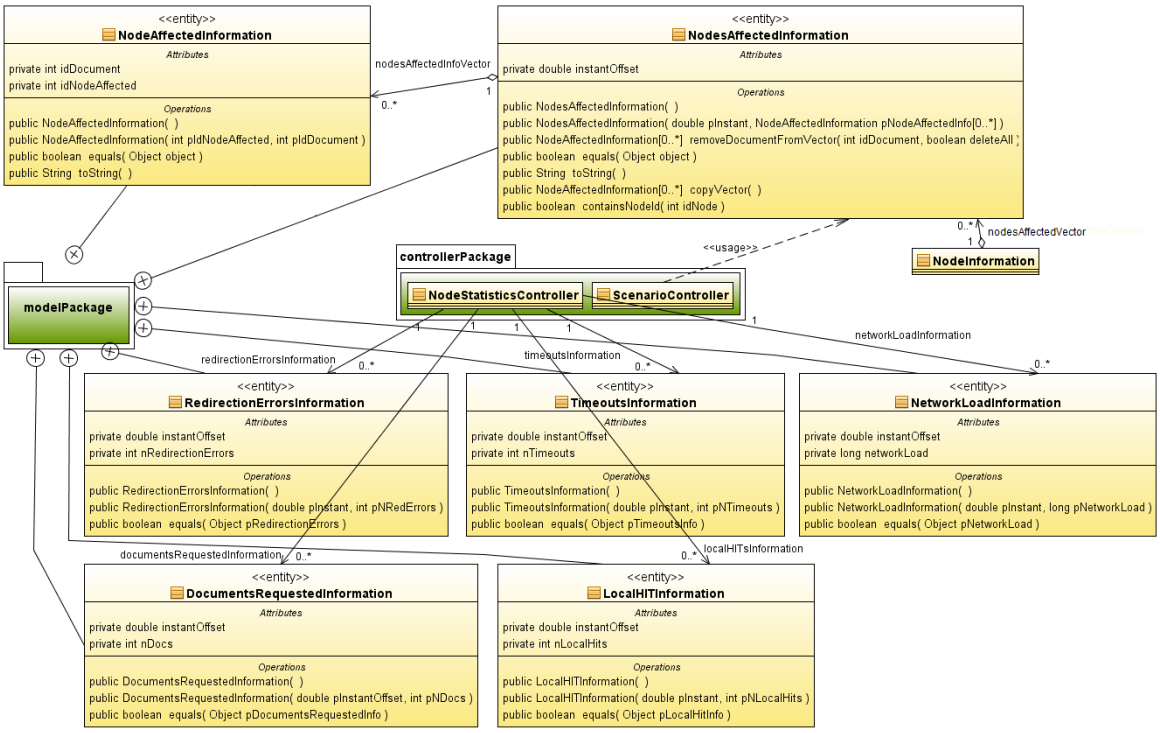


Figura 4.34. Diagrama de clases - Información de estadísticos

Respecto a las dos clases restantes que aparecen en el diagrama, es decir, *NodeAffectedInformation* y *NodesAffectedInformation* hay que destacar que proveen la información utilizada para que el controlador del escenario gráfico represente, en forma de halo, aquellos nodos que están involucrados en la petición realizada por un nodo que se encuentra seleccionado. Así, la clase *NodeAffectedInformation* indica a través de qué documento solicitado, un nodo está relacionado con el nodo que lo solicitó, como se observa en sus atributos. La clase *NodesAffectedInformation* contiene como atributos una colección de objetos de la clase anterior, indicando los nodos que están relacionados con otro, además del instante de tiempo para el que esta información es válida. El controlador del escenario establece una relación de uso con ella. Como se aprecia además, la clase *NodeInformation* contiene un atributo formado por una colección de elementos del tipo *NodesAffectedInformation*, que presenta la evolución temporal de los nodos que están relacionados con el nodo cuya información está contenida en la instancia de esa clase *NodeInformation*.

Información de la caché local y de la caché de redirecciones

Las clases que se encargan de gestionar la información utilizada para representar el contenido de la caché local y de la caché de redirecciones en un instante determinado de la animación y para un nodo dado se encuentran en el diagrama de clases de la figura 4.35. En ella se presentan cinco clases, entre las que se encuentran *CacheInformation* y *CacheRedirectionsInformation*, que se caracterizan porque implementan la interfaz *Comparable* de Java. Al implementar dicha interfaz se debe redefinir el método *compareTo* para realizar correctamente la ordenación de las entradas de la caché local y de la caché de redirecciones atendiendo al criterio que verifique los requisitos en cada caso. Así, la caché local estará organizada en entradas ordenadas por la posición que ocupan debido a la política de reemplazo que implementa, mientras que la caché de redirecciones presentará entradas con la información de cada documento organizadas por el identificador del mismo.

Con respecto a la información para el estudio de la caché local, se presenta la clase principal *CacheInformation*, donde se define la información que posee cada entrada de la memoria. Los atributos definidos para esta clase son la posición en la memoria, el identificador del documento que contiene dicha entrada, el tamaño del documento, el número de veces que ha sido accedido ese documento, la evaluación de la política de reemplazo y el instante de expiración del documento. Como en otras muchas clases, en esta clase se sobreescriben los métodos *toString* y *equals*. La clase *CacheInformationTimed*

contiene como atributo una colección de instancias de la clase anterior, emulando el contenido de la caché local, y añade un tributo para identificar el instante de tiempo en el cual todas esas entradas son válidas. Esta clase, además de los constructores, los métodos *getters* y *setters* y los métodos sobreescritos ya comentados en varias ocasiones, añade algunos métodos para agregar y eliminar una entrada de caché.

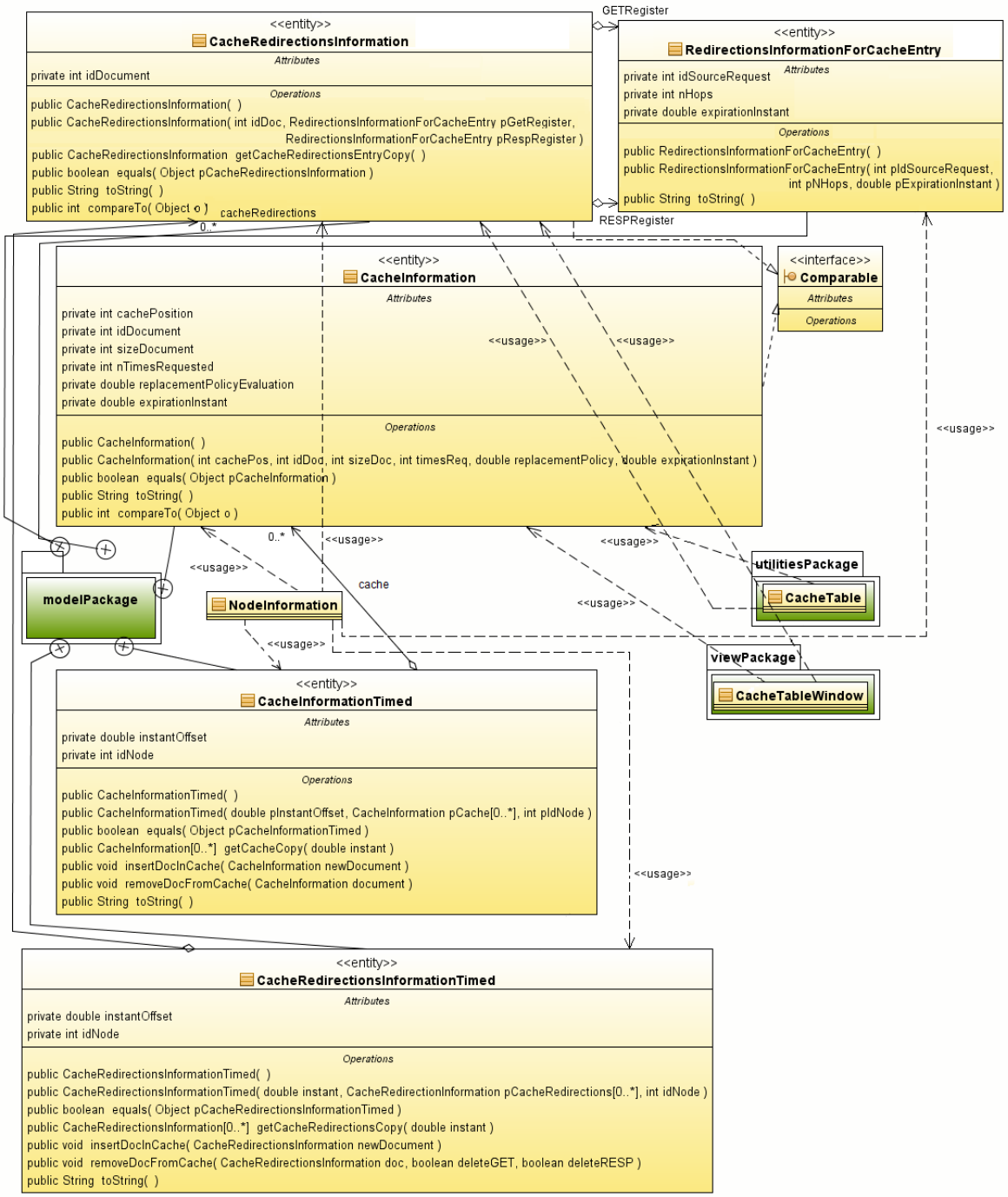


Figura 4.35. Diagrama de clases - Información de la caché local y de la caché de redirecciones

Con respecto a la caché de redirecciones, existen dos clases que definen su contenido. Una de ellas es *CacheRedirectionsInformation*, que contiene un primer atributo para el identificador del documento, y la otra clase, que define los atributos restantes, es *RedirectionsInformationForCacheEntry*, de la cual se crearán dos instancias para definir los registros GET y RESP relacionados con un documento. Estos registros podrán estar vacíos o no. Para emular el contenido completo de la caché de redirecciones se utiliza la clase *CacheRedirectionsTimed*, que de forma similar a lo que ocurre en la caché local, contiene una colección de elementos del tipo *CacheRedirectionsInformation*, además del instante temporal para el que es válido este conjunto de entradas. Además de los métodos que normalmente aparecen, se añaden nuevos métodos para agregar y eliminar una entrada de la caché de redirecciones.

Por último, en el diagrama se observan las relaciones de uso que se establecen con las clases que contiene.

Información de los nodos y de sus movimientos

A continuación se va a explicar la información que se mantiene de cada nodo a lo largo de la animación. Para ello, se definen las clases que aparecen en el diagrama de la figura 4.36. En dicho diagrama aparece la clase principal de información de los nodos, denominada *NodeInformation*. Los atributos que aparecen explícitamente en la representación gráfica de la clase representan la identificación del nodo, la posición inicial que ocupa tras cargar el escenario y el instante a partir del cual se calculan sus estadísticos. Además de estos, se observan una serie de relaciones de asociación que implican nuevos atributos.

El primer atributo que se define a través de una relación de asociación contiene una colección (en concreto, de tipo *Vector* de Java) de elementos que son instancias de la clase *NodeStatisticsTimed*, donde se tiene la evolución en el tiempo de los estadísticos de un nodo.

El siguiente atributo es también de tipo *Vector* y contiene una colección de elementos de tipo *MessageGroup11*, en el cual se almacenarán todos aquellos mensajes que sean de actualización del contenido de la caché local de ese nodo. Gracias al uso de genéricos de Java, se puede establecer que los elementos de una colección sean de un tipo determinado, detectando posibles errores debidos a la realización de cambios de conversión o *castings* en tiempo de compilación en lugar de que ocurran en tiempo de ejecución. Esto sucede gracias a que el compilador reconoce el tipo de elementos que deben ir en la colección, y

se fuerza a que sean listas homogéneas, evitando los errores mencionados anteriormente. En este caso, se ha utilizado la clase *MessageGroup11* porque los mensajes de actualización de la caché local son de este tipo o bien son instancias de clases hijas, que por las propiedades de la herencia de los lenguajes orientados a objetos pueden figurar en cualquier sitio donde aparezca la clase padre.

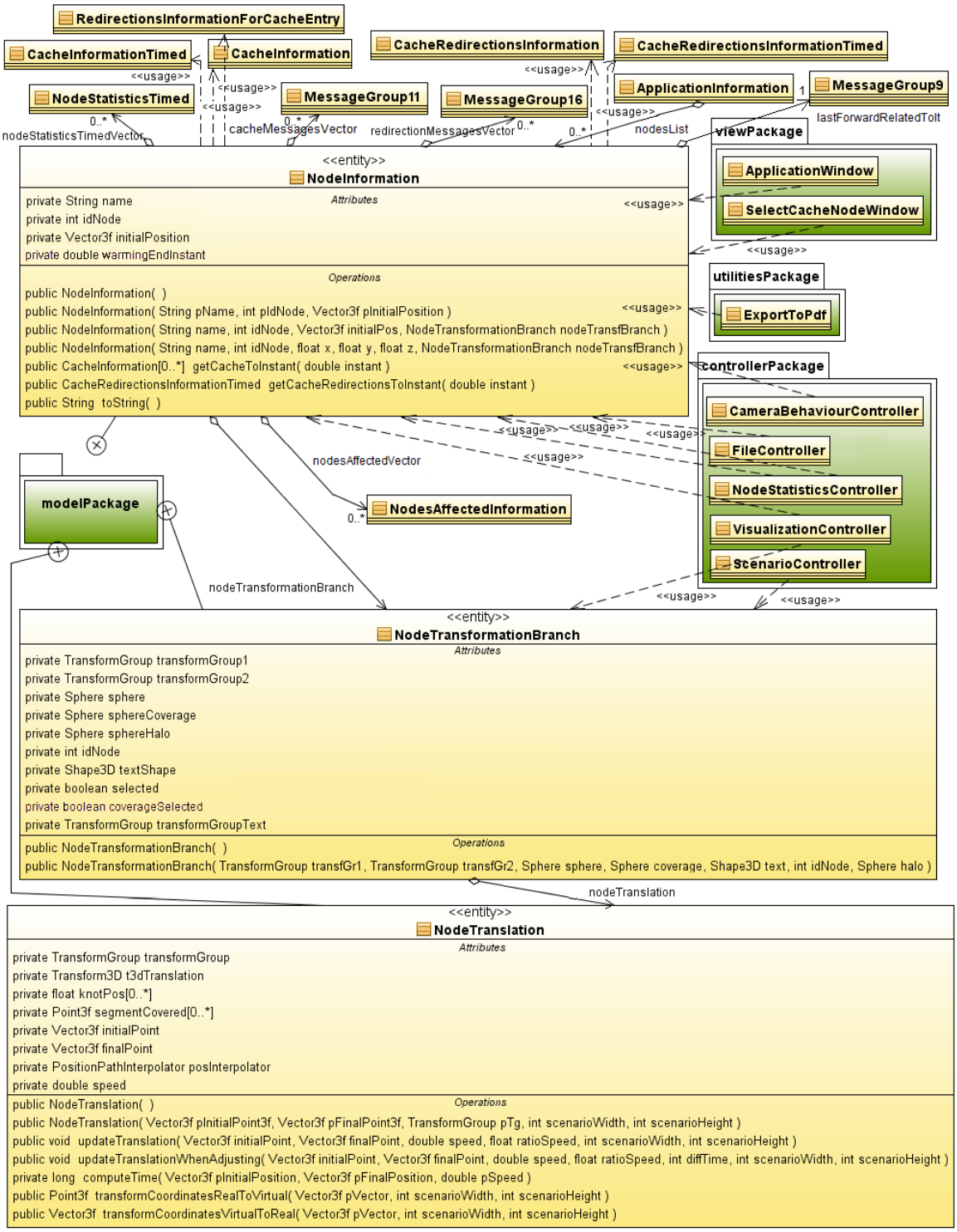


Figura 4.36. Diagrama de clases - Información de los nodos y de sus movimientos

De la misma forma que sucede con la caché local, para la caché de redirecciones se establece un nuevo atributo que representa un *Vector* de elementos del tipo *MessageGroup16*, puesto que los mensajes de actualización de la misma van a ser instancias de esa clase o de sus clases hijas.

Existe otra relación de asociación con la clase *MessageGroup9*, que define un atributo que contendrá el último mensaje de este tipo que se ha enviado por la red y que está relacionado con una petición que ha realizado este nodo; de esta manera, se podrá obtener la información necesaria para realizar la animación de los mensajes que van por la red y que están relacionados con este nodo.

Otro atributo de esta clase contiene la información de la evolución temporal de los nodos que están interviniendo en el envío de la petición de un documento por parte de este nodo y en su posible respuesta. Esta información, que se recoge en una colección de tipo *Vector* de elementos de la clase *NodesAffectedInformation*, será utilizada para representar los halos de los nodos que se relacionen directamente con nodos que se encuentren seleccionados en un instante dado.

El último atributo de *NodeInformation* (relacionado con las otras dos clases del diagrama *NodeTransformationBrach y NodeTranslation*), contiene la información gráfica de los movimientos y las rotaciones que realiza el nodo por el escenario, además de la información sobre la apariencia de las esferas gráficas que representan al propio nodo, a su cobertura y a su halo y el aspecto del texto que muestra su identificador numérico. Este atributo es una instancia de la clase *NodeTransformationBrach*, puesto que para el diseño de esta aplicación se ha dividido la información gráfica de cada nodo en ramas diferentes de la estructura en árbol del grafo de escena de los programas que utilizan Java3D.

La clase *NodeTransformationBrach* contiene diversos atributos utilizados para la animación de los nodos en la red. En primer lugar, posee tres atributos de tipo *TransformGroup* (de la API Java3D) que se utilizan para las traslaciones y rotaciones de las esferas y del texto identificativo por la red. Además tiene tres instancias de la clase *Sphere* (de la API Java3D) para representar las esferas que corresponden al nodo, a la cobertura y al halo. Para representar el texto en 3D se utiliza la clase *Shape3D*. Para detectar si en un instante determinado se tiene un nodo seleccionado o su cobertura, se definen dos variables booleanas a tal fin.

El último atributo definido en la clase *NodeTransformationBrach* es una instancia de la clase *NodeTranslation*. Esta clase es la que se va a dedicar exclusivamente a llevar la

información de los movimientos rectilíneos que siguen los nodos por la red. Así, se definen dos atributos que contienen los puntos de inicio y de fin del movimiento con las coordenadas del escenario real, además de un atributo para la velocidad a la que se deben desplazar, teniendo en cuenta el ratio entre la velocidad de la animación y la de la simulación de base. El atributo más importante y que a su vez define el resto de atributos es la instancia de *PositionPathInterpolator*. Como ya se comentó en el capítulo 3, en Java3D existen una serie de utilidades que permiten realizar movimientos en objetos indicando algunos puntos clave del movimiento, de forma que automáticamente se obtienen las posiciones intermedias. Y la clase *PositionPathInterpolator* utilizada necesita de dos estructuras de tipo *array* indicando los puntos origen y destino en las coordenadas definidas por el escenario virtual. Hay que destacar que, además del constructor vacío por defecto y del constructor con parámetros, se definen cinco métodos más. Así, los métodos *updateTranslation* y *updateTranslationWhenAdjusting* se encargan de actualizar los movimientos de los nodos, y en el caso del segundo además tiene en cuenta el tiempo pasado desde que se produjo el cambio de dirección del nodo, utilizado cuando se están ajustando las posiciones. Ambos métodos emplean como parámetros los puntos inicial y final del movimiento actual, la velocidad del movimiento, el ratio de velocidades de la animación respecto de la simulación de base, las dimensiones del escenario y, para el caso del segundo método, el tiempo que ha transcurrido desde que se produjo el cambio de dirección de nodo. El método *computeTime* calcula el tiempo que debe tardar un nodo en recorrer un movimiento rectilíneo dado el punto inicial, el punto final, la velocidad del propio movimiento y el ratio de velocidades de la animación respecto de la simulación de base. Finalmente, los métodos *transformCoordinatesRealToVirtual* y *transformCoordinatesVirtualToReal* son capaces de transformar las coordenadas de un punto del escenario real al escenario virtual y viceversa.

Información del escenario gráfico

Los atributos necesarios para crear y representar gráficamente el escenario con el uso de la API Java3D se presentan en la clase *Scenario*, representada en el diagrama de la figura 4.37. En ella se aprecia que, además de los atributos presentados en la parte de la representación gráfica de la clase a tal fin, se establecen otros tres atributos a través de relaciones de tipo asociación. El primero de ellos establece una relación de asociación hacia la clase *ApplicationInformation*, aunque a su vez ésta establece una relación de agregación con *Scenario*. Los otros dos atributos se establecen por relaciones con los controladores *ScenarioController* y *CameraBehaviourController*, que se encargan de

manejar la información para el escenario gráfico y para la posición de la cámara desde donde se observa el escenario gráfico, respectivamente. De los demás atributos, se puede decir que la gran mayoría son instancias de objetos propios de la API Java3D para representar la raíz del árbol del grafo de escena, el universo de escena, las líneas de los bordes del escenario, las líneas de las coordenadas, el instante actual de la animación mostrado en el escenario, etc.

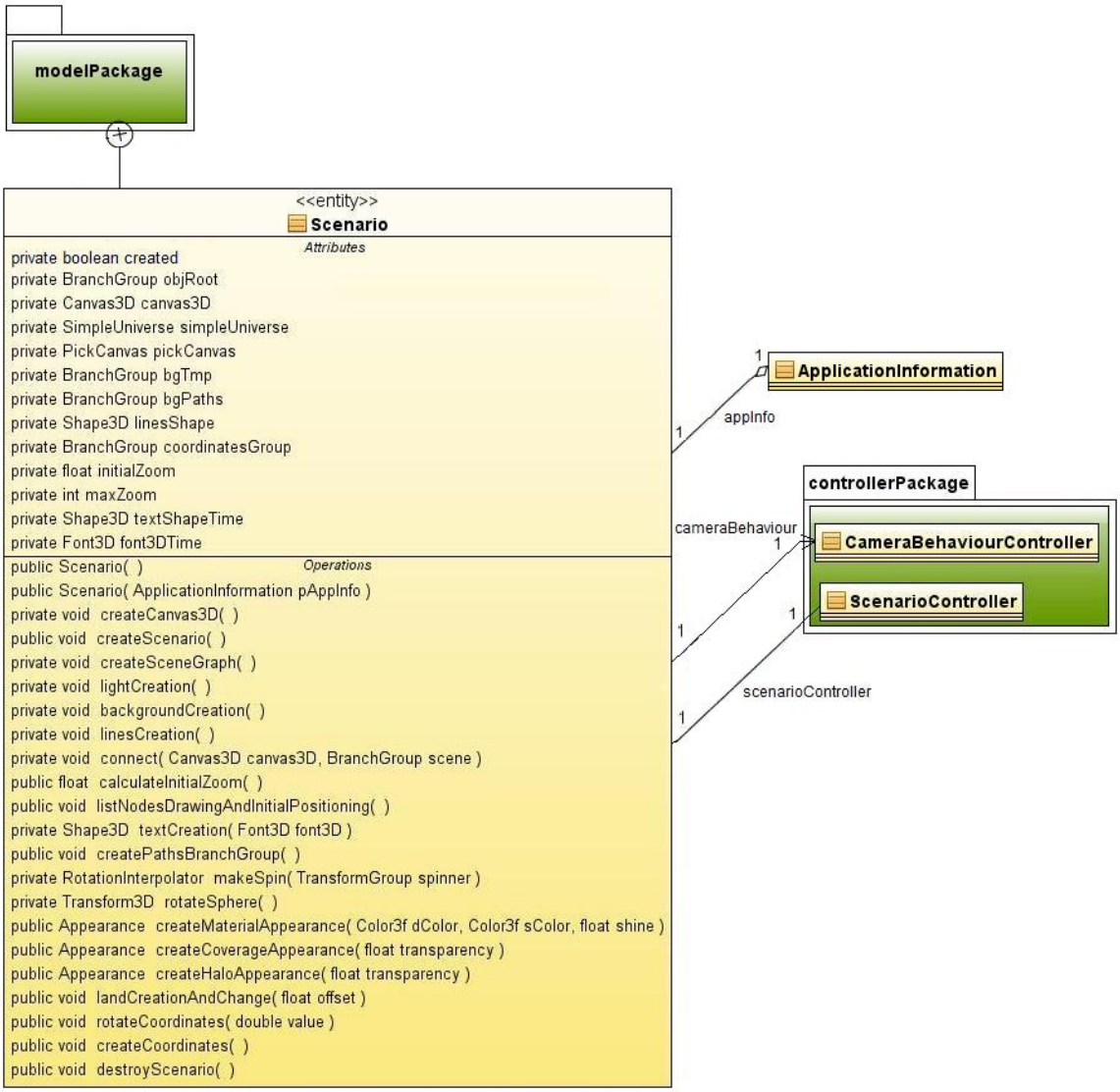


Figura 4.37. Diagrama de clases - Información del escenario gráfico

Los métodos de clase que aparecen son los siguientes: *createScenario*, que se encarga de la creación del escenario gráfico completo; *createCanvas3D*, donde se crea una instancia de la clase Canvas3D de Java3D con la configuración gráfica deseada; *createSceneGraph*, donde se crea el objeto raíz del grafo de escena; *lightCreation*, donde se añaden los objetos de luz al escenario, que son concretamente luz ambiental y luz direccional; *backgroundCreation*, donde se fija la imagen o textura de fondo del escenario;

linesCreation, donde se generan las líneas de borde del escenario atendiendo a las dimensiones cargadas desde el archivo de movilidad de los nodos; *connect*, que conecta el objeto raíz del grafo de escena con el universo virtual de forma que se pueda renderizar el escenario, y además crea la instancia de la clase *CameraBehaviourController*, que gestiona los puntos de vista a modo de cámara; *calculateInitialZoom*, en el cual se obtiene el *zoom* o distancia inicial al escenario donde se debe posicionar el punto de vista o la cámara desde donde se visualiza el escenario, teniendo en cuenta las dimensiones del mismo; *listNodesdrawingAndInitialPositioning*, en el cual se generan las esferas que representan los nodos, las coberturas y los halos, el texto que los identifica, los movimientos de rotación de las esferas y los movimientos de traslación que tendrán los nodos, y finalmente, posicionará los nodos en los puntos iniciales del escenario virtual dados por el archivo de movilidad de los nodos; *textCreation*, donde se obtiene el texto en 3D que mostrará el identificador del nodo; *createPathsBranchGroup*, donde se crea un objeto del árbol de escena que contendrá las animaciones para el envío de mensajes por la red; *makeSpin*, que genera una instancia de la clase de Java3D denominada *RotationInterpolator* que modifica la componente rotacional de la transformación de las esferas que aparecen en el escenario mediante interpolaciones; *rotateSphere*, donde se definen los planos de rotación de las esferas; *createMaterialAppearance*, *createCoverageAppearance* y *createHaloAppearance*, donde se define la apariencia, es decir, los colores, las luces y la textura de las esferas de los nodos, coberturas y halos, respectivamente; *landCreationAndChange*, donde se generan las líneas de separación que aparecen a modo de rejilla y donde se modifica el número de líneas presentadas en función del valor del *zoom*; *createCoordinates*, donde se crean los números que indican las coordenadas en metros; *rotateCoordinates*, donde se giran los números en 3D que muestran las coordenadas al rotar el punto de vista; y *destroyScenario*, donde se eliminan los objetos creados en el escenario y que son hijos del objeto raíz de árbol.

Información de la clase principal de la aplicación

La clase *ApplicationInformation*, como se ha comentado en numerosas ocasiones, contiene la información fundamental de la aplicación, como se observa en el diagrama de la figura 4.38. La instancia de esta clase, única instancia creada durante la ejecución de la aplicación, contiene como atributos a todos los controladores exceptuando a *ActionKeyMouseListenerController* (que es el controlador genérico asociado en primera instancia a los eventos de la ventana principal de la aplicación) y a *CameraBehaviourController* (que es el controlador del movimiento de la cámara asociado

al escenario); además, contiene a la instancia de la clase *Scenario*, a la instancia de la clase *ApplicationWindow*, a una colección de elementos de tipo *NodeInformation* con la información asociada a cada nodo del escenario y a dos colecciones de elementos de los tipos *NodeEventsMessaging* y *NodeEventsPositioning*, donde se almacenan los eventos cargados del archivo de salida de la simulación y del archivo de movilidad de los nodos, respectivamente. De esta forma, como la instancia de la clase *ApplicationInformation* se pasa a los constructores de todas las clases controladoras, éstas pueden acceder a los componentes del *Scenario*, a los componentes de la ventana principal de la aplicación y a otros controladores, haciendo uso de ella como puente y disminuyendo así el acoplamiento entre clases.

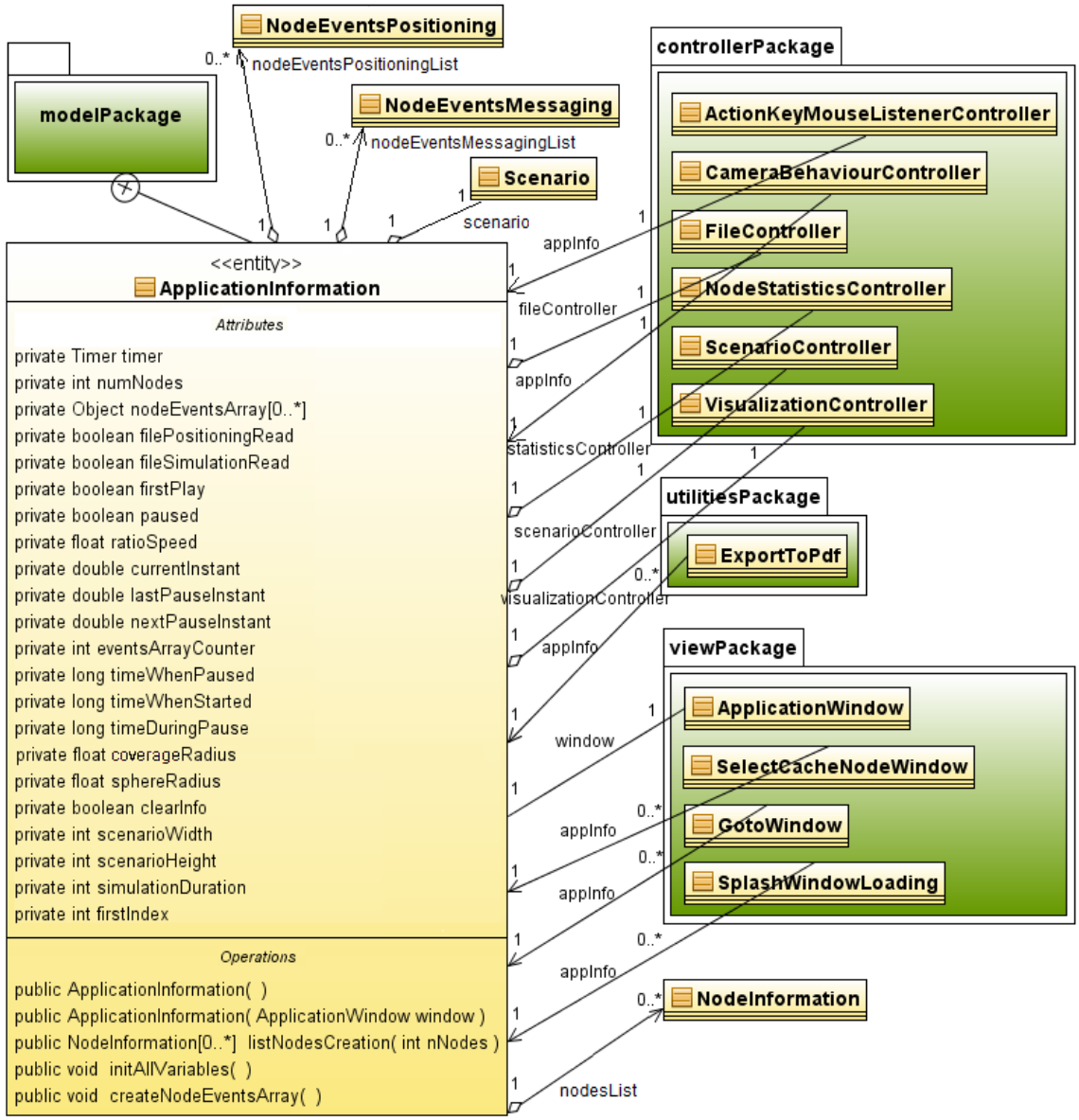


Figura 4.38. Diagrama de clases - Información de la clase principal de la aplicación

El resto de asociaciones unidireccionales que se aprecian en la figura indican que las clases con las que se relaciona contienen como atributo la instancia de esta clase *ApplicationInformation* para acceder a información de interés para su funcionamiento.

Los atributos que aparecen explícitamente en el diagrama en la zona correspondiente de la clase son: *timer*, el temporizador que controla la secuenciación de eventos durante la animación; *numNodes*, el número de nodos del escenario, cargado de los archivos de entrada de la aplicación; *nodeEventsArray*, una estructura de tipo *array* que contiene la mezcla de las dos colecciones con los eventos cargados desde los dos archivos de entrada de la aplicación, ordenados temporalmente y que se usa a lo largo de la visualización de la animación; *filePositioningRead* y *fileSimulationRead*, controlan si los archivos de entrada de la simulación se han cargado satisfactoriamente; *firstPlay*, indica si se ha producido la reanudación inicial de la animación, generando los movimientos iniciales de todos los nodos; *paused*, indica si la animación se encuentra pausada o no; *ratioSpee*d, almacena la relación entre las velocidades de la animación y de la simulación de base; *currentInstant*, instante actual de la animación; *lastPauseInstant*, contiene el instante en que se ha producido el último evento reproducido durante la animación, afectado por el factor del ratio de la velocidad que hubiera en el momento en que se ha actualizado el valor de este atributo; *nextPauseInstant*, contiene el instante en que se va a producir el siguiente evento a reproducir durante la animación, afectado por el factor del ratio de la velocidad que hubiera en el momento en que se ha actualizado el valor de este atributo; *eventsArrayCounter*, número que indica la posición en el *array* global de eventos del siguiente evento que se va a reproducir en la animación, pudiendo controlar que no se avance más allá de la longitud del *array* que contiene todos los eventos ordenados temporalmente; *timeWhenPaused*, donde se almacena el instante de tiempo en el que se produjo la pausa de la animación; *timeWhenStarted*, donde se almacena el instante de tiempo en el que ha generado el último evento que se ha reproducido en la animación; *timeDuringPause*, contiene el tiempo que ha permanecido pausada la animación entre dos eventos sucesivos, y es utilizado para restar correctamente este tiempo a la hora de ajustar el temporizador para realizar la reanudación de la animación, pudiéndose dar el caso de que se pause la animación más de una vez entre los dos eventos sucesivos; *sphereRadius* y *coverageRadius*, donde se almacenan los radios de las esferas de los nodos y de sus coberturas, respectivamente; *clearInfo*, se utiliza para evitar que se realicen algunas actividades cuando se ha dejado la aplicación como recién cargada; *scenarioWidth* y *scenarioHeight*, con las dimensiones del escenario cargadas del archivo de movilidad de

los nodos; y *simulationDuration*, que contiene la duración de la simulación cargada de los ficheros de entrada de la aplicación.

Además de los constructores, los demás métodos que se han definido en esta clase son: *listNodesCreation*, donde se inicializa el listado de objetos que contendrá la información acerca de los nodos; *initAllVariables*, donde se inicializan todos los atributos de la clase para que se pueda generar un nuevo escenario a partir de la carga de nuevos ficheros de entrada; y *createNodeEventsArray*, donde se produce la mezcla de los eventos cargados desde el archivo de salida de la simulación y desde el archivo de movilidad de los nodos, realizando su ordenación temporal y llamando al controlador de estadísticos para calcular la evolución temporal de los valores estadísticos de cada nodo.

4.5.2.3.3. Paquete de la Vista: *View Package*

La vista de la aplicación contiene aquellas clases que forman parte de la interfaz con el usuario, tanto aquellas ventanas por donde el usuario interactúa como las ventanas de información utilizadas en determinados momentos de la aplicación, que se irán especificando a continuación. Las clases que forman parte de la vista de la aplicación, atendiendo al MVC, se han agrupado en el paquete denominado *viewPackage*, cuyo contenido se detalla en la figura 4.39.

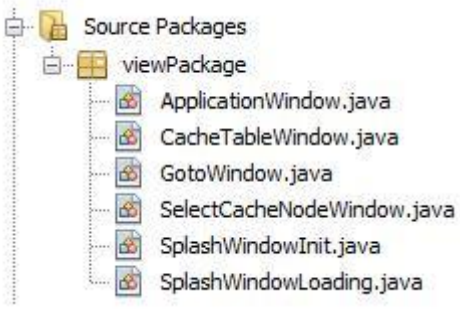


Figura 4.39. Contenido del paquete de la Vista: *View Package*

Se van a ir explicando los diagramas de clases generados para aquellas clases que forman parte del paquete de la vista, como se ha realizado anteriormente para otros paquetes.

Información de las ventanas de aviso de estado

Las ventanas emergentes de aviso de estado o de información de estado indican algún aspecto del estado de la aplicación. Las dos clases que aparecen en el diagrama de la figura 4.40 heredan de la clase *JWindow* de Java, que a su vez se encarga de generar ventanas en

cualquier parte del escritorio del usuario y se caracterizan por ser ventanas sin barra de título ni botones de gestión. Estas ventanas que únicamente poseen alguna imagen de fondo y texto para dar algún tipo de información y que se crean y destruyen desde el propio software las denominaremos ventanas *splash* o *Splash Windows*.

La clase *SplashWindowInit* genera la ventana que se muestra al arrancar la aplicación con el logo de la misma durante unos instantes. Esta ventana es creada como atributo de la clase inicial de la aplicación, *Main*, como se aprecia en el diagrama. Además de los métodos constructores, se ha definido un nuevo método denominado *resizeImageIcon* para redimensionar el tamaño de la imagen de fondo que se coloca en dicha ventana.

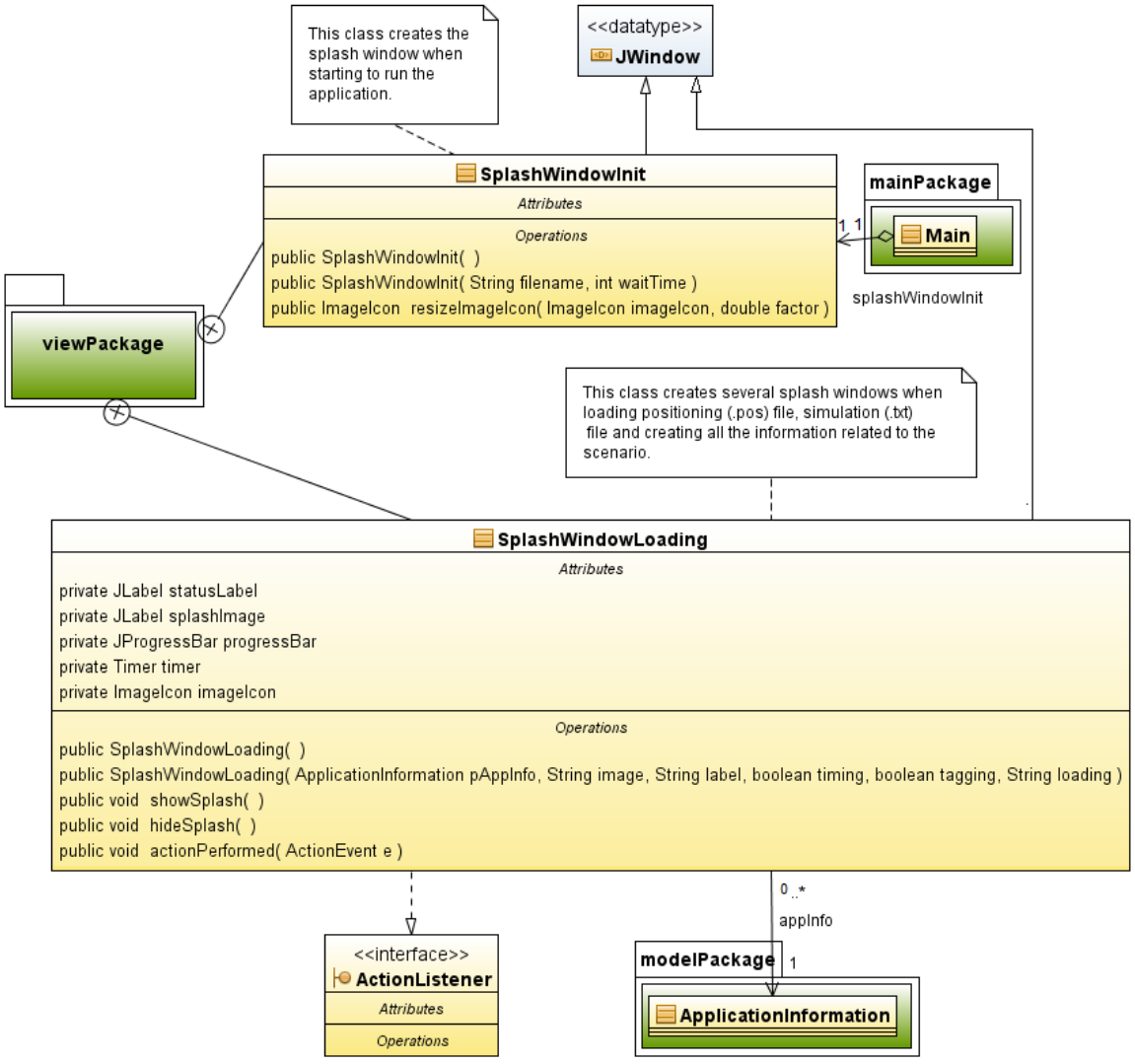


Figura 4.40. Diagrama de clases - Información de las ventanas de aviso de estado

Por otra parte, la clase *SplashWindowLoading* tiene un propósito diferente, puesto que genera ventanas indicando si se está cargando la información del archivo de salida de la simulación o del archivo de movilidad de los nodos, o bien si se está generando el

escenario y mezclando los eventos de ambos archivos. En este último caso, se utiliza además la barra de progreso que aparece como atributo para indicar los últimos instantes de la carga del escenario. Como se observa en la figura, esta clase implementa la interfaz *ActionListener* de Java, de forma que se introduce un temporizador que cuando agota su retardo, se produce la llamada al método *actionPerformed* donde se avanza la barra de progreso para el caso de la ventana que lo contiene. La relación de asociación que establece con la clase *ApplicationInformation* es debido a que necesita un atributo de ella, en concreto requiere de la ventana principal de la aplicación para indicar cuál es la ventana primaria y fijar que las ventanas del tipo *SplashWindowLoading* sean secundarias y se muestren siempre encima de ella. Además de los constructores de clase, los otros dos métodos que aparecen, *hideSplash* y *showSplash*, se utilizan para cerrar la ventana creada y abrir una nueva, respectivamente.

Ventana principal de la aplicación

La ventana principal de la aplicación se crea tras el arranque de la aplicación, y contiene todos los controles necesarios para llevar a cabo las funciones planteadas a través de los requisitos definidos durante la fase de análisis. En dicha ventana se establece una barra de menús a través de la cual se pueden acceder a la mayoría de las funciones implementadas a través de los controles que también aparecen en la propia ventana. Además, en ella se inserta la zona dedicada para el renderizado de la animación.

En el diagrama de la figura 4.41 se observa cómo esta clase *ApplicationWindow* es hija de la clase *JFrame* de Java, y se pueden apreciar como sus atributos son los componentes añadidos a la ventana en forma de controles, todos ellos componentes propios de los paquetes de Java *javax.swing* y *java.awt*. Como ya se mencionó al explicar la clase *ApplicationInformation*, ambas se encuentran relacionadas a través de una asociación bidireccional, de forma que una es atributo de la otra, y viceversa. Además, esta clase es la encargada de crear la instancia del controlador genérico como atributo propio, *ActionKeyMouseListenerController*, donde se gestionarán los eventos que se produzcan en los componentes de la ventana, aunque su cometido sea el de redirigir hacia el controlador específico la función a realizar. También se establecen relaciones de tipo asociación con algunas clases del paquete de utilidades: con *InitialTimerListener*, puesto que desde la instancia de esta clase se crea la ventana principal, siendo un atributo de la misma; *JTextFieldLimit*, donde se limita el número de cifras a introducir en las cajas de texto presentes en la ventana, en función del valor de la duración de la simulación; y *BackgroundPanel*, donde se crea el panel derecho de la aplicación con la imagen de fondo

presentada. Finalmente, también se establece una relación de uso con *NodeInformation* para obtener un listado con los nombres de los nodos.

El constructor de la clase genera las instancias de las clases *ApplicationInformation* y *ActionKeyMouseListenerController*, añadiendo el icono de la aplicación y el tamaño de la ventana. El resto de métodos de la clase son: *createAndPlaceComponents*, donde se crean y posicionan los componentes en la ventana, los cuales van a aparecer inicialmente deshabilitados hasta que el escenario se haya creado correctamente (realmente no tiene sentido que se encuentren habilitados cuando no se puede realizar la visualización de ninguna animación al no haber un escenario virtual); *makeMenuItem* y *makeSubMenu*, que devuelven instancias de elementos de la barra de menús con la cadena de texto y la imagen (dada su ruta) que son pasados como parámetros; *createButton*, *createNorthPanelButton* y *createArrowButton* devuelven elementos de tipo *JButton*, es decir, los botones de Java, que tendrán en cada caso diferente tamaño y con una imagen de fondo que se encuentra en la ruta pasada como parámetro; *lightButtons*, donde cambia la imagen de fondo de algunos de los botones de la ventana, pasando de ser imágenes con tonos grisáceos a imágenes con colores que se hacen visibles cuando se habilitan dichos botones; *enableComponents* y *disableComponents*, donde se habilitan y deshabilitan, respectivamente, los componentes de la ventana cuando se activa o desactiva un escenario dado; *changeSliderTime*, donde se modifica el extremo con el valor que puede tomar la barra horizontal o *slider* temporal asociado a la duración de la simulación, además de mostrar algunos de sus valores intermedios; *loadCheckBoxes*, método en el cual se carga en los listados destinados a la selección de nodos y a la muestra de las coberturas de nodos cada uno de los nodos que existen en el escenario, generando para cada una de las entradas un elemento de tipo *JCheckBox* de Java y añadiendo además una primera entrada para la selección o deselección de todo el listado de una vez; *addListenersToComponents*, donde se asocia a cada componente la instancia del controlador genérico atendiendo al tipo de eventos que se pueden producir en un componente; *startComponents*, donde se reinician todos los componentes con los valores que deben tener por defecto cuando se procede a una nueva carga de ficheros; *clearAll*, que se encarga de reiniciar los datos de la aplicación cuando se presiona el botón destinado a tal fin de la ventana. Como ocurre en las demás clases presentadas, no se han añadido los métodos denominados *getters* y *setters* para la accesibilidad de los componentes con el propósito de reducir el tamaño de los diagramas.

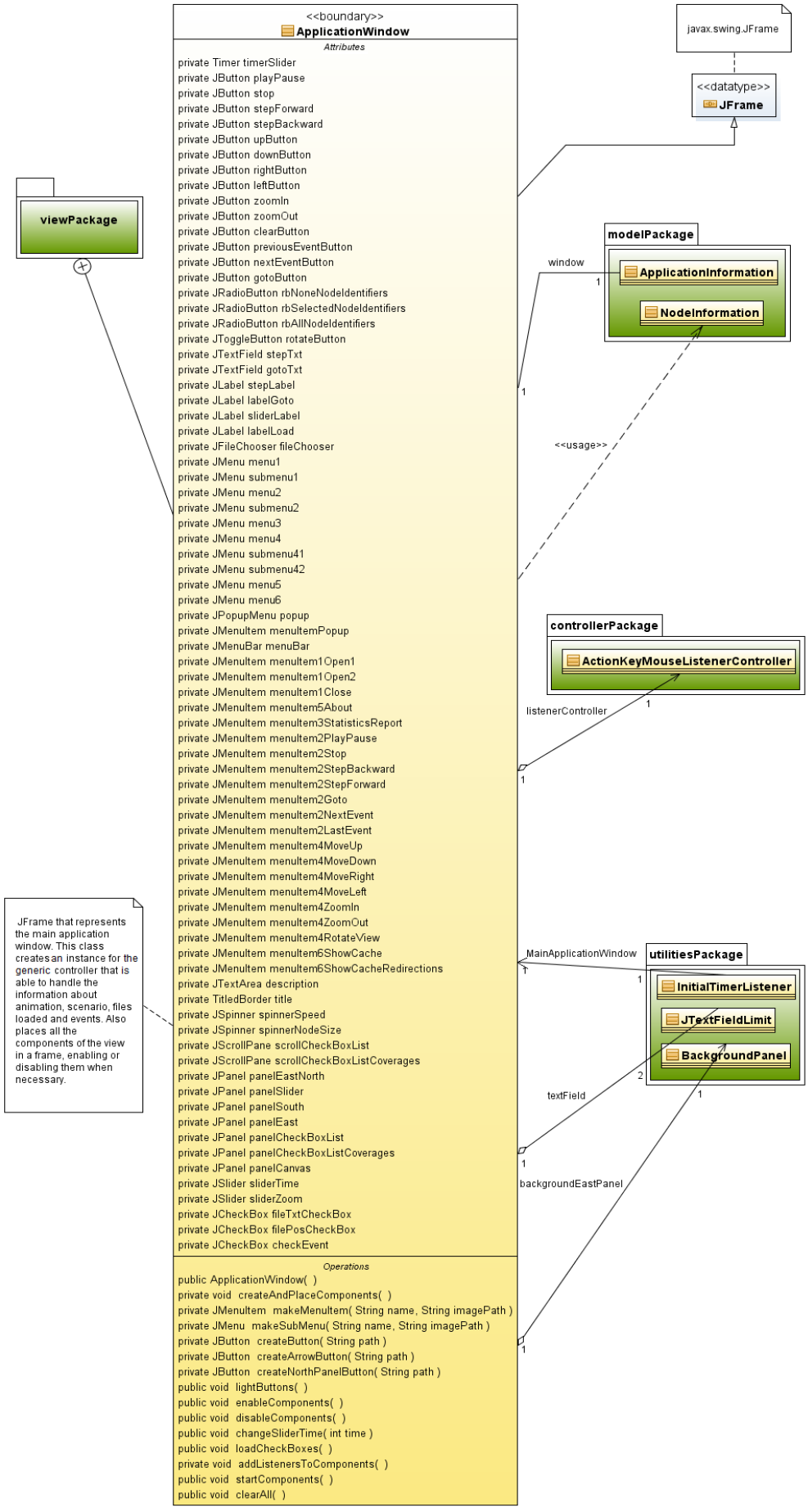


Figura 4.41. Diagrama de clases - Ventana principal de la aplicación

Ventanas auxiliares

El diagrama de clases que contiene la última parte de la vista de la aplicación atendiendo al MVC se presenta en la figura 4.42.

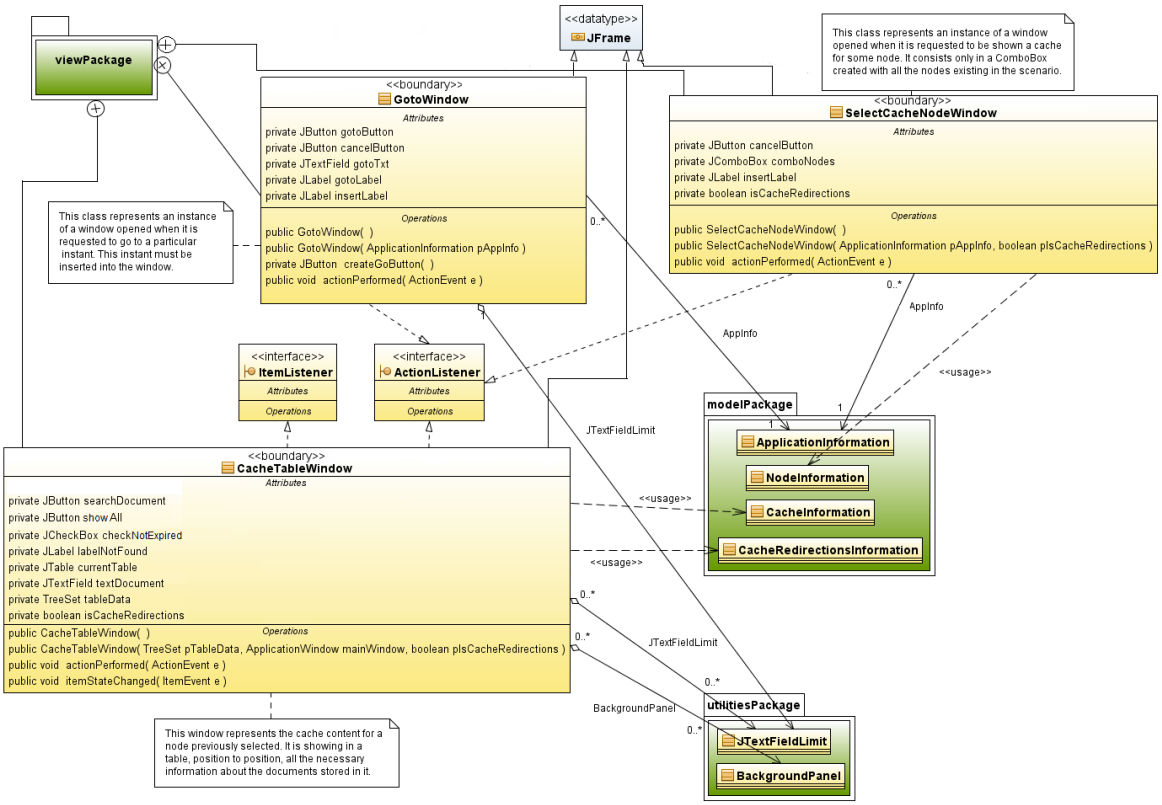


Figura 4.42. Diagrama de clases - Ventanas auxiliares

Las tres clases que aparecen en el diagrama representan las ventanas "*Go to*", las ventanas de selección de nodos para la muestra de la caché local y de la caché de redirecciones, y por último las ventanas con los resultados de las consultas realizadas de la caché local y de la caché de redirecciones**.** Estas clases son directamente clases hijas de la clase *JFrame* de Java.

La clase *GotoWindow* representa la ventana que se crea al acceder a través del menú de la aplicación cuando se desea avanzar o retroceder en el tiempo hacia un instante absoluto temporal. Este instante temporal se debe insertar en una caja de texto que aparece en esta ventana. Además, implementa la interfaz de Java *ActionListener* debido a que contiene el código (a través del método *actionPerformed*) que gestiona el evento de pulsar el botón correspondiente para realizar el desplazamiento temporal. Además de los atributos que aparecen explícitamente en la zona de la clase a tal fin, gracias a la relación de asociación unidireccional posee como atributo a la instancia de la clase *ApplicationInformation* para

poder recuperar la información de la duración de la simulación. Además, necesita de la clase *JTextFieldLimit* para limitar el número de cifras de la caja de texto donde se debe introducir el instante de tiempo en función de las cifras de la duración de la simulación.

La clase *SelectCacheNodeWindow* representa una ventana que muestra un listado con todos los nodos de la aplicación, en forma de menú desplegable, con el propósito de elegir para qué nodo se desea visualizar el contenido de la caché local o de la caché de redirecciones. Como la clase presentada anteriormente, implementa la interfaz de Java *ActionListener* debido a que contiene el método *actionPerformed* que gestiona el evento de elegir a un nodo y que crea, a su vez, una instancia de la clase *CacheTableWindow*. Además de los atributos presentes en la propia clase, posee como atributo a la instancia de la clase *ApplicationInformation* para poder acceder al instante actual de la animación y al listado de los nodos de la aplicación y mostrar así la identificación de cada nodo. Por este mismo motivo se establece la relación de uso con la clase *NodeInformation*.

Finalmente, la clase *CacheTableWindow* es la ventana encargada de mostrar el contenido de la caché local o de la caché de redirecciones para el nodo seleccionado, ya sea a través de la ventana de selección de nodos que se acaba de presentar o bien a través del menú contextual asociado a la esfera de cada nodo en el escenario gráfico. Muestra el contenido de la caché, posición por posición, en una tabla cuyas columnas dependerán del tipo de caché solicitada. Además, presenta una serie de controles para realizar la búsqueda de un determinado documento por su identificador y para hacer un filtrado de las entradas mostradas atendiendo al instante de expiración. Para lograr estas funcionalidades, implementa las interfaces de Java *ActionListener* (añadiendo el método *actionPerformed*) e *ItemListener* (añadiendo el método *ItemStateChanged*). Para mostrar el contenido de las cachés, necesita utilizar las clases *CacheInformation* y *CacheRedirectionsInformation*. Como ocurría con la ventana principal de la aplicación, esta clase posee como atributos a instancias de las clases *JTextFieldLimit*, para limitar el número de cifras a introducir en una caja de texto, y *BackgroundPanel*, donde se establece una imagen de fondo para la tabla mostrada.

4.5.2.3.4. Paquete del Controlador: *Controller Package*

Las clases controladoras poseen las acciones que se llevan a cabo debidas a eventos generados en la aplicación, organizadas atendiendo a diferentes grupos de funcionalidades. En este paquete se encuentran las seis clases controladoras que se observan en la figura 4.43.

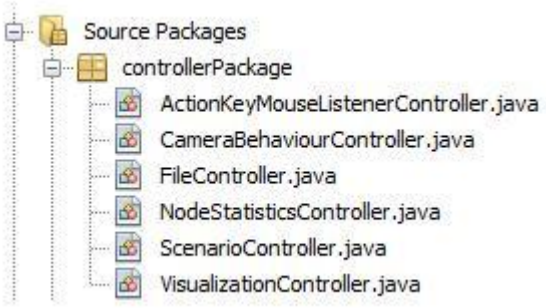


Figura 4.43. Contenido del paquete del Controlador: *Controller Package*

Controlador genérico

La clase de tipo controladora que se ha estado clasificando como genérica se presenta en el diagrama de clases de la figura 4.44 y se denomina *ActionKeyMouseListenerController*, debido al tipo de interfaces que implementa. Como ya se ha comentado en varias ocasiones, este controlador se encarga de detectar todos los eventos que se producen en la ventana principal de la aplicación, dirigiendo el control del programa hacia el controlador más adecuado de entre los que se verán a continuación. Estos controladores sí que contendrán el código necesario para responder según el evento que se haya producido.

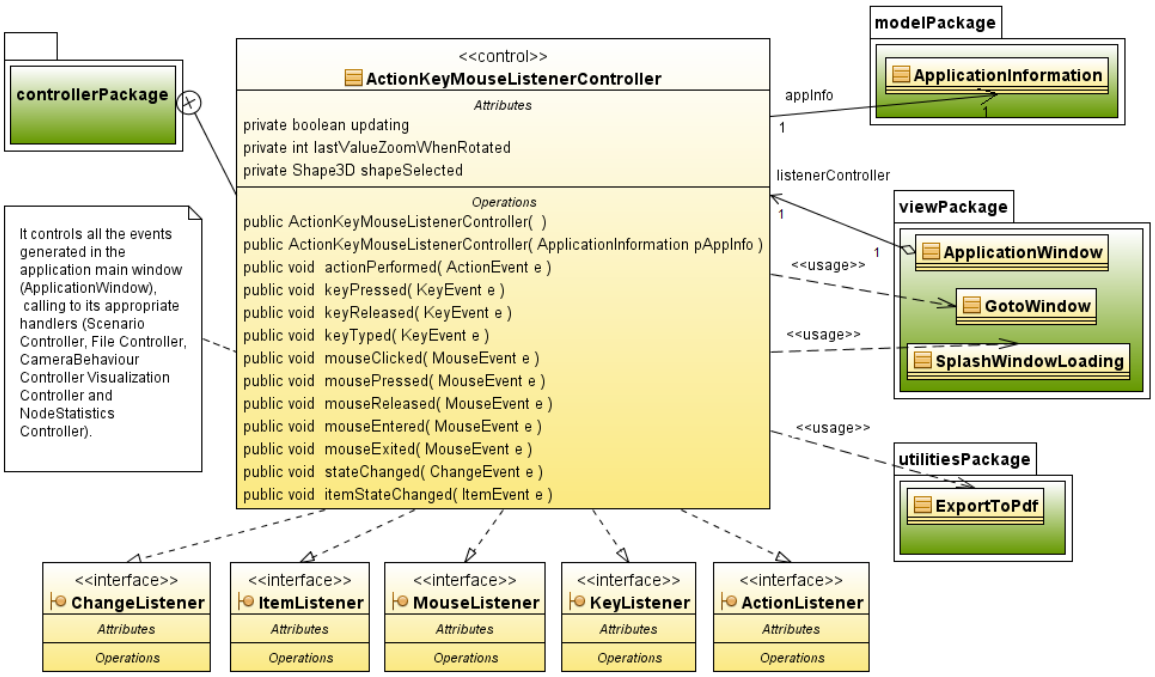


Figura 4.44. Diagrama de clases – Controlador genérico: *ActionKeyMouseListenerController*

Para poder detectar cada uno de los diferentes eventos de la ventana, debe implementar todos los interfaces de Java que aparecen en la figura. Cada uno de esos interfaces obliga a

implementar una serie de métodos, de forma que se controlen las acciones llevadas a cabo por el usuario a través del ratón y el teclado, además de los eventos producidos por los temporizadores utilizados. Además de los atributos de la clase que aparecen de forma explícita, esta clase controladora posee la instancia de *ApplicationInformation* para poder acceder a los demás controladores y dirigir cada evento hacia su función asociada. En el diagrama además se observa como la instancia creada de esta clase en la aplicación es un atributo de *ApplicationWindow*, como se comentó anteriormente. Finalmente, se establecen además relaciones de uso con otras clases de los paquetes *viewPackage* y *utilitiesPackage*.

Controlador del movimiento de la cámara de visualización

Uno de los controladores específicos que se han creado para esta aplicación se encarga de modificar los movimientos que sigue la cámara que está mostrando continuamente el contenido del escenario gráfico. Así, para poder moverse por el escenario, e incluso para rotar el punto de vista del mismo, se accede a este controlador y sus métodos. El diagrama de clases de este controlador denominado *CameraBehaviourController* se observa en la figura 4.45. Es importante destacar que esta clase extiende, es decir, es hija de la clase *Behavior* de Java3D que es la clase base que define la gran cantidad de comportamientos que se pueden encontrar en esta API, como las interpolaciones de movimientos, de rotaciones, de atributos de color en objetos del escenario, etc. En el diagrama además se observa que la instancia de esta clase es un atributo de la clase *Scenario*, ya que se crea desde esta propia clase, además que establece una relación de uso con la clase *NodeInformation* para tomar la información de las transformaciones que van siguiendo los nodos.

Además del atributo que se establece a través de una relación de asociación con la clase *ApplicationInformation*, el resto de atributos están relacionados directamente con la visualización del escenario. Así, los objetos *TransformGroup*, *Transform3D* y *RotPosPathInterpolator* establecen el movimiento que sigue la cámara gracias a la interpolación de puntos intermedios en los movimientos dados un punto inicial y final y un ángulo inicial y final. Por otra parte se obtiene el criterio por el cual se aplica este comportamiento de movimiento de la cámara, denominado *WakeupCondition*, criterio creado como suma de determinados eventos producidos por el ratón o por el teclado. El resto de atributos indican si el punto de vista está rotado o no, el *zoom* inicial que se ha estimado que tenga la cámara al cargar el escenario, y los saltos mínimos en los movimientos de la cámara en todas las posibles direcciones. Además de los constructores de clase, para simular el comportamiento de la cámara se han creado los siguientes

métodos: *cameraInitialPosition* y *cameraInitialPositionWhenRotated*, donde se posiciona la cámara en la posición inicial con el *zoom* estimado para que quede el escenario completamente dentro de la zona de renderizado, tanto con el punto de vista normal como rotado; *initialize*, método necesariamente sobreescrito de la clase *Behavior* para indicar cuál es la nueva condición para que se detecte el cambio de comportamiento de la cámara; *processStimulus*, nuevamente método sobreescrito de la clase *Behavior*, que junto al método *processAWTEvent* se le indica a la cámara del escenario qué debe hacer cuando se produce alguno de los eventos definidos en la condición de activación del comportamiento; *processZoomInOut* y *processUpDownRightLeftKey*, métodos que realizan el cambio en el punto de vista de la cámara una vez que se ha detectado si el tipo de evento producido implica un cambio en el *zoom* o si se debe generar un desplazamiento hacia arriba, abajo, derecha, izquierda o rotación de la cámara, respectivamente; y *paintCoordinates*, donde se cambian las líneas de fondo del escenario a modo de rejilla en función de valor del *zoom* que se posea cuando la vista del escenario no se encuentre rotada.

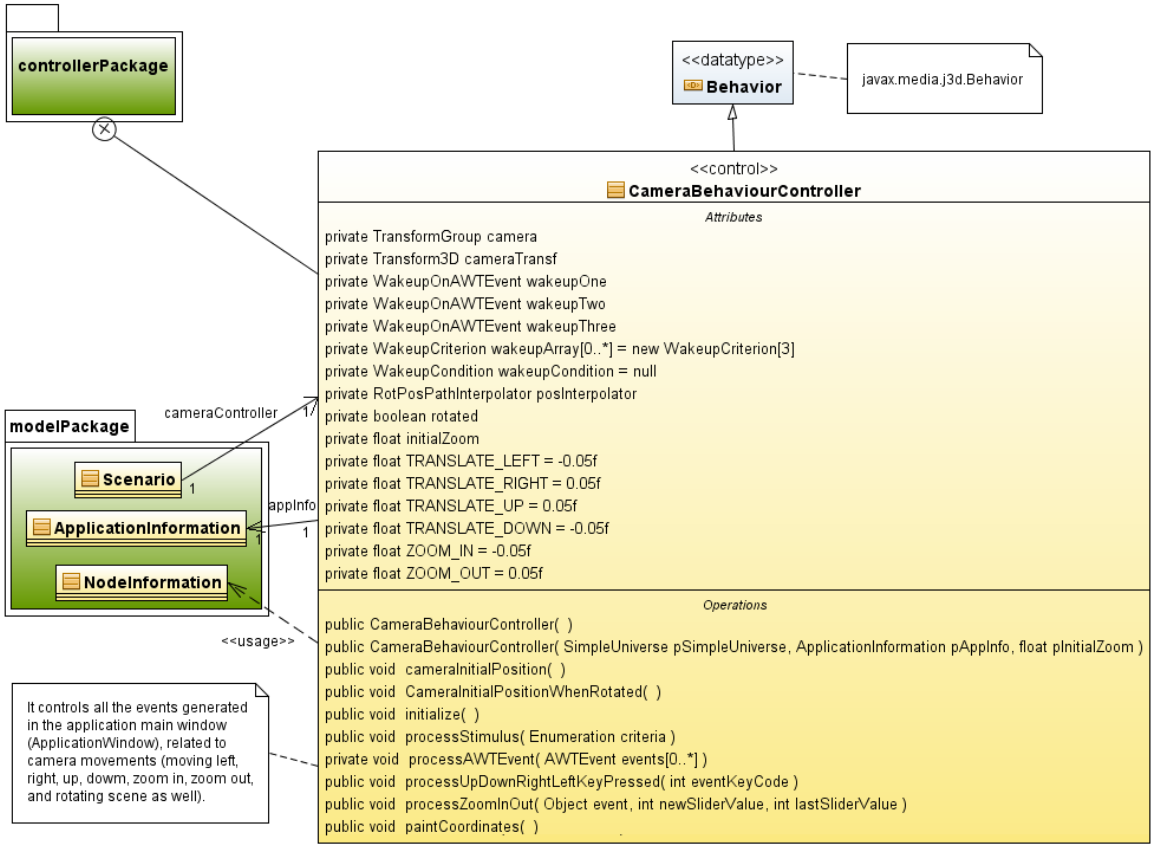


Figura 4.45. Diagrama de clases – Controlador del movimiento de la cámara de visualización: *CameraBehaviourController*

Controlador de la carga de datos de archivos

En la figura 4.46 se presenta la clase *FileController*, que es el controlador encargado de la apertura y cierre de ficheros y de la carga de los datos de los mismos.

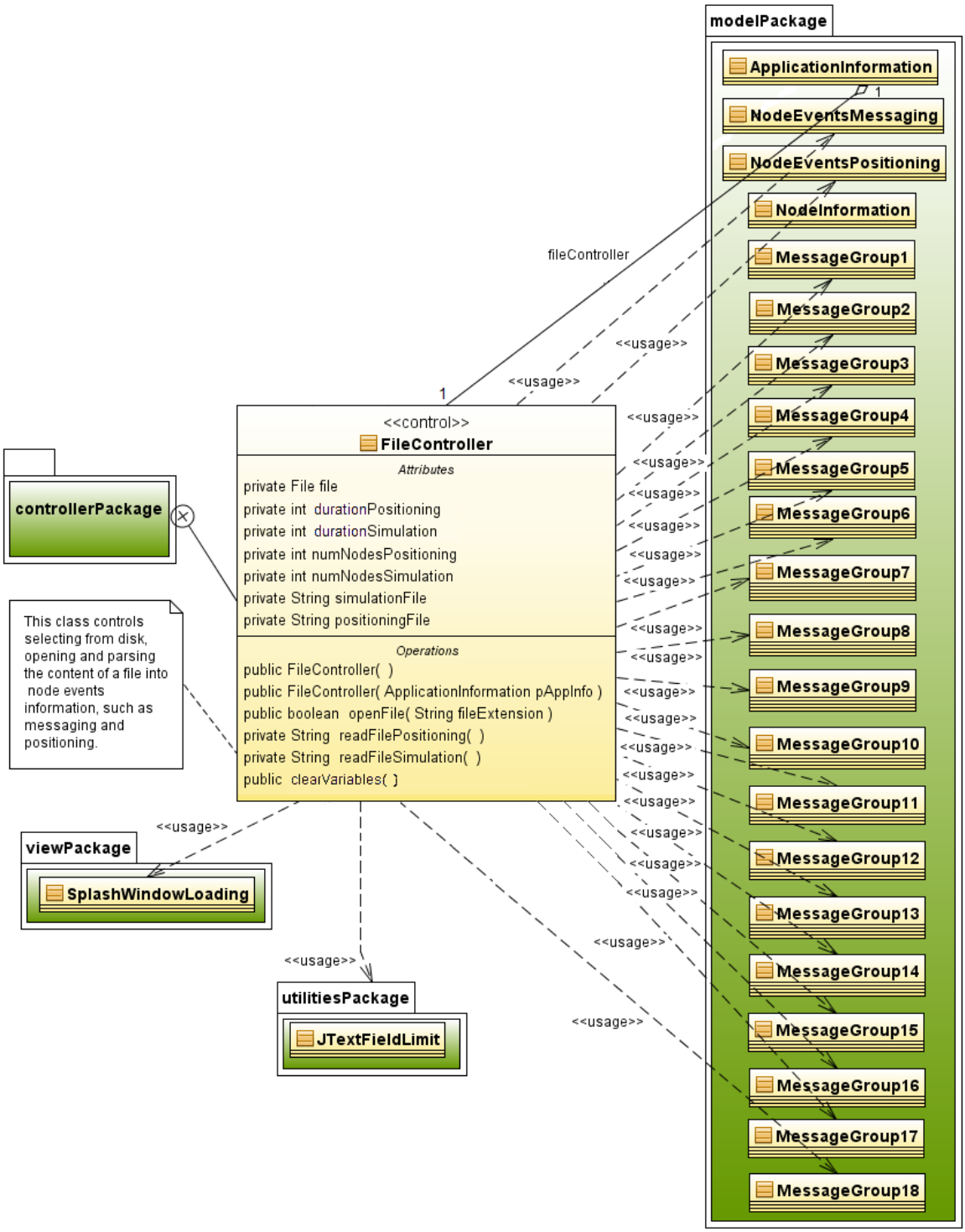


Figura 4.46. Diagrama de clases – Controlador de la carga de datos de archivos: *FileController*

Cuando se verifica que se ha cargado correctamente el archivo de movilidad de los nodos o bien que se han cargado ambos archivos si se ha empezado por el archivo de salida de la simulación, y además se cumple la integridad de los datos, entonces este controlador pasará a invocar los métodos para cargar los elementos del escenario, mezclar los eventos provenientes de los ficheros de entrada de la aplicación, cargar los listados de nodos para la selección de nodos y coberturas, y habilitar los controles de la ventana principal de la aplicación.

En el diagrama se observa como la instancia creada de esta clase será un atributo de la instancia de *ApplicationInformation*. Además, establece relaciones de uso con todas las clases que definen los diferentes tipos de eventos de los nodos, ya sean de mensajes o de cambios de dirección, y con las clases *JTextFieldLimit*, para limitar el número de cifras de las cajas de texto de la aplicación gracias al dato obtenido acerca de la duración de la simulación, y *SplashWindowLoading*, ya que genera ventanas de información indicando si se está cargando alguno de los ficheros o si se está creando el escenario.

Los atributos que presenta esta clase albergan los datos de la duración de la simulación, el número de nodos y el nombre completo del archivo (para ambos archivos de salida de la simulación y de movilidad de los nodos). Como métodos, además de los constructores por defecto y con parámetros, se presentan: *openFile*, que se encarga de las acciones llevadas a cabo en el diagrama de actividades de la figura 4.11; *readFilePositioning* y *readFileMessaging*, que se encargan de leer línea por línea el contenido de los ficheros de movilidad de los nodos y de salida de la simulación respectivamente, creando los objetos necesarios para almacenar toda esa información; y *clearVariables*, que reinicia el valor de los atributos para comenzar con la carga de nuevos ficheros.

Controlador para cálculo de estadísticos

Las acciones llevadas a cabo para la obtención de los estadísticos de un nodo se engloban en la clase controladora *NodeStatisticsController*, mostrada en la figura 4.47.

En el diagrama de observan todas las relaciones de uso que se establecen con las clases que definen los mensajes intercambiados por la red, debido a que algunos de esos mensajes son los que producen alteraciones en los valores estadísticos calculados para un determinado nodo. Además de ellas, existen otras relaciones de uso a destacar: con la clase *NodeInformation*, ya que ésta contiene como atributo una colección de tipo *Vector* con la evolución temporal de los valores estadísticos de ese nodo; con la clase *NodeEventsMessaging*, clase padre de todos los tipos de mensajes cargados desde el

archivo de salida de la simulación; y con las clases *NodeAffectedInformation* y *NodesAffectedInformation*, puesto que utiliza colecciones de elementos de este tipo para determinar los nodos que son intermedios en los caminos seguidos por las peticiones realizadas por un nodo dado, acción que se lleva a cabo durante el cálculo de los estadísticos que se produce tras la carga inicial del archivo de salida de la simulación.

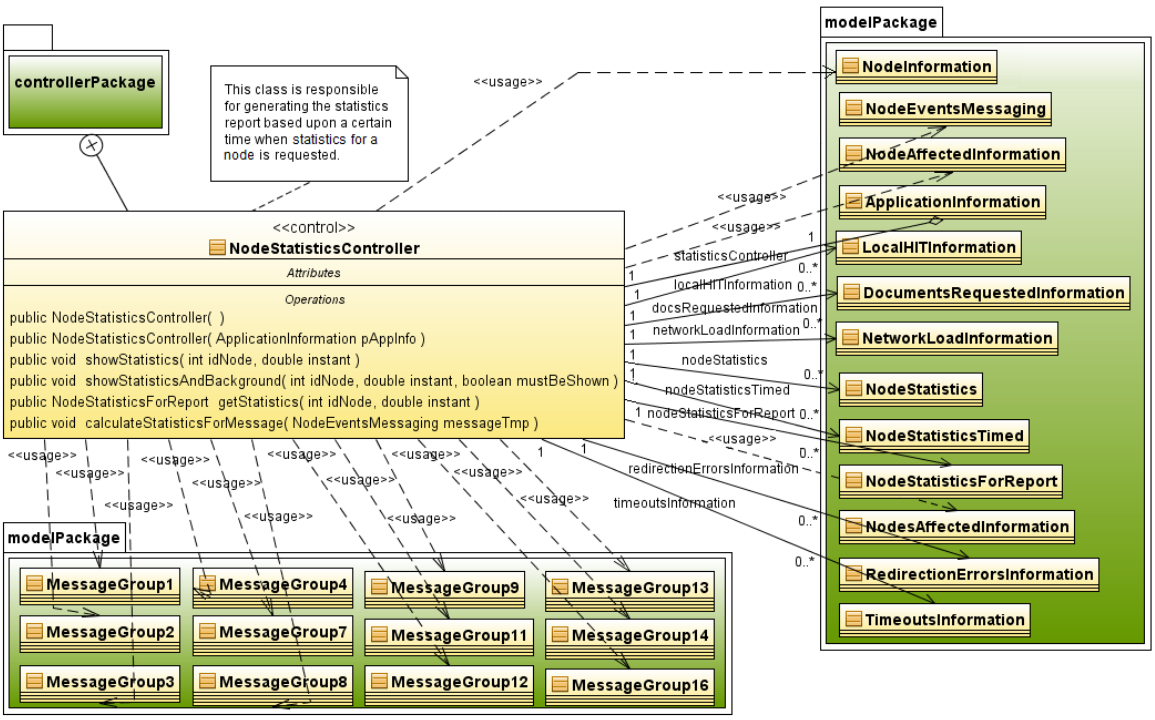


Figura 4.47. Diagrama de clases – Controlador para cálculo de estadísticos: *NodeStatisticsController*

Los atributos que se establecen para esta clase se explican a través de relaciones de asociación unidireccionales con otras clases, entre los que se tienen: *NodeStatistics*, *NodeStatisticsTimed* y *NodeStatisticsForReport* donde se almacena la información de estadísticos calculada en un determinado instante de tiempo y para un cierto nodo; y *LocalHITInformation*, *DocumentsRequestedInformation*, *NetworkLoadInformation*, *RedirectionErrorsInformation* y *TimeoutsInformation*, puesto que definen estadísticos que se actualizan de forma diferente al resto de estadísticos debido a decisiones de diseño. Finalmente, se observa que la clase *ApplicationInformation* contiene un atributo que es instancia de esta clase, como ocurre con la mayor parte de los controladores.

Controlador del escenario gráfico

La clase controladora que se va a presentar a continuación es la responsable de controlar y gestionar los eventos relacionados con el escenario gráfico y con los cambios

de apariencia que se deban realizar en el mismo. Este controlador no tiene nada que ver con la temporización, ya que se encargará de los aspectos gráficos relacionados con los nodos, las coberturas, los halos, las animaciones relativas a peticiones de documentos, las animaciones relativas al intercambio de mensajes por la red, los aspectos gráficos relativos a la pausa y a la reanudación del movimiento de los nodos, etc. El diagrama de clases de este controlador se presenta en la figura 4.48. En él se observa que los atributos definidos van a estar relacionados en su gran mayoría con aspectos gráficos del escenario, como las características de apariencias, materiales, colores, comportamientos interpoladores de colores y de movimientos propios de la API Java3D, pero sin olvidarse de otros atributos definidos para albergar valores temporales con el propósito de ahorrar el número de variables definidas en tiempo de ejecución, como se ha hecho para otras muchas clases.

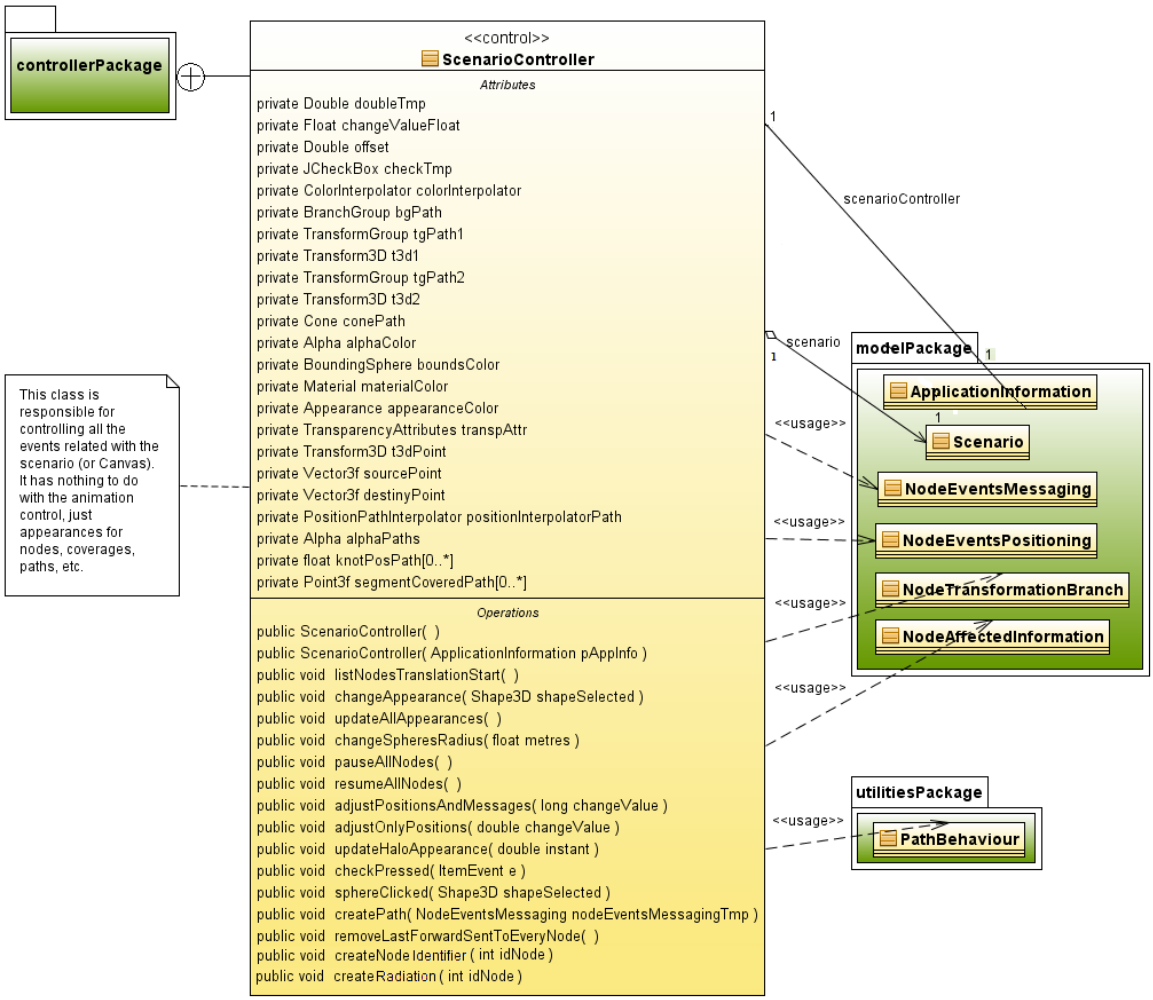


Figura 4.48. Diagrama de clases – Controlador del escenario gráfico: *ScenarioController*

Respecto a las relaciones de uso que se establecen con otras clases, se observan las siguientes: con las clases *NodeEventsMessaging* y *NodeEventsPositioning*, que son

importantes para recalcular la posición que deben tener los nodos tras un salto temporal o por eventos, función de suma importancia para la aplicación; con *NodeTransformationBranch*, donde se almacena la información de interés para la representación gráfica relacionada con un nodo, como los objetos de tipo esfera asociados a los nodos, sus coberturas y sus halos, las transformaciones propias de Java3D para los movimientos de rotación y traslación y los objetos de tipo texto en 3D para mostrar el identificador de los nodos; con la clase *NodeAffectedInformation*, ya que necesita detectar en cada instante de tiempo los nodos que se encuentran afectados por otro nodo que esté seleccionado para poder mostrar su halo; y con *PathBehaviour*, para la creación de las animaciones durante una cantidad de tiempo que representan el envío de mensajes por la red y la radiación debida a una petición de un documento. Además, se establece una relación de asociación con la clase *ApplicationInformation* y otra de tipo agregación con la clase *Scenario*, puesto que este controlador es un atributo de la misma.

Los métodos que se han implementado y que son de interés para este estudio son: *listNodesTranslationStart*, donde se establecen los primeros movimientos de los nodos por el escenario en función de los datos cargados desde el archivo de movilidad de los nodos; *changeAppearance*, que cambia la apariencia de un nodo del escenario, pasando como parámetro su identificador, cuando se produce un evento que pueda afectar a la misma; *updateAllAppearances*, que actualiza la apariencia de aquellos nodos de la red que puedan verse afectados a la vez por un determinado evento; *changeSphereRadius*, que cambia el radio de la esfera que representa a un nodo y, consecuentemente, de su halo si se estuviese mostrando en ese momento; *resumeAllNodes* y *pauseAllNode*s, que activa o pausa, respectivamente, los movimientos de todos los nodos del escenario a través de las características de los elementos interpoladores de Java3D utilizados; *adjustPositionsAndMessages* y *adjustOnlyPositions*, que se encargan de actualizar la posición en el escenario de los nodos cuando se producen saltos temporales en la animación, con la diferencia de que el primero además tiene en cuenta aspectos de la actualización de los eventos próximos a ejecutar, que aunque en principio tengan más relación con el controlador de la visualización, en este caso particular ha sido necesario englobar las dos funcionalidades en un mismo método para reducir el tiempo de ejecución, crítico en algunas ocasiones; *updateHaloAppearance*, donde se actualiza el aspecto de los halos en un determinado instante de tiempo; *checkPressed*, método que selecciona o desmarca un nodo o una cobertura cuando se pulsa sobre alguno de los elementos de tipo *jCheckBox* de Java que aparecen en los listados para nodos y coberturas; *sphereClicked*, donde se selecciona un nodo que ha sido clicado; *createPath*, que maneja los parámetros

necesarios del nodo para poder representar la animación de los mensajes por la red en función del tipo de mensaje pasado como parámetro; *removeLastForwardSentToEveryNode*, donde se limpia la información acerca de los últimos mensajes enviados por la red relacionados con cada uno de los nodos; *createNodeIdentifier*, método que genera una animación con el identificador de un nodo durante unos instantes; y *createRadiation*, que genera la animación que se aprecia bajo ciertas condiciones cuando se produce una petición de un documento por parte de un nodo.

Controlador de la visualización de la animación

A continuación se presenta en la figura 4.49 el diagrama de la clase controladora que se encarga de gestionar los eventos relativos a la visualización y temporización de la animación, que además corresponde con el último diagrama de clases realizado dentro del paquete *controllerPackage* de la aplicación.

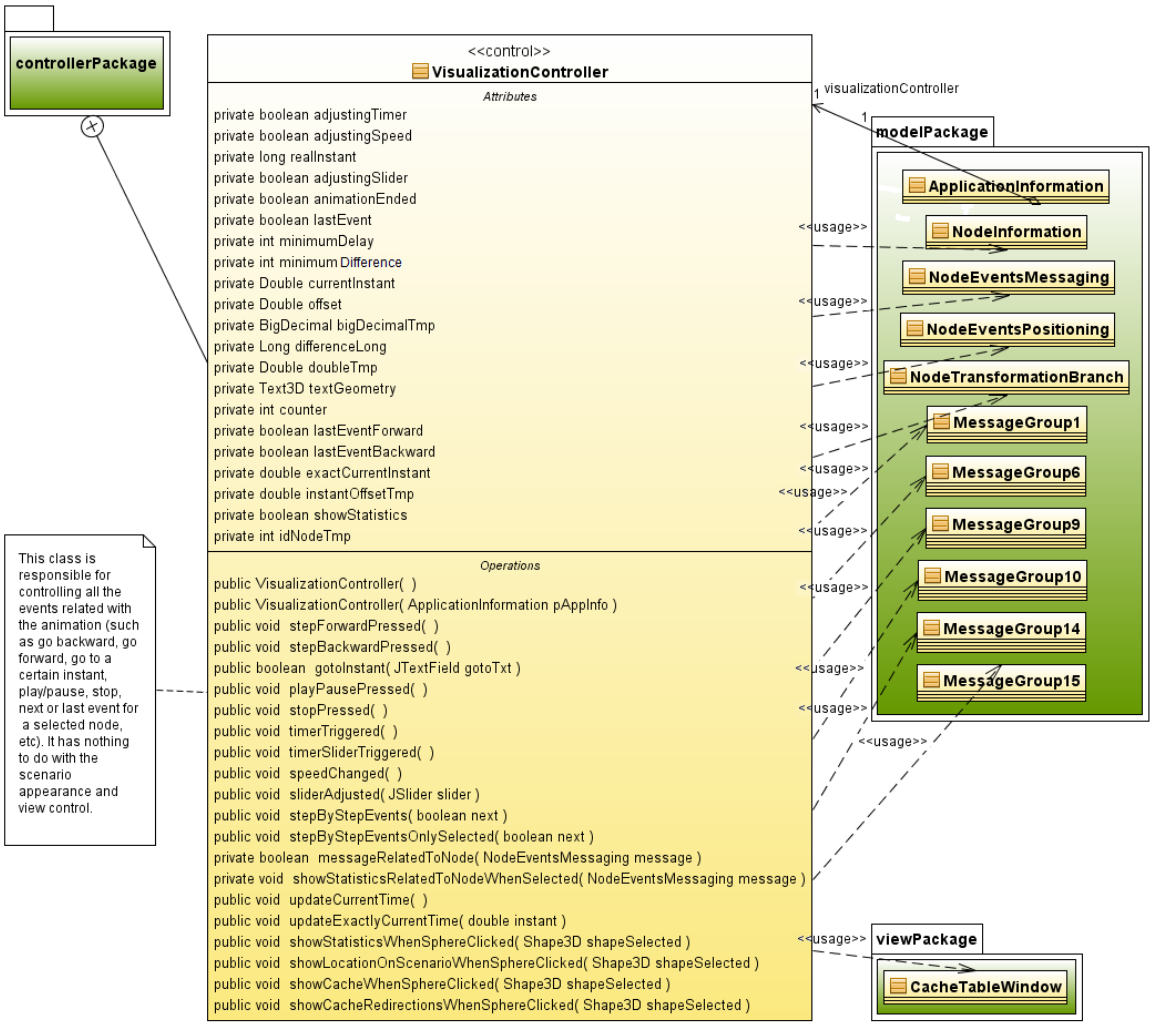


Figura 4.49. Diagrama de clases – Controlador de la visualización de la animación: *VisualizationController*

Como se ha comentado en varias ocasiones, esta clase controladora se denomina *VisualizationController*, y se encarga de controlar los eventos relacionados con la animación como avanzar, retroceder, ir a un instante de tiempo concreto, pausar y reanudar la animación, parar la animación, avanzar por eventos de los nodos de la red, etc.

En el diagrama se observan las diferentes relaciones que se establecen con otras clases. Así, las relaciones de uso que se establecen son: con las clases *NodeEventsMessaging*, *NodeEventsPositioning* y las clases que definen alguno de los tipos de mensajes que producen cambios en los estadísticos, para detectar cuándo se deben mostrar los estadísticos en la ventana principal de la aplicación; con *NodeInformation* y *NodeTransformationBranch*, para manejar la información relativa a un nodo; y con *CacheTableWindow*, porque el código de esta clase controladora es capaz de crear las ventanas con los contenidos de la caché local o de la caché de redirecciones de un cierto nodo a través del menú contextual asociado a dicho nodo del escenario. Como ha ocurrido con otros controladores, esta clase es un atributo de la clase principal de la aplicación *ApplicationInformation*.

Respecto de los atributos de esta clase, se puede decir que la mayoría de ellos se utilizan para tener datos de la temporización y de la reproducción de la animación. Es de interés destacar la función que realizan los métodos de esta clase, que además ya se explicó gracias al uso de diagramas de actividades: *stepForwardPressed*, donde se realiza el avance por saltos temporales hacia un instante posterior de la animación, acciones representadas en el diagrama de actividades de la figura 4.12; *stepBackwardPressed*, donde se realiza el retroceso por saltos temporales hacia un instante anterior de la animación, acciones representadas en el diagrama de actividades de la figura 4.13; *gotoInstant*, donde se avanza o retrocede en la animación hacia un instante absoluto en milisegundos introducido por el usuario; *playPausePressed*, a través del cual se activa la animación si se encontraba pausada, acciones que se representan en el diagrama de actividades de la figura 4.14, o bien se pausa la animación si se encontraba activa, acciones que también vienen especificadas en el diagrama de actividades de la figura 4.15; *stopPressed*, a través del cual se para la animación, llevando los nodos a sus posiciones iniciales, acciones que vienen detalladas en el diagrama de actividades de la figura 4.16; *timerTriggered*, método en el que se realizan las acciones necesarias cuando se agota el retardo fijado para el temporizador global de eventos de la animación, y que, como ha ocurrido anteriormente, se especifica en el diagrama de actividades de la figura 4.17; *timerSliderTriggered*, donde se realiza el avance de la barra horizontal o *slider* temporal

cada vez que se agota el retardo fijado para el temporizador creado a tal fin, realizando las acciones detalladas en el diagrama de actividades de la figura 4.18; *speedChanged*, donde se produce el cambio de la velocidad de visualización de la animación llevando a cabo las acciones representadas en el diagrama de actividades de la figura 4.19; *sliderAdjusted*, método en el que se actualiza el instante de la animación al desplazar manualmente el *slider* temporal; *stepByStepEvents* y *stepByStepEventsOnlySelected*, a través del cual se produce el avance y/o retroceso por los eventos relacionados con todos los nodos, en el primer caso, o solo de los nodos seleccionados, en el segundo caso, con las acciones que se representan en el diagrama de actividades de la figura 4.20; *messageRelatedToANode*, donde se estudia si un mensaje está relacionado con algún nodo del escenario que se encuentre seleccionado en ese momento; *showStatisticsRelatedToNodeWhenSelected* y *showStatisticsWhenSphereClicked*, a través de los cuales se muestra en la ventana principal de la aplicación un resumen con el valor actual que tengan los estadísticos de un cierto nodo, con la diferencia de que el primero estudia si el mensaje que se pasa como parámetro implica un cambio de estadísticos y que el nodo afectado esté seleccionado y el segundo muestra estos estadísticos cuando se solicite al clicar en el menú contextual de un determinado nodo; *showLocationOnScenarioWhenSphereClicked*, método que recupera las coordenadas reales de un nodo en el escenario y las muestra por pantalla cuando se solicite al clicar en el menú contextual de un determinado nodo; y *showCacheWhenSphereClicked* y *showCacheRedirectionsWhenSphereClicked*, a través de los cuales se muestra el contenido de la caché local o de la caché de redirecciones, según sea el caso, al clicar en el menú contextual de un determinado nodo.

4.5.2.3.5. Paquete de Utilidades: *Utilities Package*

Como último epígrafe se pretende acercar a aquellas clases que se han utilizado para actividades concretas que, si bien no corresponden con funcionalidades ni con el modelado de datos propios de la aplicación, han sido necesarias para el cumplimiento de los requisitos de la misma. Estas clases se han separado del resto gracias al uso del paquete de utilidades, *utilitiesPackage*, cuyo contenido se detalla en la figura 4.50.

Para los dos diagramas que se han realizado de este paquete, se va a introducir la finalidad de cada una de las clases a rasgos generales, sin mencionar las clases con las que se establecen relaciones de asociación o de uso, puesto que éstas ya se han comentado en los diagramas de dichas clases. Tampoco nos vamos a centrar ni en los atributos que éstas presenten ni en sus métodos, puesto que no se consideran lo suficientemente relevantes para el entendimiento del software creado como sí ha ocurrido en las demás clases.

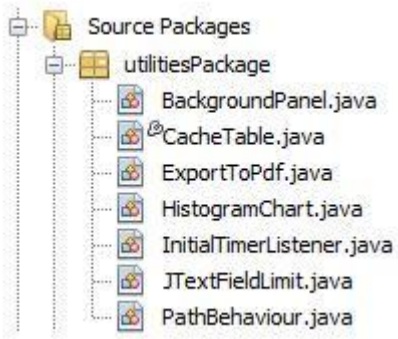


Figura 4.50. Contenido del paquete de Utilidades: *Utilities Package*

Clases de utilidades (1)

El primero los diagramas que contiene algunas de las clases de utilidades se muestra en la figura 4.51.

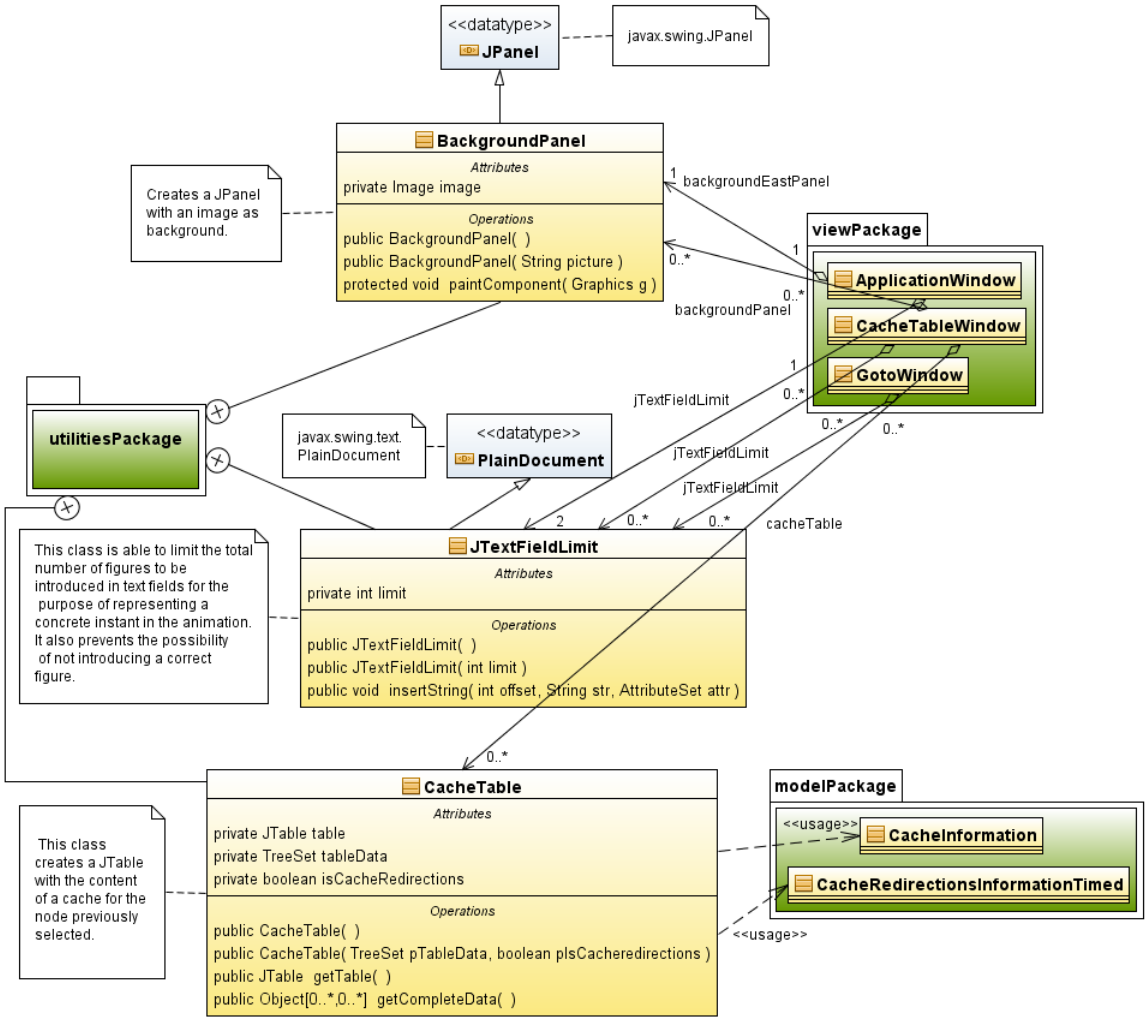


Figura 4.51. Diagrama de clases – Clases de utilidades 1: *Utilities (I)*

La finalidad de cada una de estas clases es:

- *BackgroundPanel*, utilizada para establecer una imagen de fondo en la zona derecha de la ventana principal de la aplicación y en las tablas con el contenido de las cachés local y de redirecciones de un nodo.
- *JTextFieldLimit*, en la cual se establece una propiedad para limitar el número de cifras a introducir en las cajas de texto utilizadas tanto en la ventana principal de la aplicación como en las ventanas para la muestra del contenido de las cachés local y de redirecciones, así como en la ventana de avance o retroceso de la animación a través de un instante absoluto.
- *CacheTable*, clase encargada de generar una tabla organizada en filas y columnas con el contenido de las cachés local y de redirecciones.

Clases de utilidades (2)

El segundo de los diagramas de clases de utilidades se muestra en la figura 4.52.

Como se ha realizado anteriormente, se va a identificar la finalidad de cada una de las clases que aparecen en el diagrama:

- *PathBehaviour*, clase que extiende de las clases de comportamiento definidas en la API Java3D con el propósito de eliminar del escenario gráfico, tras un cierto tiempo, aquellos elementos gráficos añadidos como consecuencia de las animaciones generadas para mostrar el tráfico de la red (elementos 3D del escenario con forma de cono que se desplazan, salto a salto, desde el nodo origen hasta el nodo destino que sigue un determinado mensaje), para crear la radiación en un nodo cuando realiza la petición de documento, y para representar los identificadores de nodo.
- *InitialTimerListener*, clase que controla el fin del temporizador inicial fijado para mostrar el logo de la aplicación cuando ésta se arranca y que se encarga de la creación de la ventana principal.
- *ExportToPDF*, que se encarga de generar el informe en PDF con el resumen de los estadísticos estudiados para todos los nodos de la red en un determinado instante de tiempo.
- *HistogramChart*, clase dedicada a la creación de los histogramas para cada uno de los estadísticos bajo estudio con el propósito de comparar los valores que presenten los

diferentes nodos del escenario. Estos diagramas son añadidos al informe PDF generado con la clase anterior.

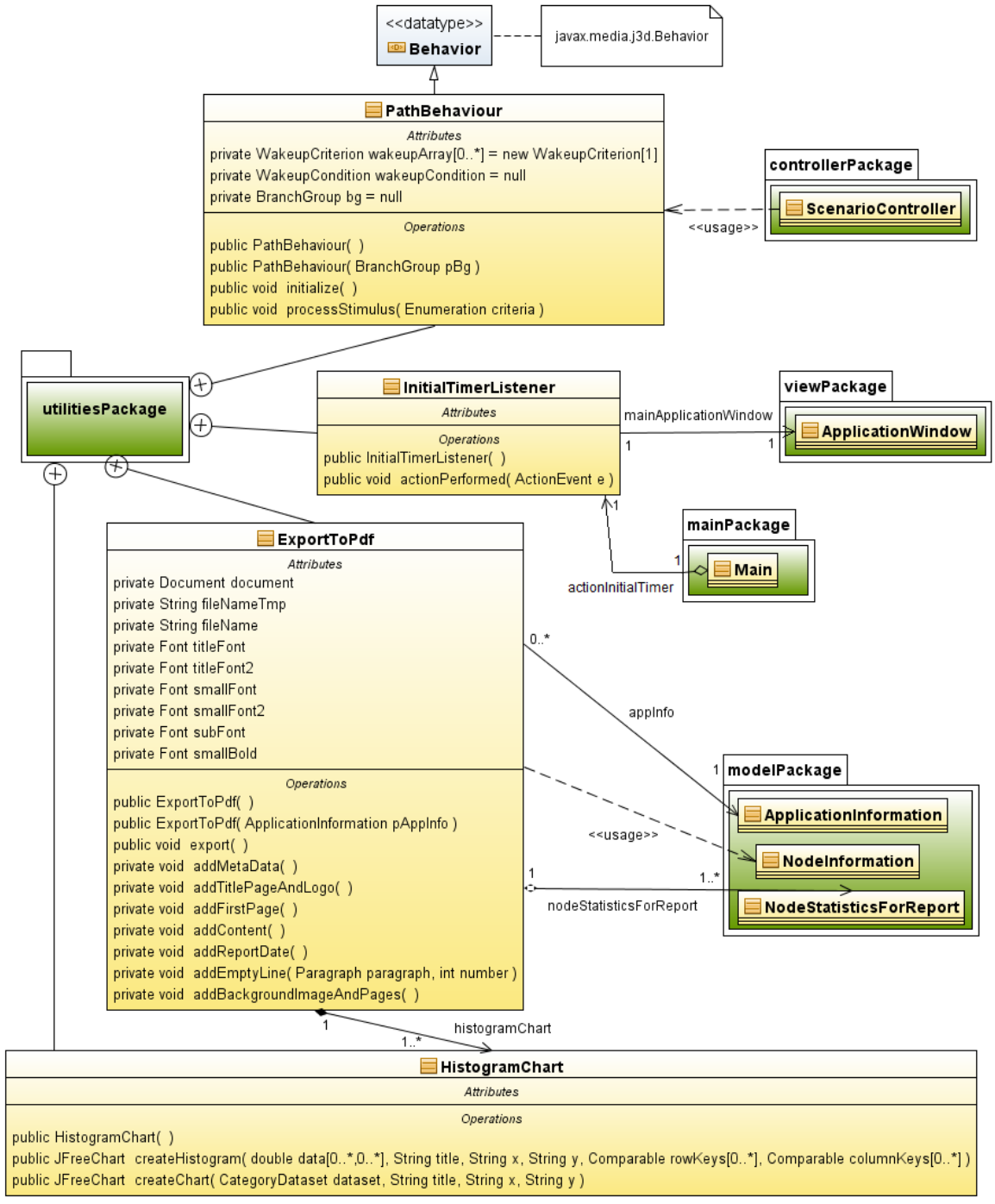


Figura 4.52. Diagrama de clases - Clases de utilidades 2: *Utilities (II)*

CAPÍTULO 5: Plan de pruebas

5.1. INTRODUCCIÓN A LA PRUEBA DEL SOFTWARE

Durante y después del proceso de implementación, el programa que se está desarrollando debe ser comprobado para asegurar que satisface su especificación y entrega la funcionalidad esperada. La verificación y la validación es el nombre dado a estos procesos de análisis y pruebas. La verificación y la validación tienen lugar en cada una de las etapas del proceso del software, comenzando con las revisiones de los requisitos y continuando con las revisiones del diseño e inspecciones del código hasta la prueba de producto final. La **verificación** y la **validación** representan conceptos diferentes: la validación se centra en comprobar que el producto creado es el correcto, mientras que la verificación implica estudiar si el producto se ha realizado correctamente. Es decir, el papel de la verificación implica comprobar que el software está de acuerdo con su especificación, teniendo en cuenta los requisitos funcionales y no funcionales, mientras que la validación se va a centrar principalmente en demostrar que el software hace lo que el cliente espera que haga. Dentro del proceso de verificación y validación, existen dos aproximaciones complementarias para el análisis y comprobación de los sistemas:

- Las inspecciones del software analizan y comprueban las representaciones del sistema tales como el documento de requisitos, los diagramas de diseño y el código fuente de programa. Pueden usarse las inspecciones durante todas las etapas de proceso. Se trata de una técnica estática, puesto que no necesita ejecutar el software en ningún ordenador.
- Las pruebas del software implican ejecutar una implementación de software con datos de prueba. Se examinan las salidas de software y su entorno operacional para comprobar que funciona tal y como se requiere. Las pruebas son una técnica dinámica de verificación y validación.

Aunque el uso de las inspecciones del software no es generalizado, la prueba de programas siempre será la principal técnica de validación y de verificación. Las pruebas implican ejecutar el programa con datos similares a los datos reales procesados por el programa. Los defectos en los programas se descubren examinando las salidas de software y buscando las anomalías. Las pruebas de defectos intentan revelar defectos en el sistema

en lugar de simular su uso operacional, y su objetivo es hallar inconsistencias entre un programa y su especificación.

Los **casos de prueba** son especificaciones de las entradas para la prueba y la salida esperada del sistema más una afirmación de lo que se está probando. El diseño de casos de prueba se centra en un conjunto de técnicas para la creación de casos que satisfagan los objetivos globales de la prueba. Para diseñar un caso de prueba hay que tener en cuenta que: la prueba es un proceso de ejecución de un programa con la intención de descubrir un error; un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces; y una prueba tiene éxito si descubre un error no detectado. Es decir, probar puede parecer un paso destructivo del software, pero hay que cambiar la idea de que una prueba tiene éxito si no descubre errores.

Para ser más efectivas las pruebas deberían ser dirigidas por un equipo independiente, es decir, los diseñadores de los casos de prueba no deberán ser los mismos que se encargaron del desarrollo del software, para evitar en la medida de lo posible que se arrastre el mismo error. Es de destacar que una prueba del software no puede asegurar la ausencia de defectos, si bien sólo puede demostrar que existen. Las pruebas deberían empezar por cuestiones muy específicas e ir abordando cada vez más funcionalidades; en cualquier caso, no son posibles las pruebas exhaustivas.

Existen dos tipos de pruebas:

- Pruebas de la caja negra: se llevan a cabo sobre la interfaz del software, es decir, comprobando su comportamiento entrada/salida. Se comprueba si la entrada se acepta de forma adecuada y si produce la salida correcta. Buscan obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales del software, tomando como entradas los datos, los eventos que se puedan producir y los requisitos funcionales, y estudiando su salida. Trata de encontrar errores en funciones incorrectas o ausentes, errores de interfaz, errores en estructuras de datos en accesos a repositorios externos, errores de rendimiento y errores de inicialización y terminación, ignorando intencionadamente la estructura de control.
- Pruebas de la caja blanca: prueban detalles procedimentales, de funcionamiento interno. Intentan asegurar que todas las sentencias y condiciones han sido ejecutadas al menos una vez, pero es difícil realizar una prueba exhaustiva con todas las posibilidades. Por ello se realizan pruebas selectivas, que se centran en encontrar los

caminos que no son los habituales, porque las sentencias que se ejecutan de forma excepcional son las que tienen mayor probabilidad de fallos.

En Ingeniería del Software, como en otras muchas disciplinas, el principio de Pareto o regla del 80-20 se puede enunciar de diversas formas:

- Si se habla de los costes de desarrollo de software se puede decir que el 80% del esfuerzo de desarrollo (en tiempo y recursos) produce el 20% del código, mientras que el 80% restante es producido con tan sólo un 20% del esfuerzo.
- Si se habla de pruebas del software, objeto de este capítulo, el principio nos dice que el 80% de los fallos de un software es generado por un 20% del código de dicho software, mientras que el otro 80% genera tan solo un 20% de los fallos.

Las pruebas de caja blanca se han ido realizando a medida que se incrementaba el tamaño del software. Una vez que se ha finalizado el desarrollo del software, se ha estudiado detenidamente, incluso a nivel de detalles procedimentales, cómo evoluciona el control del programa en respuesta a los diferentes eventos que se pueden producir, eventos que han sido explicados con la ayuda de los diagramas UML. Siguiendo con el principio de Pareto, se ha intentando llegar en la medida de la posible a aquellas situaciones que forman parte del 20% del software que tiene mayor posibilidad de producir errores. Por las características de las pruebas de caja blanca, más relacionadas con el seguimiento del control del programa, no se va a añadir más información.

Para las pruebas de caja negra, sin embargo, se han ideado una serie de casos de prueba con diferentes fines. Así, en primer lugar, se han generado una serie de escenarios con diferente número de nodos, velocidad de los mismos y tráfico generado. Para estos escenarios diferentes se ha medido el consumo de memoria que se produce, debido a que es una característica fundamental y crítica teniendo en cuenta que los archivos de texto de entrada a la aplicación pueden tener tamaños del orden de cientos de *megabytes*. En segundo lugar, se han creado diversos tipos de escenario diferentes en los que se han añadido las diferentes estrategias de caché consideradas en este proyecto para comparar su correcto funcionamiento. Además, para estos últimos escenarios se ha comprobado el cumplimiento de cada uno de los requisitos planteados durante la fase de análisis, recogidos en el capítulo cuarto de esta memoria.

A continuación se van a mostrar las características de las pruebas realizadas, explicando alguno de los parámetros que se han ido modificando para la obtención de los diferentes escenarios, así como los resultados obtenidos.

5.2. PRUEBAS REALIZADAS Y RESULTADOS OBTENIDOS

5.2.1. MODELO DE SIMULACIÓN

Antes de pasar a describir los resultados obtenidos, se van a explicar los parámetros que se han manejado para obtener los diferentes escenarios.

Los escenarios utilizados para la realización de las pruebas se han generado con el simulador NS-2, aunque como se ha comentado en otras ocasiones, no es obligatorio su uso para conseguir los archivos de entrada a la aplicación. Para las simulaciones se han activado todas las estrategias de caché planteadas.

El área que ocupan los nodos va a ser 1000 metros por 1000 metros, aunque la aplicación está preparada para diferentes valores de dimensiones.

Existen 1000 documentos diferentes susceptibles de ser solicitados, que se encuentran distribuidos entre dos servidores fijos de la red, de la siguiente manera: los documentos de identificador impar estarán almacenados en un nodo servidor de datos y los de identificador par, en el otro. De esta forma se consigue distribuir el tráfico por la red. El tamaño fijado para cada uno de los documentos ha sido de 1000 *bytes*. Estos documentos que van a ser almacenados en la caché local del nodo van a poseer un tiempo de expiración o TTL, cuyo valor va a seguir una distribución exponencial [51] según la ecuación (1). El valor medio de la función exponencial viene dado por $\mu = 1/\alpha$ y se ha fijado a un valor de 2000 segundos para todos los escenarios. Si se quisiera probar con diferentes valores de la variabilidad del tiempo de expiración de documentos, se modificaría este valor medio, obteniendo diferentes resultados, aunque quedan fuera del objetivo de este proyecto.

$$P(x) = \alpha e^{-\alpha x} \quad \textbf{(1)}$$

En cuanto al número de nodos utilizados, se han utilizado escenarios desde 25 nodos hasta 100 nodos como máximo. Además, se han considerado diferentes situaciones de movilidad de los nodos. Así, se han planteado tres diferentes configuraciones para nodos sin movilidad, utilizando rejillas de 5x5, 7x7 y 9x9 nodos, en las cuales a mayor número de nodos repartidos a modo de rejilla, más nodos se encuentran a un salto de éstos. Por otra parte, se han considerado otras 3 configuraciones diferentes de movilidad de los nodos, de forma que se desplazan por el escenario a velocidades de 1, 3 y 5 m/s.

El número total de peticiones de documentos que realizan los nodos durante el tiempo de simulación depende de la duración de la simulación y del patrón estadístico que siguen las mismas. Cuando una petición realizada por un nodo es servida, éste pasa a realizar la siguiente petición. Así, se ha considerado que el tiempo de espera en un nodo desde que una petición es servida hasta que realiza la siguiente petición de un documento sigue una distribución de probabilidad exponencial, ya expresada en (1), con un valor medio que se irá modificando entre 5, 25 y 50 segundos. De esta forma, cuanto menor sea el tiempo entre las peticiones sucesivas realizadas por los nodos, mayor será el tráfico generado por la red. Con los valores medios utilizados se ha podido evaluar un amplio rango de actividad de los nodos. Hay que tener en cuenta que cuando una petición no es servida, el nodo vuelve a solicitar el mismo documento cuando salte el *timeout* definido para ese nodo. El valor de los temporizadores para solicitar de nuevo el documento se ha fijado en 3 segundos para todas las simulaciones.

Además de la distribución de probabilidad que sigue el tiempo entre peticiones de documentos, se debe exponer el patrón estadístico que siguen dichas peticiones en cuanto al documento solicitado. Se define la ley Zipf [52] como (2):

$$P(i) = \frac{\beta}{i^{\alpha}} \qquad (\mathbf{2})$$

La ley Zipf indica, para las peticiones de documentos, que la probabilidad *P(i)* de que se solicite el *i-ésimo* documento más popular es inversamente proporcional a su ranking de popularidad. Así, el documento que esté en el número 1 del ranking de popularidad tendrá una probabilidad mayor que el que esté en las posiciones siguientes en el ranking. El parámetro α representa la pendiente o relación del número de referencias a los documentos en función de su ranking de popularidad *i*, fijada para las pruebas a 0.8, y el parámetro β representa el desplazamiento de la función, fijado a 1.

Para terminar con el modelo de la simulación utilizado, falta indicar que cada nodo implementa una caché local a nivel de aplicación que utiliza una determinada política de reemplazo. Además, la aplicación acepta cualquier política de reemplazo siempre que se respete el formato de los mensajes de actualización de la propia caché. Para las pruebas realizadas en las que se ha considerado que existe caché local en cada nodo, se ha establecido un tamaño de caché de 100000 *bytes* que, teniendo en cuenta que se ha fijado el tamaño de los documentos igual a 1000 *bytes*, estas memorias albergarán como máximo a 100 documentos. Para evitar la influencia en los resultados de las simulaciones de un

arranque frío de las cachés, se ha establecido que el 20% de las peticiones únicamente sirvan para que presenten un estado de llenado de documentos que condujera a lo que sería una situación normal de intercambio de documentos en la red. Las políticas de reemplazo que se han utilizado en las simulaciones han sido LRU, ya explicada en el capítulo tercero y no basada en una valoración, y las políticas TDS_D *(Time and Distance Sensitive – mainly Distance*, [53]) y SXO (*Size X Order*, [54]), ambas basadas en una valoración. Estas dos últimas pertenecen al grupo de políticas de reemplazo que se basan en la optimización de otras políticas de reemplazo muy extendidas como LRU pero teniendo además en consideración algunos parámetros del sistema, como el tamaño de los datos, el tiempo de acceso, la distancia en saltos a la que se encuentra el documento, etc. Así, en el caso de la política de reemplazo SXO, para obtener la valoración asociada se tienen en cuenta dos parámetros: el tamaño de los documentos almacenados (ya que aquellos documentos de mayor tamaño serían mejores candidatos para abandonar la memoria por ocupar mayor cantidad de espacio) y el tiempo de acceso de un documento (medido como el tiempo transcurrido desde se accedió la *k-ésima* vez, lo que implica que aquellos documentos que se accedieron hace más tiempo son los menos probables de ser accedidos de nuevo). Para el caso de la política de reemplazo TDS_D, para obtener la valoración se estudian los parámetros siguientes: tiempo de acceso, con el mismo significado que en el caso anterior; y la distancia en saltos a la que se encuentra un determinado documento en otro nodo de la red (de forma que se intente reducir la duplicidad de documentos en nodos cercanos). En esta última política de reemplazo planteada se le va a dar más peso al segundo parámetro, es decir, a la distancia a la que se encuentre el documento en la red.

5.2.2. CARACTERÍSTICAS DE LOS EQUIPOS UTILIZADOS PARA LA REALIZACIÓN DE LAS PRUEBAS

Debido a los requisitos de prueba de la aplicación en diferentes sistemas operativos, se han utilizado un total de 3 equipos para la realización de las pruebas. Así, para los sistemas operativos Ubuntu 9.10, Windows Vista y Windows 7, se han empleado dos equipos de iguales características: procesador Intel Core 2 Duo, 2.0 GHz (GigaHercios) y 4 GB (*GigaBytes*) de memoria RAM. Para el sistema operativo Windows XP se ha empleado un equipo con las siguientes características: Intel Pentium R, 1.7 GHz y 1 GB de memoria RAM (*Random Access Memory*).

5.2.2. CARACTERÍSTICAS DE LAS PRUEBAS REALIZADAS

Antes de centrarnos en los tipos de pruebas realizadas, es importante destacar que uno de los aspectos más importantes de la aplicación y que además se trata de un elemento clave de la funcionalidad que ofrece este software, como es el cálculo de los estadísticos, se ha ido verificando con sumo cuidado. Además de centrarnos en aquellos instantes en los que se debe modificar el valor de cada uno de los estadísticos estudiados, se ha verificado que el cálculo de los mismos que ofrece el software utilizado para la simulación de los diferentes escenarios (en nuestro caso NS-2) arroja los mismos valores que los que calcula la aplicación. Cada uno de esos valores estadísticos que ofrece la aplicación se van a explicar detalladamente en el manual de usuario que presenta el capítulo sexto.

Para la realización de las pruebas de consumo de memoria se ha utilizado uno de los equipos de las características presentadas bajo el sistema operativo Ubuntu 9.10, pudiendo fijar el máximo de memoria a utilizar por la JVM de 2.5 GB.

Para llevar a cabo las pruebas de comprobación de los requisitos se han empleado todos los sistemas operativos mencionados, utilizando escenarios que usen una cantidad memoria inferior a la disponible en cada equipo. Se han seleccionado escenarios al azar en cuanto a la actividad o tráfico en la red, número de nodos y movilidad de los mismos. Para esos escenarios diferentes se han comprobado cada uno de los requisitos, verificando que están alineados con el resultado final obtenido. Se podría especificar para comprobar cada uno de los requisitos (o por grupos de requisitos) la sucesión de los pasos seguidos para fijar un determinado estado inicial en la aplicación, indicando las características del propio estado, y las precondiciones que se deben cumplir. Además, tras generarse los posibles eventos, se podría explicar el comportamiento esperado en cada uno de esos casos y el comportamiento que finalmente se ha producido en la aplicación. Pero no se ha considerado de interés añadirlo por el tipo de aplicación desarrollada, que es de tipo reactivo, en la cual la mayoría de los estados iniciales y las precondiciones para la prueba de requisitos representan un estado de funcionamiento normal de la aplicación (exceptuando los requisitos relacionados con la carga inicial de los ficheros). Es más, gracias al manual de usuario y a la explicación realizada de las funcionalidades de la aplicación, que se presentará en el siguiente capítulo, se podrá deducir sin problemas las características que deba tener la propia aplicación para la consecución de cada uno de los requisitos. Por tanto, simplemente se concluye este apartado indicando que se han estudiado cada uno de los requisitos presentados en las tablas 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 4.10, 4.11, 4.12, 4.13, 4.14, 4.15 y 4.16, centrando la exposición que se

realiza en el presente capítulo en los resultados obtenidos con las pruebas respecto al consumo de la memoria.

5.2.3. ESTUDIO DEL CONSUMO DE MEMORIA

Además de la verificación de los requisitos en cada uno de los equipos y sistemas operativos planteados, que ya se ha comentado que carece de interés una descripción detallada de ello, se han realizado una serie de pruebas específicas para estudiar las limitaciones de la aplicación, aspecto que se ha considerado fundamental para su buen desempeño.

Las pruebas realizadas se han centrado en el estudio del consumo de memoria que necesita la aplicación para cargar la información de los archivos de entrada a la aplicación y representarla gráficamente, puesto que dichos archivos pueden llegar a ser de mucha extensión, en particular para los archivos de salida de las simulaciones con NS-2 con el tráfico generado en la red. Debido a que esta cantidad de datos puede representar un aspecto crítico para ejecutar la aplicación, se va a presentar la memoria total consumida, de forma aproximada, para la visualización de cada uno de los escenarios. En principio, para todas las simulaciones realizadas con NS-2 se va a fijar una duración de 20000 segundos de simulación. Pero para aquellos escenarios que necesiten más de la memoria asignada para la ejecución de la aplicación, fijada como máximo a 2.5 GB, se irá reduciendo la duración de la simulación hasta que se pueda visualizar el escenario con la memoria máxima fijada.

Por un lado, para la realización de las pruebas del consumo de memoria para el caso de nodos fijos, se han generado escenarios con distribución de nodos en forma de rejillas 5x5, 7x7, y 9x9. A su vez, para cada uno de estos escenarios se ha modificado el valor medio de la distribución exponencial que siguen las peticiones realizadas por los nodos, utilizando a tal fin los valores 5, 25 y 50 segundos. Por otra parte, para la realización de las pruebas de consumo de memoria realizadas para los nodos móviles, se han generado escenarios con un total de 25, 50 y 100 nodos. A su vez, como ocurre en el caso de nodos fijos, se van a fijar valores medios de 5, 25 y 50 segundos entre la recepción de un documento y la petición del siguiente en un nodo. Estos escenarios diferentes con movilidad se han repetido para los casos de 1, 3 y 5 m/s de velocidad de los nodos.

Para la representación de los resultados se van a utilizar dos tablas, una para los escenarios con movilidad de nodos (tabla 5.1) y otra para aquellos que no la tienen (tabla 5.2), presentando en cada caso el número de nodos involucrados, el valor medio en

segundos entre las peticiones realizadas por los nodos, la velocidad de los nodos (en el caso de nodos con movilidad), el tamaño en *bytes* del archivo de salida de simulación (*.txt*) utilizado junto con el número de líneas del mismo (para dar una idea de los datos que debe manejar la aplicación), la duración de la simulación, los valores aproximados de memoria máxima utilizada durante la carga inicial de archivos y de memoria media utilizada durante el funcionamiento normal de la aplicación.

Nº Nodos	Veloc. nodos (m/s)	Media pet. (segs)	Tamaño archivo *.txt* (MB)	Nº líneas archivo *.txt* (x 10^3)	Duración (segs)	Consumo mem. carga archivos (MB)	Consumo mem. animación (MB)
25	1	5	82	1421	20000	970	760
		25	36	629	20000	550	620
		50	24	424	20000	510	570
	3	5	92	1590	20000	960	860
		25	36	637	20000	730	760
		50	22	395	20000	510	580
	5	5	92	1590	20000	1060	760
		25	37	644	20000	650	700
		50	23	405	20000	510	590
50	1	5	231	3862	20000	1640	1330
		25	60	1016	20000	700	700
		50	33	549	20000	550	585
	3	5	244	4120	20000	1750	1340
		25	66	1115	20000	730	730
		50	35	592	20000	560	590
	5	5	244	4159	20000	1760	1350
		25	69	1167	20000	660	690
		50	38	638	20000	620	680
100	1	5	322	5369	13500	2540	1620
		25	125	2091	20000	1050	930
		50	66	1088	20000	820	820
	3	5	344	5791	13500	2560	1650
		25	140	2360	20000	1140	930
		50	73	1225	20000	930	780
	5	5	353	5982	13500	2560	1750
		25	153	2594	20000	1450	950
		50	76	1273	20000	840	840

Tabla 5.1. Consumo de memoria en escenarios con nodos con movilidad

Se ha decidido separar ambos valores de memoria (dados en MB, *MegaBytes*) porque en algunos casos, durante la carga de ficheros, se ha necesitado de una mayor cantidad de memoria que durante el funcionamiento normal de la aplicación, limitando el tiempo de simulación de los escenarios de entrada.

Estudiando los resultados obtenidos para los nodos móviles, se observa que los únicos casos en los que ha sido necesario bajar el tiempo de simulación que se ha utilizado (para reducir el tamaño del archivo de salida de la simulación *.txt* principalmente, ya que es el archivo que realmente limita) son aquellos escenarios que constan de 100 nodos y que además generan un tráfico de red dado por el tiempo medio entre peticiones de documentos de valor 5 segundos. En esos casos, se ha necesitado ajustar el tiempo de simulación a 13.500 segundos, de forma que se ajustara aproximadamente a los 2.5 GB de memoria máxima asignada a la JVM. Esto ocurre debido a que los archivos de salida de la simulación de estos escenarios tienen entre 5 millones y 6 millones de líneas. Es decir, se está creando la información que será manejada durante la animación que contiene ese mismo número de eventos, es decir, entre 5 millones y 6 millones de eventos, ya que cada una de esas líneas del archivo representa cada uno de los denominados eventos de mensajes.

En el resto de escenarios con movilidad de los nodos cuyos resultados se recogen en la tabla 5.1, se observa que se ha podido completar el tiempo de simulación de 20000 segundos sin alcanzar la memoria máxima asignada a la JVM. Si nos centramos en estudiar el tamaño de los ficheros de salida de la simulación *.txt* para los escenarios generados con movilidad de los nodos, se puede observar que, para la mayoría de los casos, exceptuando algunos de los escenarios que contienen 25 nodos, el número de líneas de dicho archivo aumenta con la velocidad de desplazamiento de los nodos. Es decir, a igualdad del valor medio de tiempo entre la recepción de un documento y la petición del siguiente por parte de un nodo y del número de nodos, aquellos escenarios que presentan una movilidad más pronunciada poseen un mayor número de eventos de mensajes asociados. Esta característica es fácilmente demostrable teniendo en cuenta que el protocolo estudiado se basa en la información de la red que se mantiene, tanto de los nodos vecinos como de los documentos en los nodos vecinos. Si existe una mayor movilidad de los nodos, la información asociada dejará de ser válida en un menor período de tiempo, lo que motiva la generación de un mayor número de mensajes en la red. Los únicos que no siguen esta regla corresponden a los escenarios con 25 nodos que presentan un valor medio entre la recepción y la petición del siguiente documento de 50 segundos. En esos casos, se

manifiesta que no existe ninguna relación entre la velocidad de los nodos, con valores 1, 3, y 5 m/s, y el número de eventos generados. Este resultado se puede explicar considerando que el tiempo que los nodos tardan de media en solicitar documentos es lo suficientemente grande como para que, en el instante en el que se produce la petición de un documento, la información de la red que posee cada nodo ya ha dejado de ser válida, independientemente de que los nodos se desplacen a una velocidad mayor o menor.

Nº Nodos	Media pet. (segs)	Tamaño archivo *.txt* (MB)	Nº líneas archivo *.txt* (x 10^3)	Duración (segs)	Consumo mem. carga archivos (MB)	Consumo mem. animación (MB)
Rejilla 5x5 (25 nodos)	5	115	1895	20000	1060	860
	25	28	455	20000	550	610
	50	15	239	20000	440	500
Rejilla 7x7 (49 nodos)	5	260	4243	20000	1770	1470
	25	60	1001	20000	700	730
	50	32	534	20000	530	610
Rejilla 9x9 (81 nodos)	5	334	5499	15000	2500	1770
	25	103	1726	20000	1020	920
	50	55	918	20000	690	730

Tabla 5.2. Consumo de memoria en escenarios con nodos sin movilidad

Para los resultados de las simulaciones de escenarios sin movilidad de los nodos, recogidos en la tabla 5.2, se observa una situación similar a la de los escenarios con movilidad. En el único caso en el que ha sido necesario disminuir la duración de la simulación para poder representar el escenario con la memoria máxima asignada a la JVM ha sido para la rejilla de 9x9 nodos, es decir, 81 nodos, cuando el tiempo medio entre la recepción de un documento y la petición de otro documento por parte de un nodo tiene un valor de 5 segundos. En ese caso, ha sido necesario ajustar la duración de la simulación a 15.000 segundos.

Como este estudio se ha centrado en obtener el consumo de memoria de los diferentes escenarios simulados se ha decidido adjuntar una tabla que indique los valores medios de entre todos los resultados, de forma que se pueda tener un valor representativo medio del número de MB de memoria consumidos por cada 1000 líneas del archivo de salida de simulación. Hay que indicar que estos valores son meramente unas referencias obtenidas para los escenarios simulados que se caracterizan por poseer un número de nodos entre 25 y 100, lo que implica un tamaño de los archivos de salida de la simulación *.txt* entre 15 y 353 MB. En la tabla 5.3 se reflejan estos valores medios de memoria calculados a partir de

los resultados de las simulaciones de escenarios con y sin movilidad de los nodos, separando los diferentes casos por el valor medio del tiempo entre la recepción de un documento y la solicitud del siguiente por parte de un nodo.

Media de tiempo entre recepción de respuesta y siguiente petición (segundos)	Consumo de memoria medio durante la carga de archivos (MB/1000 líneas)		Consumo de memoria medio durante la animación (MB/1000 líneas)	
	Valor medio	Rango de valores [mínimo – máximo]	Valor medio	Rango de valores [mínimo – máximo]
5	0.5	[0.4172 – 0.6826]	0.3793	[0.2849 – 0.6826]
25	0.7486	[0.4831 – 1.2088]	0.7507	[0.3662 – 1.3407]
50	1.0359	[0.6599 – 1.8411]	1.1231	[0.6367 – 2.092]

Tabla 5.3. Valores medios del consumo de memoria por cada 1000 eventos del archivo de salida de simulación *.txt*

Para el cálculo de los valores mínimo, medio y máximo del consumo de memoria que realiza la aplicación en los escenarios presentados, ha sido necesario calcular para cada uno de ellos el valor de la memoria utilizada durante la carga del archivo de simulación y durante la animación, expresada en MB por cada 1000 líneas del archivo de salida de simulación *.txt*. Se ha obtenido que, para escenarios con un determinado número de nodos y con una velocidad dada (en el caso de escenarios con movilidad), estos valores han sido menores para valores medios inferiores del tiempo entre la recepción de un documento y la petición del siguiente. Es decir, los valores obtenidos para una media de 5 segundos han sido menores que para una media de 25 segundos, y a su vez éstos han sido inferiores que para una media de 50 segundos. Por otra parte, si se comparan aquellos escenarios que poseen un determinado valor medio de tiempo entre la recepción de un documento y la petición del siguiente por parte de un nodo, e independientemente de la velocidad en el caso de que se traten de escenarios con movilidad de los nodos, se observa que a medida que aumenta el número de nodos, en líneas generales disminuye el valor medio en MB por cada 1000 eventos de mensajes. Todo esto está indicando que la memoria que utiliza la aplicación (en términos del tamaño en MB del archivo de salida de la simulación *.txt* por cada 1000 líneas del mismo) tanto para la carga y lectura de ficheros como para la utilización normal de la aplicación para la visualización de animaciones, es inferior para aquellos escenarios que poseen más tráfico, es decir, más eventos de mensajes. Los escenarios que generan más tráfico son aquellos que poseen un mayor número de nodos y una media en segundos entre la recepción de un documento y la petición del siguiente menor. Aunque en un principio lo más natural sería pensar justo en el resultado contrario,

este hecho nos está indicando que, con el aumento del tráfico producido en una red, el crecimiento del consumo de memoria de la aplicación es más lento que el crecimiento del número de líneas del archivo de salida de simulación, es decir, que el número de eventos.

En resumen, en las pruebas de consumo de memoria, se ha puesto de manifiesto que únicamente para el caso de simulaciones de escenarios con máximo número de nodos (100 nodos para escenarios con movilidad y 81 nodos para escenarios con nodos fijos) en los que se ha fijado un tiempo medio entre la recepción satisfactoria de un documento y la petición del siguiente a un valor de media 5 segundos, ha sido necesario bajar la duración de la simulación hasta ajustarse a la memoria máxima fijada para la aplicación.

Para las aplicaciones desarrolladas en Java, se puede fijar por parte del propio usuario la cantidad de memoria asignada al arrancar la aplicación y la memoria máxima que se le quiere asignar a la JVM. Para realizar esto, se crea un archivo de texto a modo se *script* que contenga una línea de las siguientes características:

java -jar -Xms128m -Xmx1024m Proyecto.jar

En este caso, la aplicación se denomina *Proyecto.jar*, y se le ha asignado para su ejecución una memoria mínima de 128 MB y máxima de 1024 MB. Es importante destacar que estos archivos de texto generados tendrán la extensión *.bat* en el caso del sistema operativo *Windows* (archivo *batch* de procesamiento por lotes, que contiene un conjunto de comandos MS-DOS), o bien la extensión *.sh* en el caso de sistemas operativos Linux (archivo denominado *shell script* que contiene igualmente un conjunto de comandos, que se debe hacer además ejecutable).

CAPÍTULO 6: Manual de usuario

6.1. INTRODUCCIÓN

En el presente capítulo se van a ir explicando las acciones que deberá llevar a cabo el usuario para poder disfrutar de todas las funcionalidades que ofrece la aplicación software desarrollada en este proyecto.

6.1.1. VISIÓN GENERAL DE LA APLICACIÓN

Durante la visualización de la animación, el aspecto de la ventana principal de la aplicación viene reflejado en las figuras 6.1 y 6.2. En esos escenarios se presentan 25 y 100 nodos, respectivamente, de los cuales algunos de ellos han sido seleccionados. Además, se están mostrando las coberturas para algunos de los nodos. Asimismo, se presentan los identificadores de varios nodos.

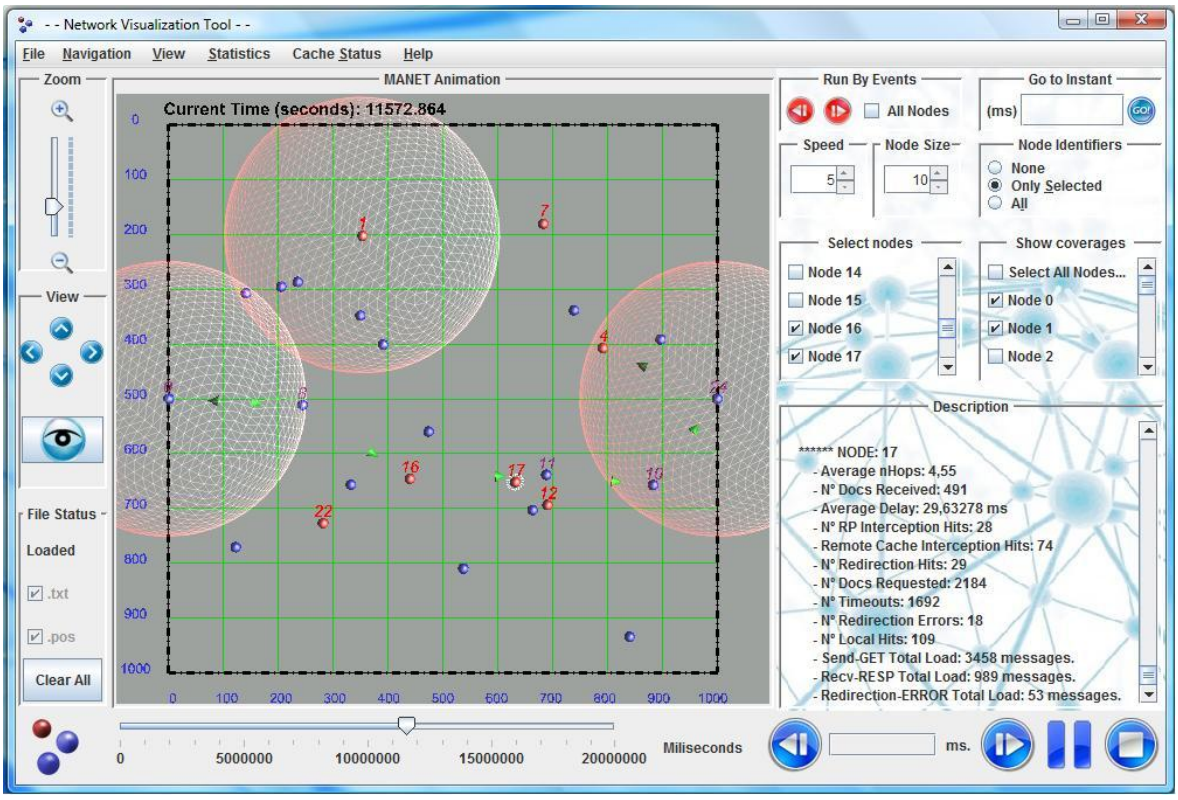


Figura 6.1. Visión general de la aplicación con un escenario de 25 nodos

Se ha pretendido comenzar este manual de usuario con una visión general de la ventana principal de la aplicación para poder ir centrándose en cada una de las partes que la

componen. Se van a ir destacando las funcionalidades que se pueden conseguir en cada uno de los elementos de la ventana por pequeñas agrupaciones.

Antes de centrarnos en las funciones que ofrece el software desarrollado, se van a introducir cada uno de las variables estadísticas que se recogen en la aplicación, definiendo para cada una de ellas el significado que poseen. Es necesario recordar que existe una serie de elementos de necesaria inserción en los documentos de entrada de la aplicación, puesto que determinados valores serán tomados únicamente de dichos archivos. Además, estos elementos deberán poseer el formato adecuado, de la misma forma que pasa con los eventos que se recogen de ambos ficheros de entrada. El formato que deben tener todos estos elementos viene detallado en el Apéndice A.

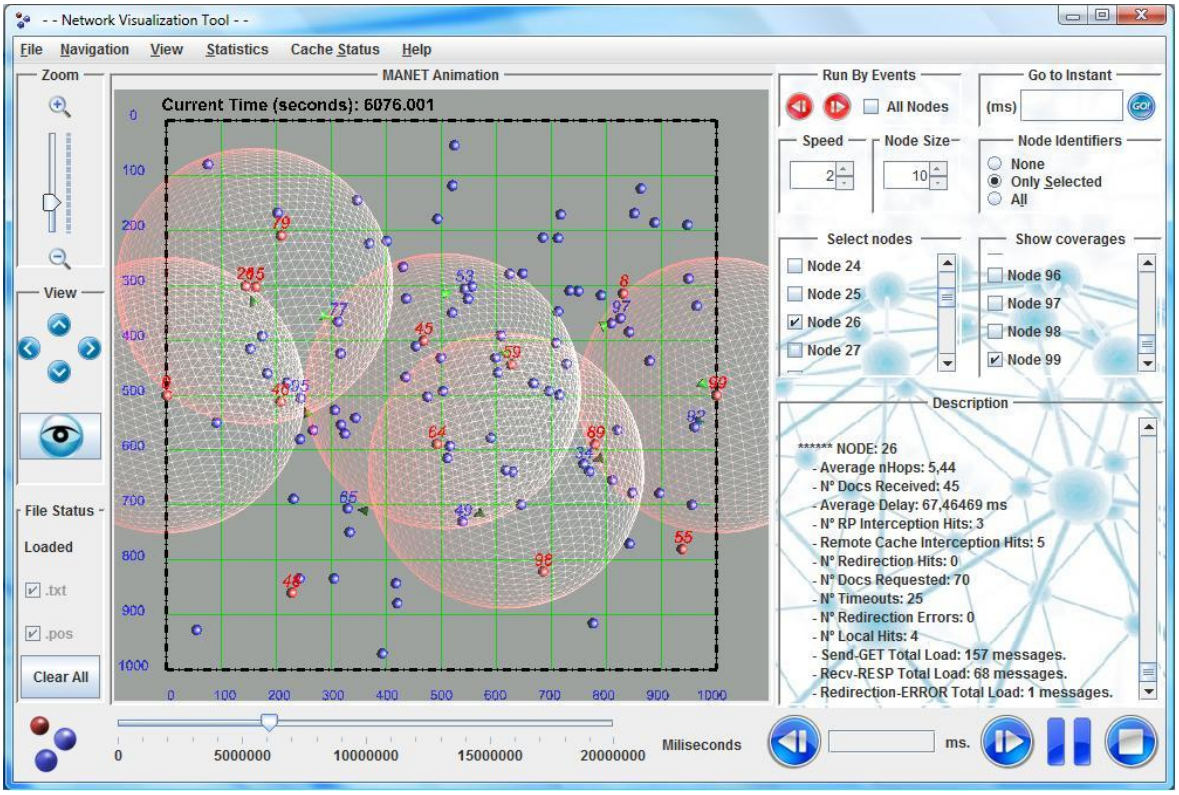


Figura 6.2. Visión general de la aplicación con un escenario de 100 nodos

6.1.2. DEFINICIÓN DE LOS VALORES ESTADÍSTICOS BAJO ESTUDIO

Debido a las características del protocolo implementado de tipo petición-respuesta junto con las estrategias de caché para estas redes MANET que se han presentado en el tercer capítulo, el estudio de algunos parámetros estadísticos es fundamental para poder comparar diferentes escenarios. Es decir, para verificar la incidencia de las estrategias de caché en este tipo de redes es necesario presentar las siguientes variables estadísticas, variables calculadas para cada nodo, que además aparecen mostradas tanto en la ventana principal de

la aplicación como en el documento PDF generado como se verá más adelante. Hay que tener en cuenta que algunas de estas variables estadísticas poseerán un valor nulo inicial (cuando representen el total) o *Not Available* en el resto de casos (cuando se trate de valores estadísticos que representan un valor medio o bien expresen un *ratio*) hasta que se produzca el evento de *reset* para un determinado nodo, instante a partir del cual se empezarán a calcular los valores estadísticos para dicho nodo. Cada uno de los ratios que se presentan se expresan en tanto por ciento, con un redondeo de dos decimales.

- *"Nº Docs Requested"*: número total de peticiones que ha realizado un determinado nodo hasta el instante actual.
- *"Nº Timeouts"*: número total de *timeouts* que se han producido en un determinado nodo hasta el instante actual. Estos *timeouts* se producen, en los escenarios simulados, a los 3 segundos después de la solicitud de un documento por parte de un nodo, aunque podría ser cualquier valor fijado para el simulador que se utilizase.
- *"Nº Docs Received"*: número total de documentos que se han recibido en un nodo de forma satisfactoria, de entre todas las peticiones realizadas, hasta el instante actual.
- *"Average nHops"*: número medio de saltos por los que han pasado los mensajes de petición y respuesta de un documento realizada por un determinado nodo, hasta el instante actual. Este valor medio sólo se actualiza cada vez que se recibe satisfactoriamente un documento solicitado. Esta variable se presenta con un redondeo de dos cifras decimales.
- *"Average Delay"*: retardo medio en milisegundos entre las peticiones de un documento y sus respuestas para un determinado nodo, hasta el instante actual. Este valor medio sólo se actualiza cada vez que se recibe satisfactoriamente un documento solicitado. Esta variable se presenta con un redondeo de cinco cifras decimales.
- *"Ratio Timeouts To Documents Requested"*: cociente entre el número total de *timeouts* generados en un determinado nodo y el número total de documentos que ha solicitado, hasta el instante actual. Este ratio da una idea del porcentaje de las peticiones que han generado *timeout* y que, por tanto, ha sido necesario retransmitirlas de nuevo.
- *"Nº RP Interception Hits"*: número total de peticiones servidas, hasta el instante actual, por un nodo diferente del nodo servidor que ha sido encontrado por el protocolo de encaminamiento por poseer una copia válida en su caché del documento solicitado.

- *"Nº Remote Cache Interception Hits"*: número total de peticiones servidas, hasta el instante actual, que han sido interceptadas por un nodo en su camino desde el nodo origen hacia el servidor.
- *"Nº Redirection Hits"*: número total de peticiones, hasta el instante actual, que han sido redirigidas hacia un nodo diferente del servidor y que han sido respondidas con éxito.
- *"Nº Redirection Errors"*: número total de peticiones, hasta el instante actual, que han sido redirigidas hacia un nodo diferente del servidor y que no han sido respondidas con éxito, puesto que el nuevo nodo destino no tenía una copia válida en su caché.
- *"Nº Local Hits"*: número total de peticiones, hasta el instante actual, que han sido servidas por la caché local.
- *"Ratio RP Interception Hits to Documents Received"*: cociente, hasta el instante actual, entre el número de peticiones servidas usando la intercepción con recursos de encaminamiento y el número total de documentos recibidos. Este ratio da una idea del porcentaje de las peticiones que han sido servidas a través de esta estrategia de caché.
- *"Ratio Remote Cache Interception Hits to Documents Received"*: cociente, hasta el instante actual, entre el número de peticiones servidas que han sido interceptadas de forma remota y el número total de documentos recibidos. Este ratio da una idea del porcentaje de las peticiones que han sido servidas a través de esta estrategia de caché.
- *"Ratio Redirection Hits to Documents Received"*: cociente, hasta el instante actual, entre el número de peticiones servidas usando el mecanismo de redirección y el número total de documentos recibidos. Este ratio da una idea del porcentaje de las peticiones que han sido servidas a través de esta estrategia de caché.
- *"Ratio Local Hits to Documents Received"*: cociente, hasta el instante actual, entre el número de peticiones servidas por la caché local y el número total de documentos recibidos. Este ratio da una idea del porcentaje de las peticiones que han sido servidas a través de esta estrategia de caché.
- *"Send-GET Total Load"*: cantidad de mensajes GET, hasta el instante actual, que han sido transmitidos o retransmitidos por el nodo.
- *"Recv-RESP Total Load"*: cantidad de mensajes RESP, hasta el instante actual, que han sido transmitidos o retransmitidos por el nodo.
- *"Redirection-ERROR Total Load"*: cantidad de mensajes de error, hasta el instante actual, que han sido transmitidos o retransmitidos por el nodo.

6.2. FUNCIONES DE LA APLICACIÓN

A continuación se van a ir presentando cada una de las funcionalidades que ofrece la aplicación agrupadas en diferentes apartados según el tipo de función que representa y el lugar de la ventana principal de la aplicación en el que aparece.

6.2.1. ARRANQUE DE LA APLICACIÓN

Para iniciar la aplicación, se debe ejecutar el archivo JAR asociado. Tras el arranque de la aplicación, va a aparecer una ventana emergente, ventana que carece de controles ni barras de menú, y que se utiliza simplemente para mostrar el logo de la aplicación durante la creación de los componentes y controles de la ventana principal de la aplicación. El aspecto de esta ventana inicial de la aplicación se muestra en la figura 6.3. Además del logo de la aplicación, aparece el logo del departamento para el que se está desarrollando este proyecto, departamento de Tecnología Electrónica de la Universidad de Málaga.

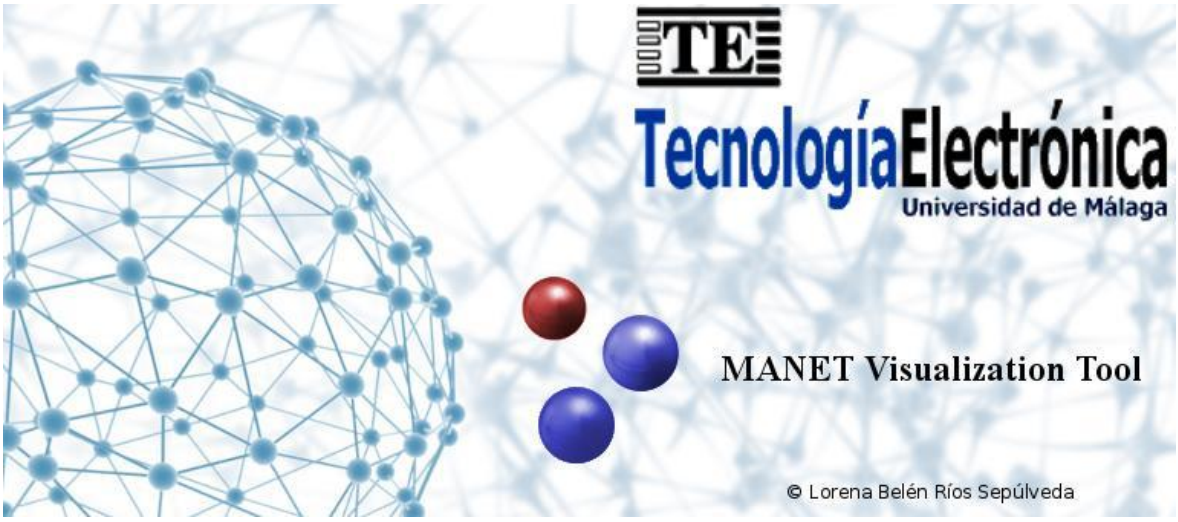


Figura 6.3. Ventana emergente inicial de la aplicación

Una vez que se ha creado la ventana principal de la aplicación, van a aparecer por defecto todos los controles de la aplicación deshabilitados, exceptuando los controles de la barra de menús que se destinan a la selección y apertura de los archivos de entrada de la aplicación, como se verá más adelante, y el botón de borrado de la información de la aplicación. Estos controles permanecerán deshabilitados hasta que se haya generado un escenario con éxito.

El aspecto de la ventana principal de la aplicación tras el arranque de la misma se presenta en la figura 6.4.

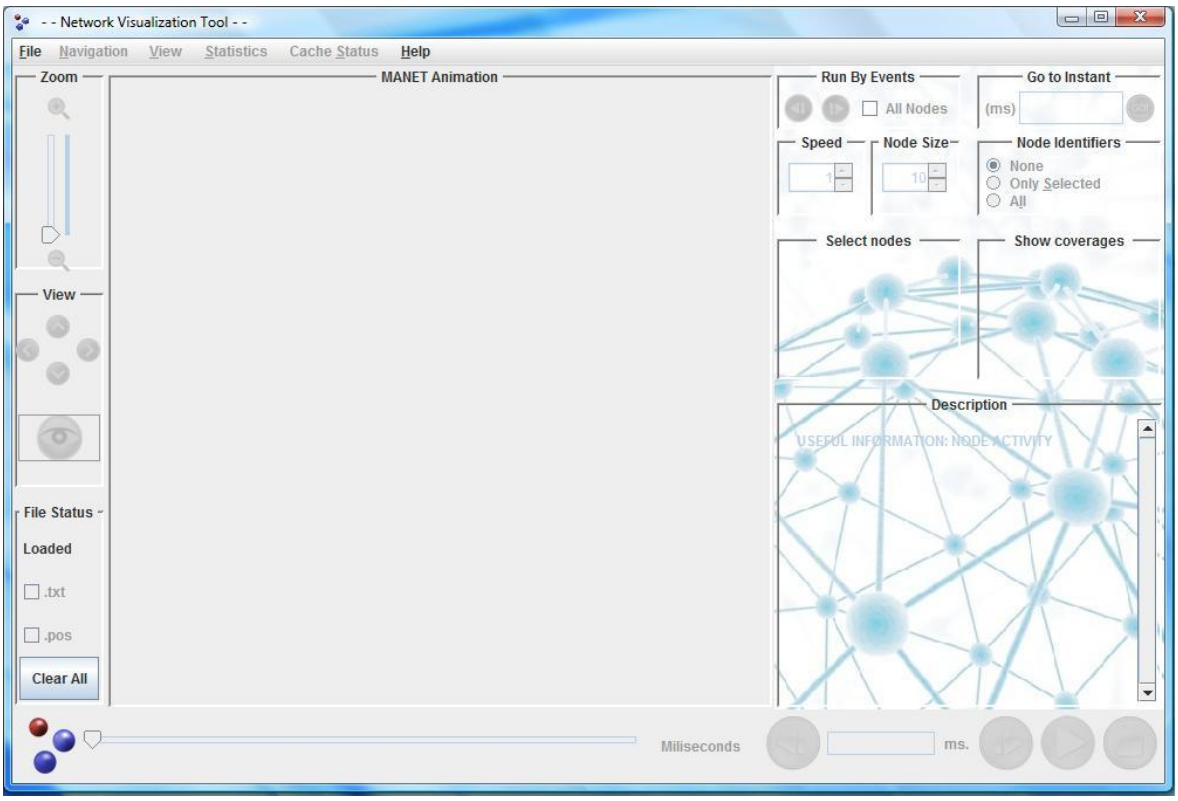


Figura 6.4. Aspecto de la aplicación recién cargada

6.2.2. APERTURA DE FICHEROS Y CARGA DE DATOS

Una vez que se ha abierto la aplicación y se ha generado su ventana principal, el siguiente paso para la creación de un determinado escenario es la selección y apertura de los archivos de entrada, para poder pasar a la carga de los datos procedente de los mismos. Para llevar a cabo esto, se utiliza el menú denominado *File* y, en concreto, el control de carga de archivos o *Load*, tal y como se muestra en la figura 6.5.

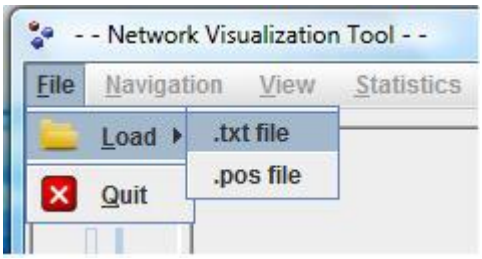


Figura 6.5. Selección y apertura de los archivos de entrada a la aplicación

Tal y como se exige en los requisitos de la aplicación, para la creación de un escenario basta con la carga del archivo de movilidad de los nodos *.pos*, en cuyo caso sólo se podrá visualizar una animación con los movimientos de los mismos, sin tener, en consecuencia, información acerca del tráfico que circula por la red. Si por el contrario se abre en primer lugar el archivo de salida de la simulación *.txt*, se necesitará de la carga del archivo de

movilidad de los nodos para la creación del escenario. Así, en el caso de que se abra en primer lugar el archivo de movilidad de los nodos, se creará automáticamente un escenario, y en cualquier instante se podrá abrir el fichero de salida de la simulación para completar la información del escenario, añadiendo el envío de mensajes por la red, y situando la animación en el instante inicial.

En la figura 6.6 se observa la ventana de información que genera la aplicación mientras se encuentra cargando los datos del fichero de salida de la simulación *.txt*.

Figura 6.6. Ventana de información durante la carga de los datos del archivo de salida de la simulación *.txt*

Además, en la ventana informativa se mostrará el nombre del archivo que se ha seleccionado y abierto. Para el caso de la carga de los datos provenientes del archivo de movilidad de los nodos *.pos*, aparecerá una ventana de información similar.

Cuando se elige la opción de cargar archivo, a través de *Load .txt file* y *Load .pos file*, se generan unas ventanas cuyo objetivo es la selección de archivos, con la posibilidad de navegar por el sistema de archivos o disco. El aspecto de este tipo de ventanas aparece en las figuras 6.7 (para el caso de archivos de salida de la simulación *.txt*), y 6.8 (para el caso de archivos de movilidad de los nodos *.pos*). Además, en estas ventanas de selección se genera un filtrado del tipo de archivos válidos para ser seleccionados según la extensión buscada en cada caso. Así, sólo se mostrarán las carpetas y aquellos archivos que posean una extensión del tipo buscado.

Cuando se ha seleccionado un archivo y se ha optado por abrirlo, se pasa a estudiar el contenido del mismo, detectando y reconociendo los valores que deben aparecer por defecto para cada tipo determinado de archivos y su formato, como se comentó anteriormente, además de identificar los diferentes tipos de eventos (ya sean eventos

debidos al intercambio de mensajes o bien a los cambios de dirección de los nodos) y su información asociada atendiendo al formato de eventos presentado en el Apéndice A.

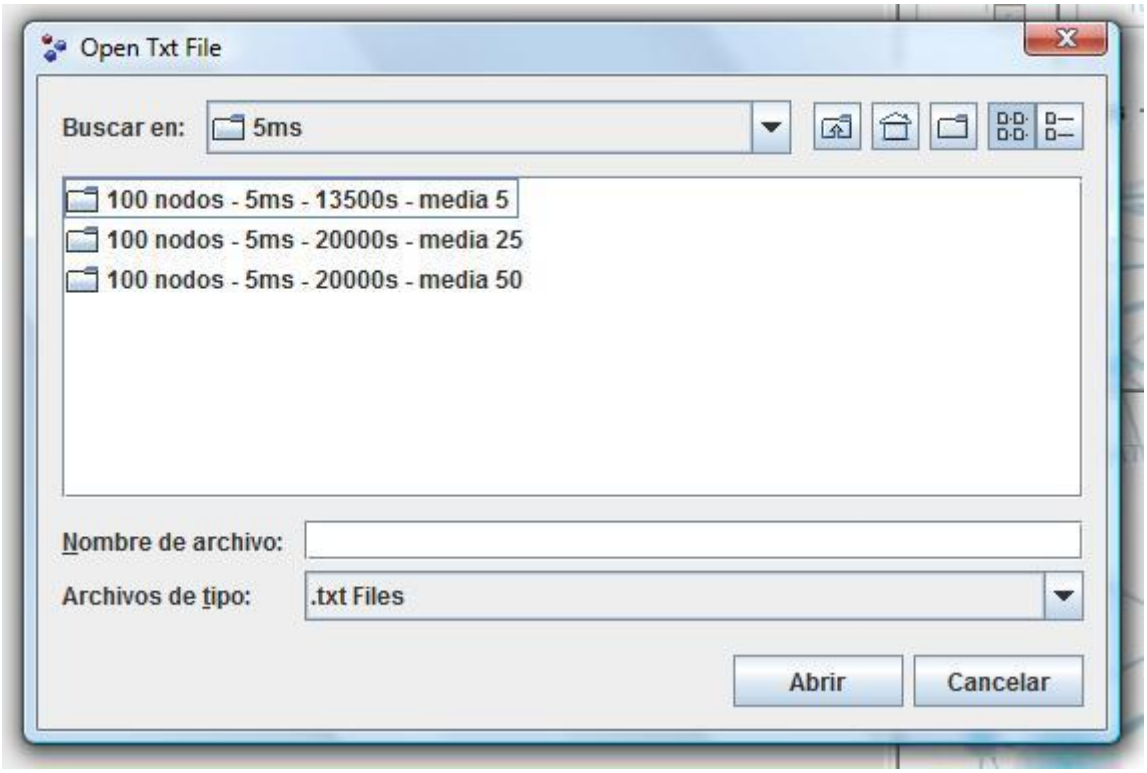


Figura 6.7. Selección del archivo de salida de la simulación *.txt*

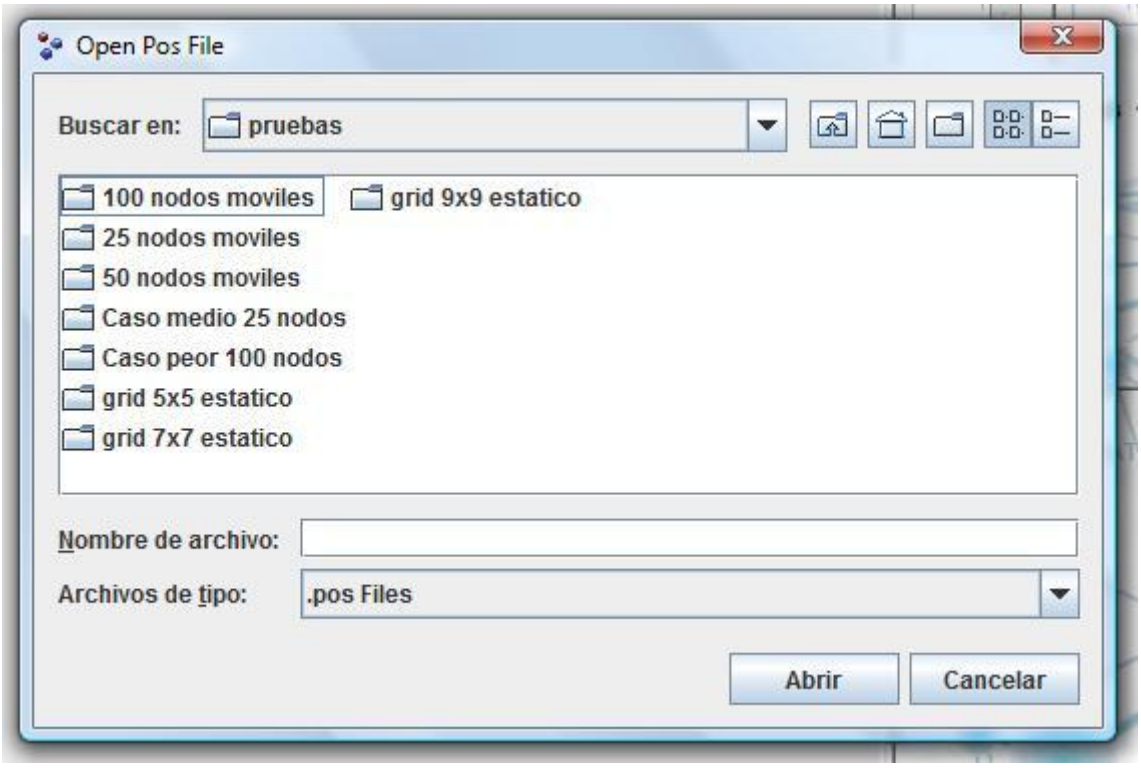


Figura 6.8. Selección del archivo de movilidad de los nodos *.pos*

Una vez que se han seleccionado los archivos de entrada de la aplicación, se pasa a la creación del escenario. Además, se procede a realizar la mezcla de eventos provenientes de los dos archivos posibles de entrada en el caso de que se hayan abierto ambos, puesto que deben ser ordenados temporalmente de forma ascendiente por el instante de ocurrencia de dichos eventos. El software muestra una ventana de información indicando este proceso, como se muestra en la figura 6.9.

Figura 6.9. Ventana de información durante la creación del escenario

Además de la creación del escenario, es necesario establecer cuáles son los primeros eventos que deben ser representados para que la animación comience desde su inicio. Este proceso viene mostrado en la figura 6.10, en la cual se observa una barra de progreso indicando el fin de la creación del escenario y el momento a partir del cual se puede comenzar con la animación de la simulación.

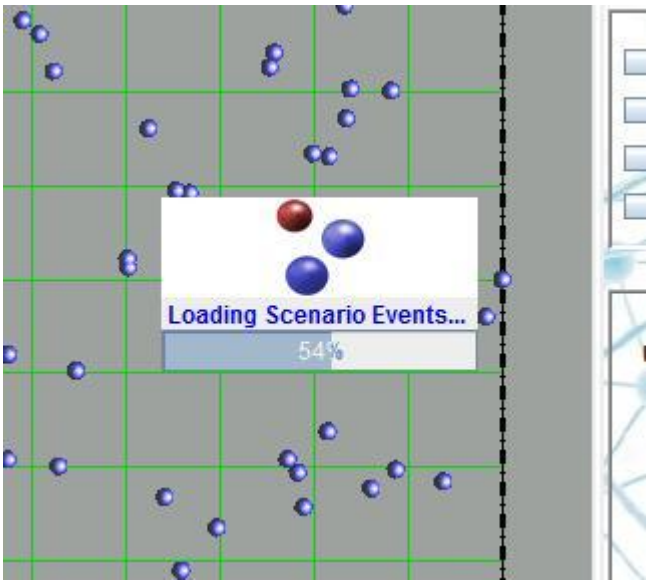


Figura 6.10. Carga de los eventos iniciales del escenario a representar

6.2.3. PANEL DE CONTROL DE LA VISTA DEL ESCENARIO

El panel lateral izquierdo de la ventana principal de la aplicación está dividido en tres partes. En este apartado nos vamos a centrar en las dos primeras partes, que constituyen las zonas superior y central, donde a su vez se encuentran diferentes controles que se irán detallando de forma separada.

6.2.3.1. Ajuste del *zoom* de la vista del escenario

La figura 6.11 muestra los controles para el manejo del *zoom* del escenario virtual.

Figura 6.11. Controles del *zoom*

Los controles de la vista actúan directamente sobre la cámara que enfoca al escenario virtual. Así, las opciones del *zoom* realizan un acercamiento o alejamiento del punto de vista respecto del escenario. Para llevar a cabo un ajuste del *zoom*, se puede realizar de tres maneras: a través de la barra vertical de desplazamiento, a través de los botones que se encuentran en la parte superior e inferior de la misma, o bien, tras posicionarse con el ratón en cualquier punto del escenario virtual, con el uso del *scroll* del mismo.

El *zoom* inicial donde se sitúa la cámara tras la carga del escenario y tras el cambio de vista rotada a vista normal (y viceversa) es estimado inicialmente por el software, de forma que el escenario virtual quede completamente dentro de la zona de la ventana principal de la aplicación destinada a ello, en función de las diferentes dimensiones que pudiera tener éste. En las figuras 6.1 y 6.2 se presenta el acercamiento inicial de la cámara de visualización obtenido para los escenarios generados, en esos casos de 1000 por 1000 metros. En dichas figuras se observa como esos escenarios quedan completamente dentro de la zona gráfica de la ventana principal de la aplicación destinada a ellos.

Es importante destacar que a medida que se aleje o acerque el punto de vista, aparecerán diferentes líneas a modo de rejilla para facilitar el seguimiento de las coordenadas en el escenario. En la figura 6.12 se puede apreciar el resultado de una aproximación de la

cámara de visualización del escenario respecto del *zoom* inicial calculado para las dimensiones de los escenarios observados en las figuras 6.1 y 6.2.

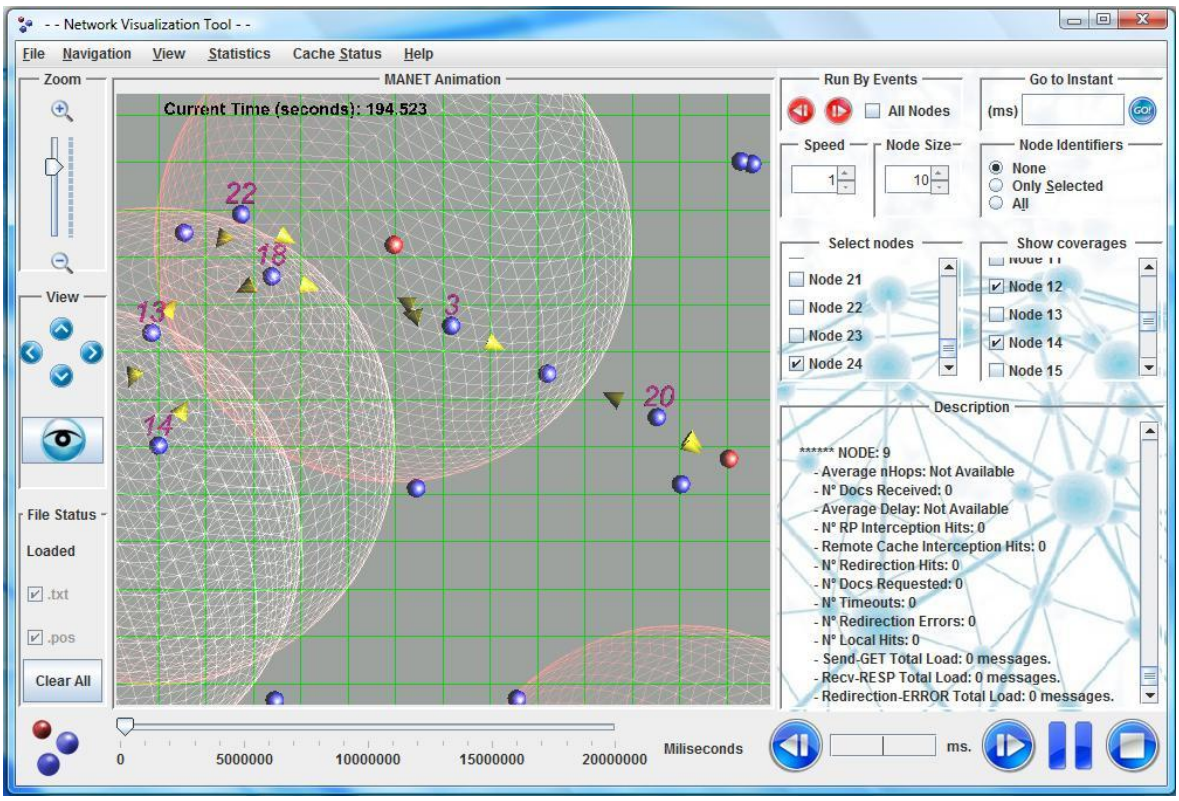


Figura 6.12. Acercamiento de la cámara de visualización hacia el escenario

Cuando se está proyectando una vista normal en la aplicación, el escenario es observado de forma que represente una vista en planta. En ese caso, las opciones del *zoom* modifican la coordenada *z* de la posición que ocupa la cámara, entendiendo que las coordenadas *x* e *y* representan el ancho y el alto del escenario, respectivamente. Cuando se modifica el punto de vista pasando a una vista rotada en las opciones del ajuste de la posición del escenario que se explicarán a continuación, los controles aquí presentados junto con el *scroll* del ratón modificarán la coordenada *y* que posee la cámara de visualización. De esta forma se conseguirá la aproximación o alejamiento del escenario natural que se busca a través de un *zoom*.

6.2.3.2. Ajuste de la posición de la vista del escenario

Además del acercamiento o alejamiento de la cámara de visualización del escenario virtual, existen otros movimientos que se pueden llevar a cabo. Así, para desplazar el punto de vista a la derecha, izquierda, arriba o abajo, se utilizarán los botones presentados en la parte superior de la figura 6.13, o bien a través de las flechas del teclado, siempre que previamente se haya posicionado el ratón en cualquier punto del escenario virtual. Además,

el software es capaz de generar un cambio de la perspectiva presentada a través de la opción de vista girada, que se puede activar seleccionando el botón con la imagen de un ojo que aparece en la parte inferior de la misma figura 6.13.

Figura 6.13. Controles de movimiento y rotación de la cámara de visualización

Este cambio de punto de vista se llevará a cabo a través de una animación durante unos cinco segundos. El resultado de este movimiento de rotación de la cámara de visualización se presenta en la figura 6.14.

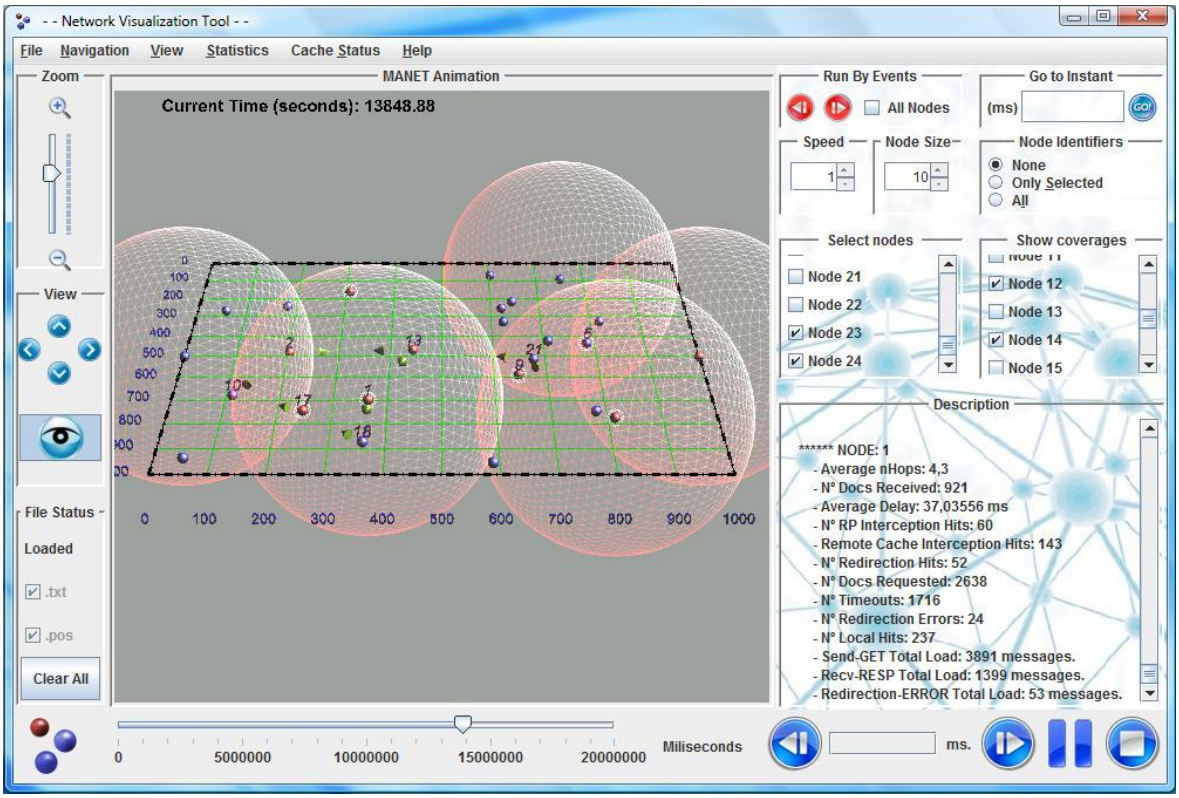


Figura 6.14. Vista rotada del escenario

Cuando la vista ofrecida por la aplicación es la vista normal o en planta, los movimientos llevados a cabo a través de los controles con las flechas indicadoras van a

modificar las coordenadas *x* e *y* que posee la cámara de visualización. De la misma forma que ocurre con los controles asignados para el *zoom*, cuando la vista mostrada haya sido rotada o girada, los controles del movimiento de la cámara hacia arriba, abajo, derecha e izquierda modificarán las coordenadas *x* y *z*. Es decir, el movimiento por el eje *x* permanece invariable, mientras que las flechas indicadoras de movimientos hacia arriba y abajo modificarán la coordenada *z*, generando el desplazamiento más afín al movimiento que muestran las flechas indicadoras de dichos controles.

6.2.4. PANEL DE CONTROL DE ARCHIVOS

La zona inferior del panel izquierdo posee dos funcionalidades diversas. Por un lado, muestra si los archivos de entrada de la aplicación han sido cargados. Por otro lado, realiza una limpieza de la información cargada de estos archivos.

6.2.4.1. Estado de los archivos cargados

Para determinar si se han seleccionado y cargado los datos de los archivos de entrada de la aplicación, se utiliza los indicadores mostrados en la figura 6.15. De esta forma, cuando se produzca la apertura de uno de los archivos de entrada, se marcará el indicador correspondiente.

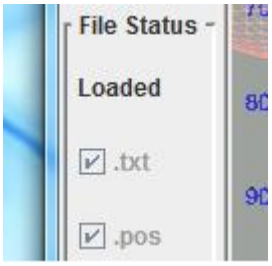


Figura 6.15. Estado de la carga de los archivos de entrada

Cuando se intenta abrir un archivo para el que ya se ha cargado anteriormente la información de otro archivo de la misma extensión, es decir, con el mismo tipo de información y finalidad, el software generará una ventana emergente avisando de ello.

6.2.4.2. Borrado de datos de la aplicación

Cuando se pretende comenzar con la creación de un nuevo escenario, o bien si nos hemos equivocado al seleccionar algún fichero de entrada para el que ya se ha cargado toda su información, se puede hacer un borrado de los datos que posee la aplicación. Para llevar a cabo esta acción, se debe pulsar el botón presentado en la figura 6.16.

Figura 6.16. Control destinado a limpiar los datos de la aplicación

En este caso, se modifica la ventana principal de forma que presenta el mismo aspecto que tras la carga inicial de la aplicación, deshabilitando la gran mayoría de controles, exceptuando el menú *File* para la selección de ficheros y el cierre de la aplicación, así como el control presentado en este apartado.

6.2.5. PANELES DE CONTROL DE LA TEMPORIZACIÓN DE LA ANIMACIÓN

En los paneles de control de la temporización de la animación se han incluido todas aquellas funcionalidades que modifican el avance natural de la animación, generando incluso saltos temporales en algunos casos.

6.2.5.1. Avance y retroceso por eventos

Para poder desplazarse por la animación en forma de saltos temporales ajustados a eventos concretos, se ha desarrollado la funcionalidad aquí presentada. Así, para poder centrarse en el estudio de los eventos que estén relacionados con un determinado o determinados nodos, se utilizan los controles presentados en la figura 6.17. Gracias a esta funcionalidad, se podrá avanzar o retroceder en la animación, pausando la misma únicamente en aquellos eventos que representen información acerca de la petición realizada por un nodo que se encuentre seleccionado en ese instante determinado. Si se desea avanzar y retroceder pausándose en todos los eventos de la animación, se deberá marca la opción "*All nodes*" o, lo que es lo mismo, para todos los nodos.

Figura 6.17. Controles para el avance y retroceso por eventos

Estos controles de avance y retroceso por eventos estarán deshabilitados mientras no exista un nodo seleccionado en el escenario y no se encuentre marcada la opción de desplazamiento para los eventos de todos los nodos de la red. Además, el software indicará si, para los nodos en cuestión, se ha alcanzado el último o el primero de los eventos.

Finalmente, los eventos en los que se posicione la animación serán mostrados en la zona dedicada a la descripción de eventos y estadísticos, en la zona inferior de este panel derecho, como se verá más adelante. Los eventos considerados podrán ser tanto eventos relacionados con el tráfico de la red (provenientes del archivo de salida de la simulación *.txt*) como eventos de cambios de dirección de los nodos (provenientes del archivo de movilidad de los nodos *.pos*).

6.2.5.2. Desplazamiento temporal absoluto de la animación (*"Go to"*)

Para posicionar la animación en un determinado instante de tiempo absoluto, ya sea anterior o posterior al instante actual de la animación, se utilizan los controles presentados en la figura 6.18.

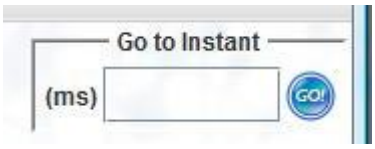


Figura 6.18. Control para el desplazamiento temporal absoluto de la animación

En la figura se observa una caja de texto donde introducir el instante temporal, que deberá venir dado en milisegundos. El software limitará la introducción de caracteres en dicha caja de texto a cifras decimales, así como el número de cifras decimales a introducir, utilizando para ello el máximo número de cifras de la duración de la simulación. Además, el software avisará si se ha introducido un instante de tiempo superior a la duración de la simulación, o bien si no se ha introducido ningún valor, dejando el campo vacío.

Para posicionar la aplicación en el instante introducido, se deberá pulsar el botón *"Go"*.

6.2.5.3. Barra de desplazamiento horizontal o *slider* temporal

En la ventana principal de la aplicación, existe una barra de desplazamiento con un ítem que avanza a medida que la animación va avanzando, representando además en sus extremos el instante inicial y final de la misma. Este ítem representará en todo momento el instante actual de la animación, siempre en milisegundos. Si se desea avanzar o retroceder por la animación, se puede desplazar el ítem hacia el instante aproximado buscado.

Este *slider* temporal se muestra en la figura 6.19.

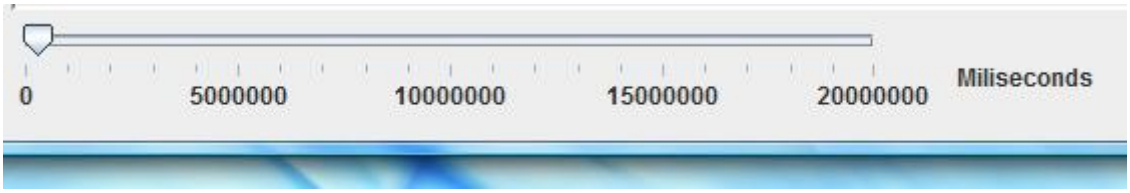


Figura 6.19. Barra de desplazamiento horizontal o *slider* temporal

6.2.5.4. Avance y retroceso por saltos temporales

Además del avance y retroceso por eventos, el software ofrece la posibilidad de desplazarse por la animación gracias a saltos temporales relativos al instante actual. Los controles necesarios para realizar esta acción se muestran en la figura 6.20.

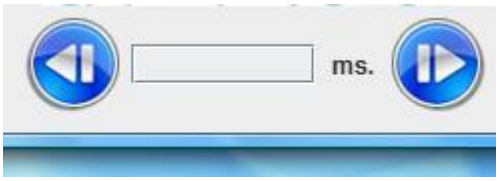


Figura 6.20. Control para el desplazamiento relativo por saltos temporales de la animación

Para introducir el salto temporal a realizar se utiliza la caja de texto central, expresando la cantidad en milisegundos. De la misma forma que sucede para la caja de texto utilizada por la opción de posicionamiento absoluto o "*Go to*", sólo se podrán introducir cifras decimales. Además, se limitará la cantidad de cifras a introducir a la cantidad de cifras que posee la duración de la simulación.

El software informa a través de ventanas emergentes de aquellos desplazamientos que no se puedan llevar a cabo, como avanzar más allá del final de la animación, o retroceder a un instante anterior al instante inicial de la misma.

Cada vez que se produce un salto temporal, ya sea de forma absoluta o de forma relativa, se mostrará un resumen con los estadísticos de los nodos que se encuentren seleccionados en ese momento en la zona destinada a tal fin.

6.2.5.5. Reanudación, pausa y parada de la aplicación

Para controlar la activación, pausa o parada de la animación, se utilizan los botones mostrados en las figuras 6.21 y 6.22. La diferencia entre ambas figuras radica en la imagen mostrada en el botón izquierdo. Así, cuando la animación se encuentre activa, la imagen indicará la clásica figura de pausa de la animación o *pause* (figura 6.21), mientras que si la

animación se encuentra pausada, la imagen indicará la clásica figura de activación de la animación o *play* (figura 6.22).

Figura 6.21. Pausa o parada de la animación

Figura 6.22. Reanudación y parada de la animación

6.2.6. PANELES DE VISUALIZACIÓN DE LA ANIMACIÓN

Las funciones relacionadas con la visualización de la animación sirven para modificar la velocidad a la que se está reproduciendo la animación y el valor del radio de los nodos y para mostrar los identificadores numéricos de los nodos.

6.2.6.1. Modificación de la velocidad de la animación y del tamaño del radio de los nodos

A continuación, se presentan dos controles cuyo funcionamiento es muy similar, aunque persiguen dos finalidades diferentes. Estos controles vienen representados en la figura 6.23.

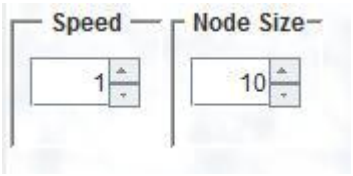


Figura 6.23. Controles para la modificación de la velocidad de visualización y radio de los nodos

Para modificar la velocidad de la animación respecto de la velocidad base de la simulación utilizada para la creación de los archivos de entrada, se utiliza el control denominado "*Speed*". Los valores máximo y mínimo de este ratio de velocidades son 10 y 0.1, respectivamente. Así, en el caso de seleccionar un ratio de valor 10, representará una

velocidad de la animación diez veces superior a la velocidad de base, y el caso de un ratio de valor 0.1, la animación avanzará diez veces más ralentizada que la velocidad de base.

De igual forma, para modificar el radio de las esferas que representan los nodos, se utiliza el control bajo el nombre "*Node Size*". Los valores posibles para este radio, representados en metros, son desde 1 a 35.

6.2.6.2. Muestra de los identificadores de nodos

Una de las maneras de especificar para qué tipo de nodos se desea mostrar sus identificadores es a través de los controles presentados en la figura 6.24. Los diferentes grupos de nodos definidos en esos controles son:

- *"None"*: no se mostrará el identificador para ningún nodo.
- *"Only Selected"*: sólo se mostrarán los identificadores de aquellos nodos que se encuentren seleccionados.
- *"All"*: se mostrarán los identificadores de todos los nodos presentes en la red.

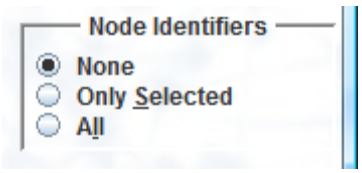


Figura 6.24. Control para la muestra de los identificadores de nodo

Estas opciones son excluyentes entre sí, por lo que será imposible elegir más de una. Es importante discernir visualmente entre los nodos que se encuentren seleccionados y aquellos que no lo estén, como se verá más adelante. Siguiendo con esta misma idea, los identificadores de los nodos que se encuentren seleccionados se mostrarán en color rojo, mientras que aquellos que no lo estén se mostrarán en tono negro, pero siempre en ambos casos sólo cuando proceda.

6.2.7. PANELES DE SELECCIÓN

A través de los paneles de selección, se van a poder visualizar las coberturas de los nodos y elegir aquellos nodos de los cuales se desea extraer información.

6.2.7.1. Selección de nodos

Para seleccionar un nodo, se podrá elegir del listado que presenta el control mostrado en la figura 6.25 denominado "*Select nodes*".

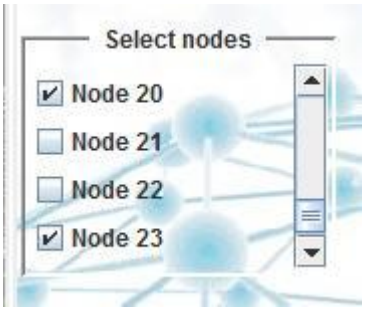


Figura 6.25. Control para la selección de nodos

Una segunda opción será clicar sobre el nodo en particular con el botón izquierdo del ratón, reflejando esta selección en el mismo listado de la figura. Una vez que el nodo se encuentre seleccionado, la esfera que representa el nodo cambiará a tonos rojos, mientras que aquellos que no se encuentren marcados presentarán tonos azules.

La selección de nodos es fundamental para el comportamiento de la aplicación. Los nodos de los que se pretenda obtener información, deberán ser seleccionados. Así, cuando la animación se encuentre activa, cada vez que se produzca un evento que actualice los valores estadísticos de alguno de los nodos seleccionados, se presentará el valor de todos los estadísticos del nodo en cuestión además del evento que ha generado el cambio de estadísticos en la zona habilitada para las descripciones (en la zona inferior del panel derecho de la ventana principal). Además, los nodos seleccionados serán tenidos en cuenta para el avance y retroceso por eventos cuando no deba realizarse para todos los nodos. El concepto de nodo seleccionado va aún más allá. Cuando un nodo se encuentre seleccionado y la animación se encuentre activa, sin producirse el avance y retroceso por eventos, los mensajes que se generen en dicho nodo y en otros nodos de la red debido a la petición de un documento realizada por él, serán representados visualmente. Así, los mensajes enviados por la red serán representados con pequeñas animaciones en forma de conos desplazándose con la dirección y el sentido de la transmisión del mensaje. Hay que considerar también que estas animaciones se podrán producir al avanzar y retroceder por eventos para todos los nodos del escenario. Además de estas animaciones, ya sea con la animación activa o con el avance y retroceso por eventos, los nodos que intervengan en el mecanismo de petición-respuesta originado por un nodo que se encuentre seleccionado presentarán unas esferas semitransparentes en forma de halo que permanecerán visibles hasta que el nodo iniciador en cuestión reciba el documento satisfactoriamente, o bien se produzca un *timeout*.

En el mismo listado de nodos, la primera opción va destinada a la selección o deselección de todos los nodos a la vez, opción bajo el nombre "*Select All Nodes…*".

6.2.7.2. Selección de coberturas de los nodos

De la misma forma que se seleccionan los nodos, existe un control destinado a la selección de aquellas coberturas que se deseen visualizar. El control que ofrece el listado con todos los nodos del escenario cuyas coberturas se pueden visualizar se muestra en la figura 6.26.

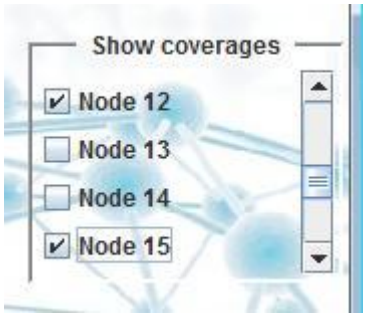


Figura 6.26. Control para la selección de coberturas de los nodos

También se presenta como primera opción del listado la posibilidad de mostrar las coberturas de todos los nodos a la vez, o bien ocultarlas todas.

Existe otra manera para visualizar u ocultar la cobertura de un determinado nodo, y es a través de una de las opciones que ofrece el menú contextual asociado a cada nodo de la red.

6.2.8. PANEL DE INFORMACIÓN

Para la descripción de los eventos que están sucediendo y los resúmenes de los valores estadísticos que sean requeridos, además de la información de las coordenadas en el escenario real de los nodos de la red, se ha destinado la zona inferior del panel derecho de la ventana principal de la aplicación. Esta zona fundamental de la ventana principal se encuentra etiquetada con "*Description*".

Hay que diferenciar los casos en los que se mostrará la descripción de algunos eventos y los estadísticos de los nodos:

- Cuando la animación se encuentre activa, cada vez que se produzca un evento que modifique alguno de los estadísticos de un nodo que se encuentre seleccionado, se mostrará un resumen con todos sus estadísticos, además del propio evento que ha generado el cambio de estadísticos.

- Cuando la animación se encuentre pausada y se esté avanzando o retrocediendo por eventos, cada vez que se posicione en un determinado evento, ya sea para los nodos seleccionados o para todos los nodos de la red, se mostrará la descripción de dicho evento. Además, como en el caso anterior, si este evento desencadena el cambio de alguno de los valores estadísticos, se mostrará un resumen con todos los estadísticos del nodo en particular. Cuando se avanza y retrocede por eventos y se produce la petición de un documento por un nodo, se genera una animación en forma de radiación en dicho nodo durante unos instantes, como se observa en la figura 6.27. En la misma figura se puede apreciar además el aspecto de los nodos seleccionados, de los no seleccionados, y los halos generados.

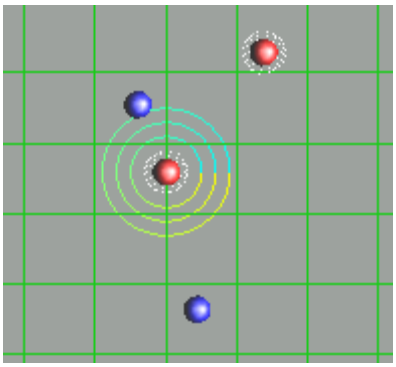


Figura 6.27. Muestra de la radiación alrededor de un nodo

- Cuando se produzca un desplazamiento del instante de la animación motivado por saltos temporales relativos al instante actual o bien por posicionamiento absoluto en un determinado instante de tiempo, se mostrará un resumen de los estadísticos de todos aquellos nodos que se encuentre seleccionados.

La aplicación detecta el instante a partir del cual se comienzan a calcular los estadísticos para un nodo. Este instante vendrá fijado por el mensaje *reset* de estadísticos explicado en el apartado A.2.3.9 del Apéndice A. Cuando los estadísticos de un nodo aún no son válidos, se añade la etiqueta "*Warming Node*" antes del valor de los estadísticos, como se muestra en la figura 6.28. En este caso, además de mostrar la etiqueta anterior, presentará un valor nulo en aquellos estadísticos que representen un número total de elementos, como por ejemplo el número total de documentos solicitados por un nodo o el número total de documentos recibidos. Para el resto de estadísticos que representen una media, como el número medio de saltos o el retardo medio, o bien que representen un ratio, no aparecerá un valor nulo, sino que vendrá representado por "*Not Available*", es decir, será un valor no disponible.

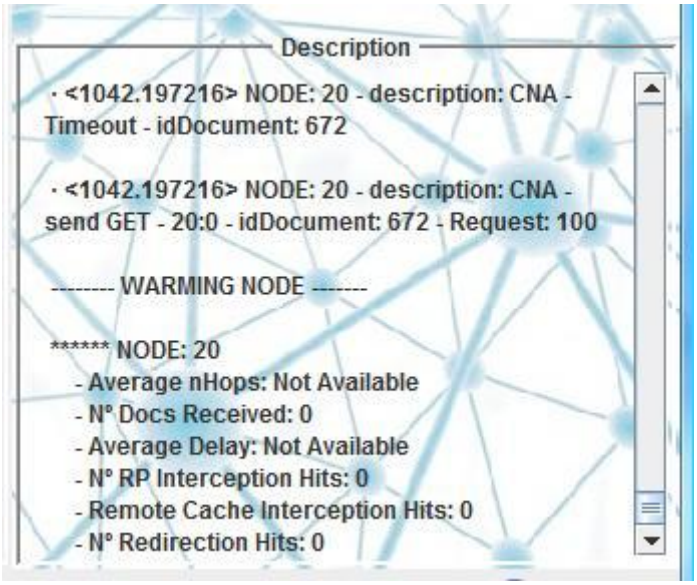


Figura 6.28. Muestra de estadísticos durante el calentamiento de la caché local

Un típico resumen de estadísticos de un determinado nodo se muestra en la figura 6.29.

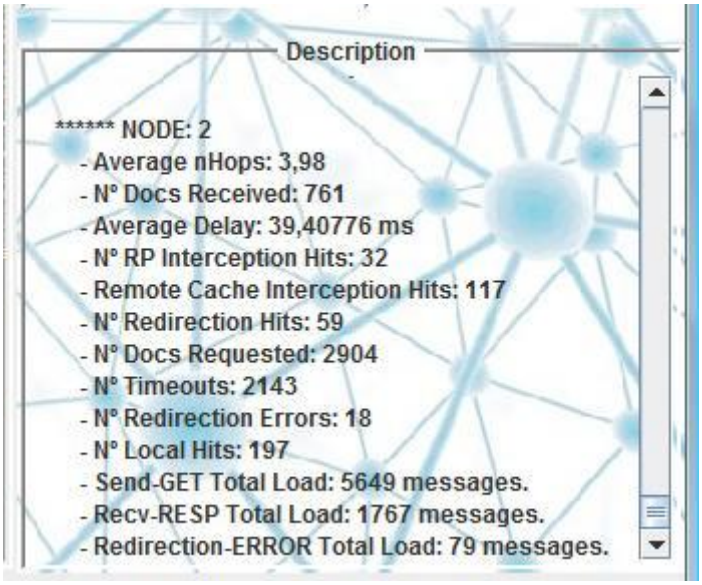


Figura 6.29. Muestra de estadísticos

6.2.9. LOGO DE LA APLICACIÓN

El logo de la aplicación está presente en la ventana principal de la aplicación, concretamente en la esquina inferior izquierda, como el aspecto mostrado en la figura 6.30.

Figura 6.30. Logo de la aplicación en la ventana principal

6.2.10. BARRA DE MENÚS

La barra de menús de la ventana principal de la aplicación presenta una gran cantidad de opciones, aunque muchas de ellas ofrecen funcionalidades que ya han sido presentadas anteriormente por realizar las mismas actividades que algunos de los controles distribuidos por la ventana. A continuación nos vamos a centrar en cada uno de sus menús.

6.2.10.1. Menú 1: *File*

El primer menú de la ventana ofrece las opciones de carga de archivos, ya comentada anteriormente, y de cierre de la aplicación, como se observa en la figura 6.31. Este menú viene identificado con el nombre *File*.

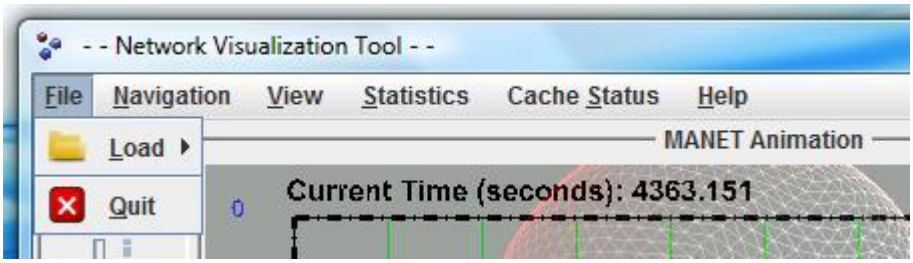


Figura 6.31. Menú *File*

6.2.10.2. Menú 2: *Navigation*

El segundo menú de la barra de menús de la ventana principal, denominado *Navigation*, representa las acciones relativas a la navegación de la animación explicadas en apartados anteriores. Además, se pueden identificar fácilmente por los iconos asociados a cada una de esas opciones. Estas opciones pueden observarse en las figuras 6.32 y 6.33.

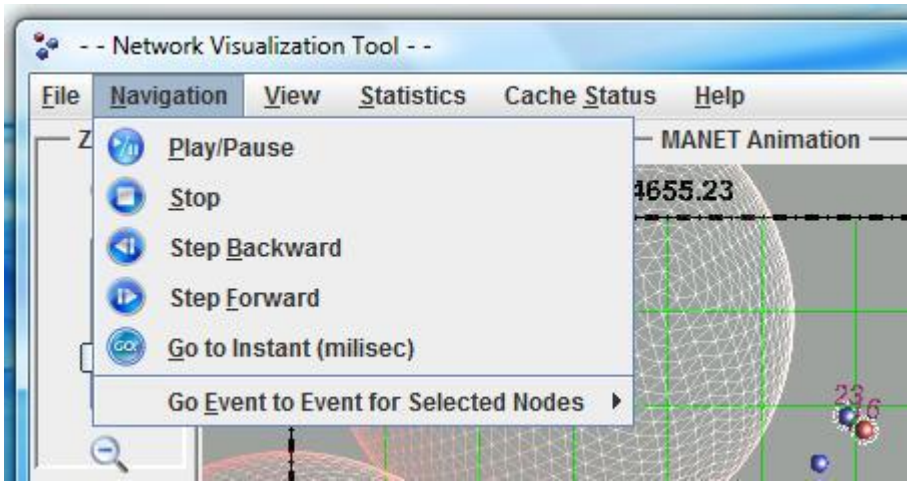


Figura 6.32. Menú *Navigation*

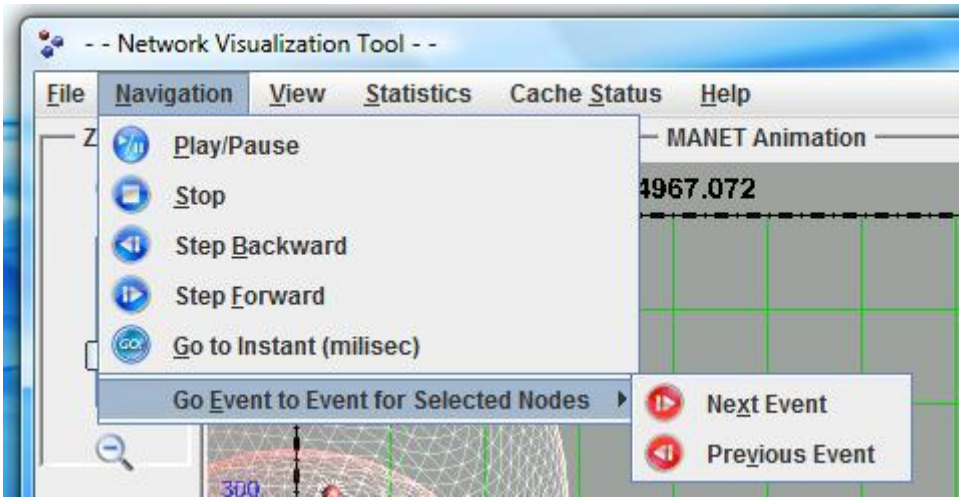


Figura 6.33. Menú *Navigation*: Submenús para avance y retroceso por eventos

6.2.10.3. Menú 3: *View*

De la misma forma que ocurre con las opciones de la visualización de la animación, las diferentes funcionalidades ofrecidas relativas a la vista del escenario gráfico, es decir, al desplazamiento de la cámara de visualización, vuelven a recogerse bajo el menú *View*.

Las diferentes opciones globales que se ofrecen vienen representadas en la figura 6.34.

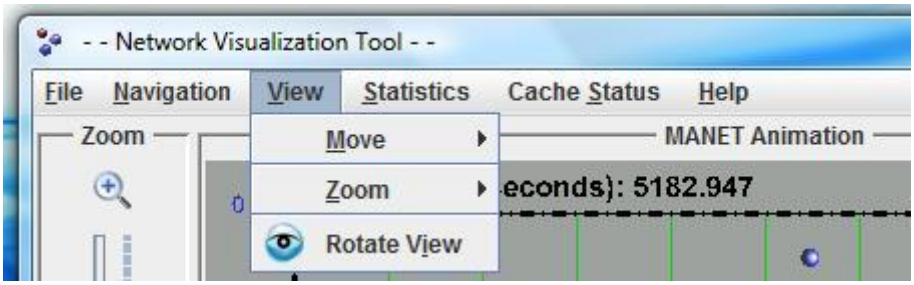


Figura 6.34. Menú View

Las dos primeras opciones globales vienen desarrolladas en las figuras 6.35 y 6.36. En la figura 6.35 se definen las opciones de movimiento por el escenario englobadas por el grupo "*Move*", mientras que en la figura 6.36 vienen explicitadas las opciones del *zoom* englobadas por el grupo "*Zoom*".

La última opción se encarga de generar una animación para la rotación de la vista. Esta rotación implica pasar a una vista girada cuando la animación se está mostrando desde una vista en planta, o bien pasar a una vista en planta si ésta se encuentra girada.

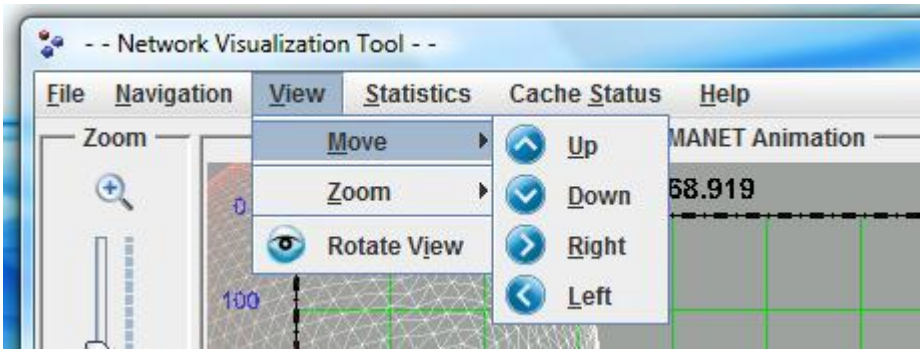


Figura 6.35. Menú *View*: Movimiento de la cámara de visualización de la animación

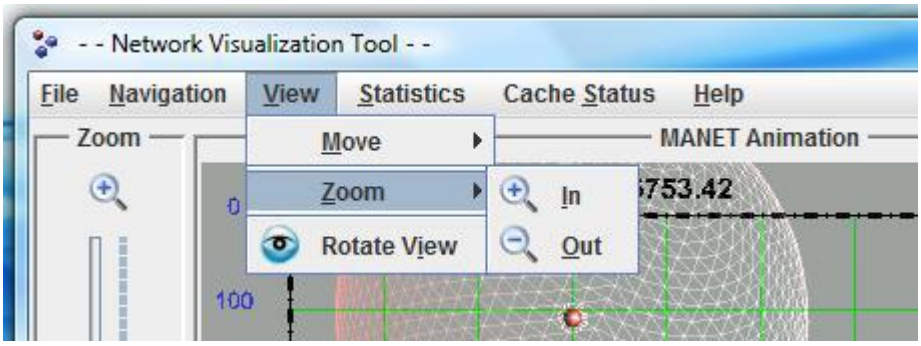


Figura 6.36. Menú *View*: Controles del *zoom*

6.2.10.4. Menú 4: *Statistics*

Una de las funcionalidades principales de esta aplicación es el cálculo de estadísticos o parámetros de estudio de la red, cuya definición ha sido introducida en el comienzo del presente capítulo. El menú "*Statistics*" contiene una única opción, que representa la creación del informe en PDF con datos válidos o alineados con el instante actual de la animación, identificada con el nombre "*Create Statistics Report at current instant*", como se puede observar en la figura 6.37.

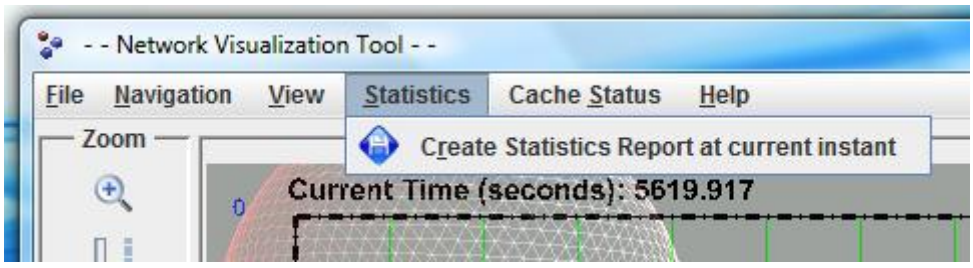


Figura 6.37. Menú *Statistics*

Una vez que se activa el control de creación del informe, el software toma el instante actual de la animación e indica, a través de una ventana de información, que se está recopilando toda la información necesaria y generando el informe PDF, como se muestra en la figura 6.38.

Figura 6.38. Ventana de información durante la creación del informe PDF

Si se produce algún error durante la creación del informe, el software avisará mediante una ventana emergente.

El documento PDF generado se almacenará en el directorio actual desde donde se ejecuta la aplicación, tendrá el nombre prefijado "*statistics.pdf*" y presenta la siguiente estructura:

- Portada, donde se incluye la siguiente información: nombre del archivo de salida de la simulación *.txt*, nombre del archivo de movilidad de los nodos *.pos*, la duración de la simulación, el número total de nodos, las dimensiones del escenario real en metros, el radio de las coberturas de los nodos, y el instante en el cual se ha solicitado la creación del informe y para el que son válidos los datos estadísticos recogidos. El aspecto de la portada se observa en la figura 6.39.

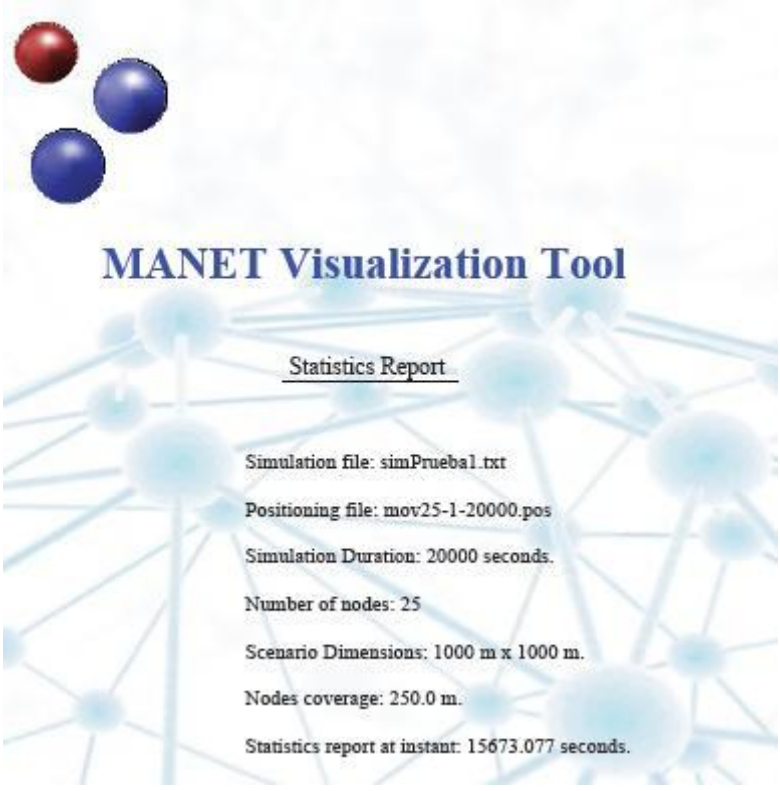


Figura 6.39. Portada del informe PDF generado

- Primer Apartado, donde se presentan los estadísticos estudiados en la aplicación para cada nodo de forma extendida, tal y como se aprecia en la figura 6.40.

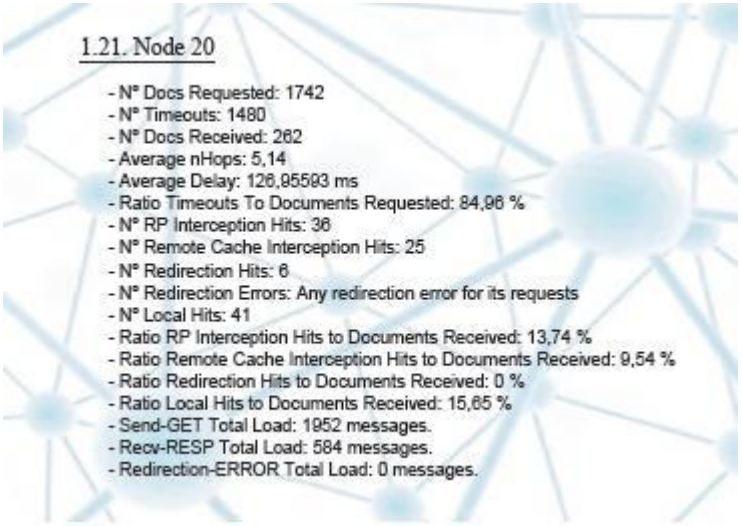


Figura 6.40. Información detallada de los estadísticos de cada nodo en el informe PDF generado

- Segundo apartado, donde se añade un histograma para cada estadístico estudiado a modo de comparación entre todos los nodos de la simulación. Un ejemplo de uno de los histogramas generados (en concreto, para el número total de peticiones de documentos realizadas por los nodos) se encuentra en la figura 6.41.

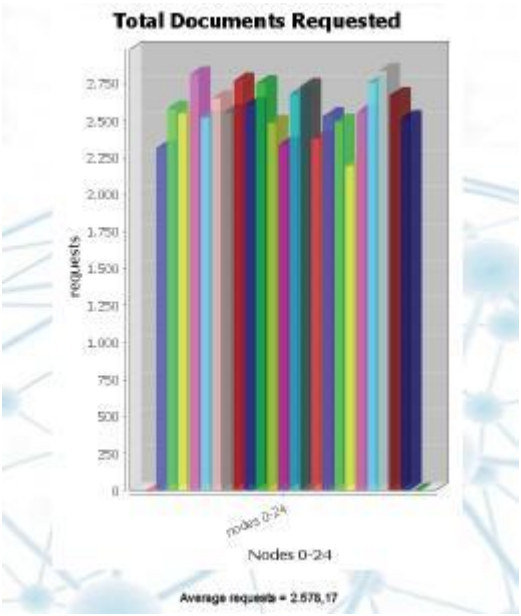
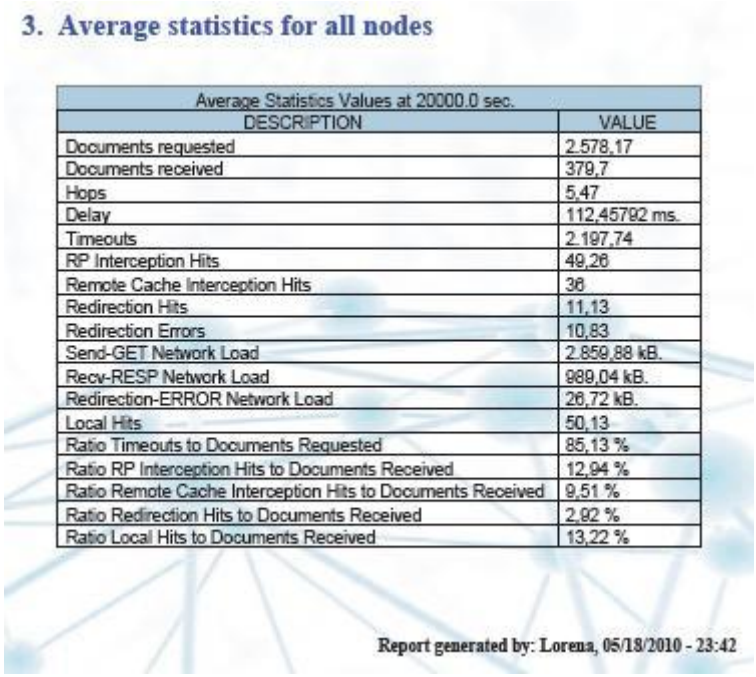


Figura 6.41. Aspecto de los histogramas del informe PDF generado

- Tercer apartado, donde se recoge el valor medio para cada uno de los estadísticos bajo estudio realizando la media entre todos los nodos, como se muestra en la figura 6.42.



3. Average statistics for all nodes

Average Statistics Values at 20000.0 sec.	
DESCRIPTION	VALUE
Documents requested	2.578,17
Documents received	379,7
Hops	5,47
Delay	112,45792 ms.
Timeouts	2.197,74
RP Interception Hits	49,26
Remote Cache Interception Hits	36
Redirection Hits	11,13
Redirection Errors	10,83
Send-GET Network Load	2.859,88 kB.
Recv-RESP Network Load	989,04 kB.
Redirection-ERROR Network Load	26,72 kB.
Local Hits	50,13
Ratio Timeouts to Documents Requested	85,13 %
Ratio RP Interception Hits to Documents Received	12,94 %
Ratio Remote Cache Interception Hits to Documents Received	9,51 %
Ratio Redirection Hits to Documents Received	2,92 %
Ratio Local Hits to Documents Received	13,22 %

Report generated by: Lorena, 05/18/2010 - 23:42

Figura 6.42. Cuadro resumen con el valor medio para todos estadísticos del informe PDF generado

6.2.10.5. Menú 5: *Cache Status*

El penúltimo menú representa el subgrupo de funcionalidades agrupadas por el nombre "*Cache Status*", a través de las que se puede obtener el contenido de las cachés local y de redirecciones para un determinado nodo. Como se verá más adelante, existe otra opción para mostrar esta información, y es con una de las opciones del menú contextual que está asociado a cada nodo de la red. Las opciones de este menú se muestran en la figura 6.43.

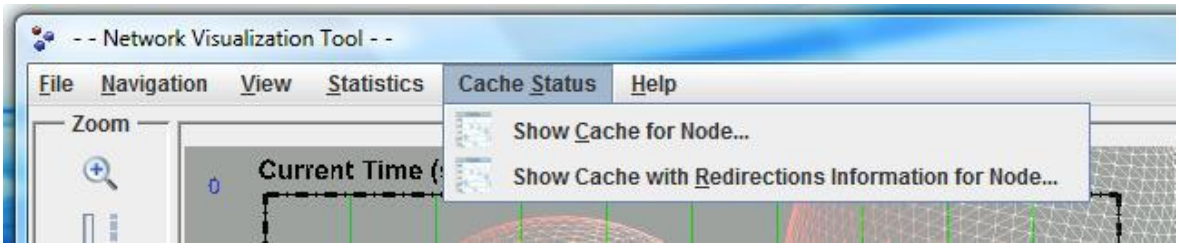


Figura 6.43. Menú Cache Status

A través de la primera opción, se solicita la visualización del contenido de la caché local de un determinado nodo, nodo que es indicado gracias a la ventana emergente de selección de la caché local a mostrar, como aparece en la figura 6.44. En este punto, también se podría decidir no visualizar ninguna caché, a través del botón "*Cancel*".

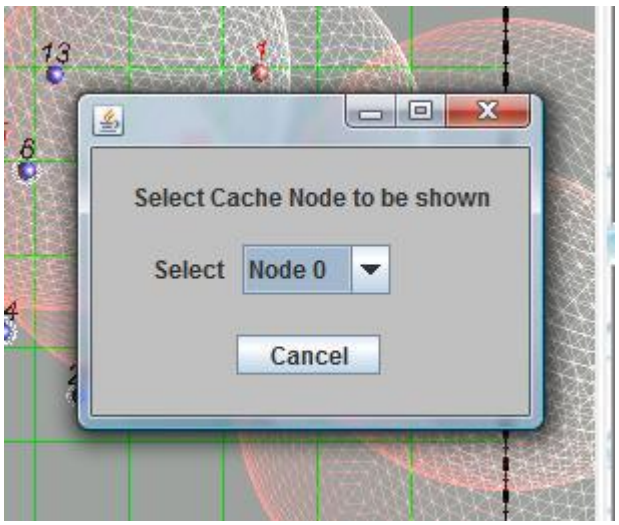


Figura 6.44. Selección del nodo cuya caché local se pretende mostrar

Una vez que se ha seleccionado un nodo, se genera una nueva ventana con el contenido de su caché local, organizando sus diversas entradas en una tabla, como se representa en la figura 6.45. El contenido de la caché local, así como el contenido de la caché de redirecciones, estará alineado con el instante actual de la animación. Este instante de tiempo, así como el nodo que se está consultando, aparecen en el borde superior de la ventana. El número total de documentos almacenados, así como el número de documentos expirados y el número para los que se tiene información del instante de expiración de valor nulo (situación que no da lugar en el protocolo utilizado), aparecen mostrados en la parte inferior de la ventana.

Se puede realizar la consulta de un determinado documento en caché a través de su identificador, que deberá introducirse en la caja de texto.

La tabla se encuentra ordenada por las posiciones de memoria, en orden ascendente, donde cada una de las filas representa una entrada de memoria diferente. Los datos que se representan en las diversas columnas son las siguientes:

- "*Position*": Posición de la entrada de caché mostrada.
- "*Id*": Identificador numérico del documento almacenado en esa posición de la caché local.
- "*Size (bytes)*": Tamaño en *bytes* del documento almacenado en esa posición de memoria.
- "*Frequency*": Número de veces que ha sido accedido este documento en caché.
- "*Cost*": Valoración realizada por determinadas políticas de reemplazo, que se utilizan para la ordenación de los documentos en caché.

- "*Expiration*": Instante a partir del cual el documento se convierte en obsoleto. Cuando este instante sea inferior al instante de la animación actual, entonces para dichas entradas se mostrará el indicativo "*Expired*".

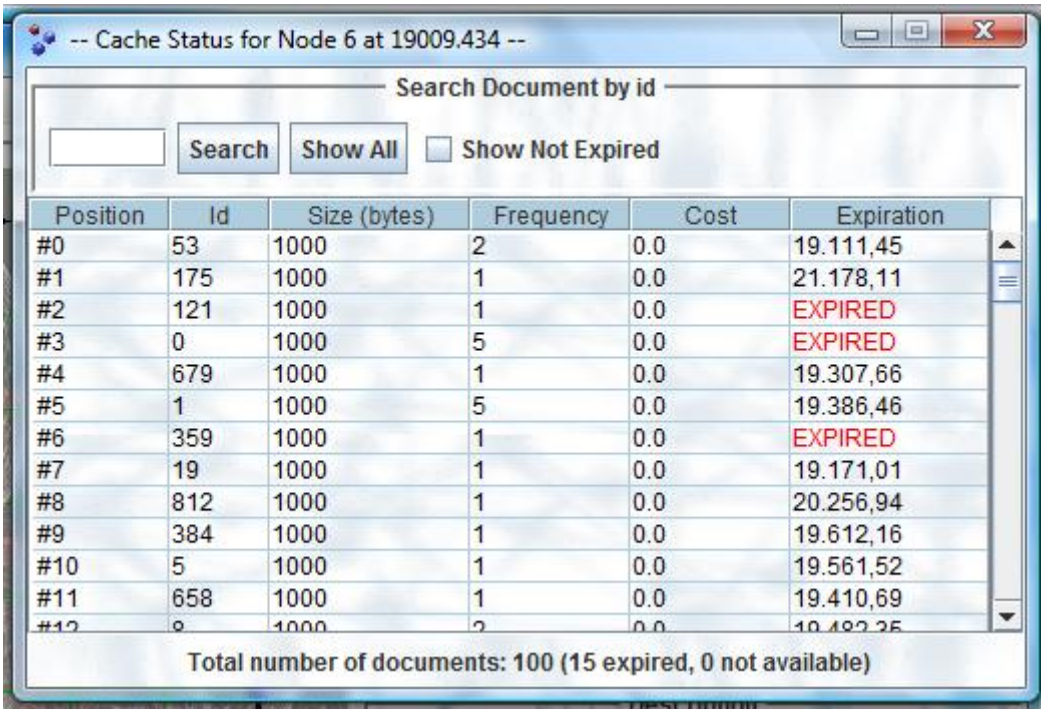


Figura 6.45. Contenido de una determinada caché local

Cuando se procede a buscar un documento en la caché, se introduce su identificador en la caja de texto. Si el resultado es exitoso, es decir, se encuentra el documento en la caché, se mostrará únicamente la entrada correspondiente, como se observa en la figura 6.46.

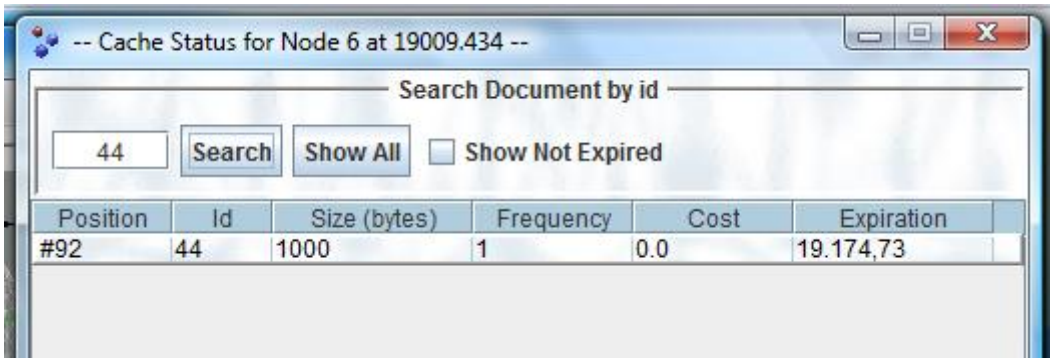


Figura 6.46. Documento encontrado en la caché local

Si por el contrario, el documento no se encuentra en la caché, se indicará con el mensaje "*Entry not found*", como queda reflejado en la figura 6.47.

Figura 6.47. Documento no encontrado en la caché local

Los controles que aparecen en la zona superior de la ventana, denominados "*Show All*" y "*Show Not Expired*", se encargan de mostrar todo el contenido de la caché local o únicamente aquellas entradas que contienen documentos no expirados, respectivamente.

En el caso de que se presione el botón "*Search*" para realizar la búsqueda de un documento en la caché pero no se haya introducido ningún valor en la caja de texto, se indicará con el mensaje "*Empty field*", como ocurre en la figura 6.48. La caja de texto sólo permitirá la introducción de cifras decimales.

Figura 6.48. Campo vacío en la búsqueda de un documento en la caché local

Con respecto a la consulta de la caché de redirecciones de un determinado nodo para el instante actual de la animación, se va a proceder de manera similar. Así, el primer paso es seleccionar el nodo del que queremos obtener esta información, gracias al menú desplegable que ofrece el software, que se muestra en la figura 6.49.

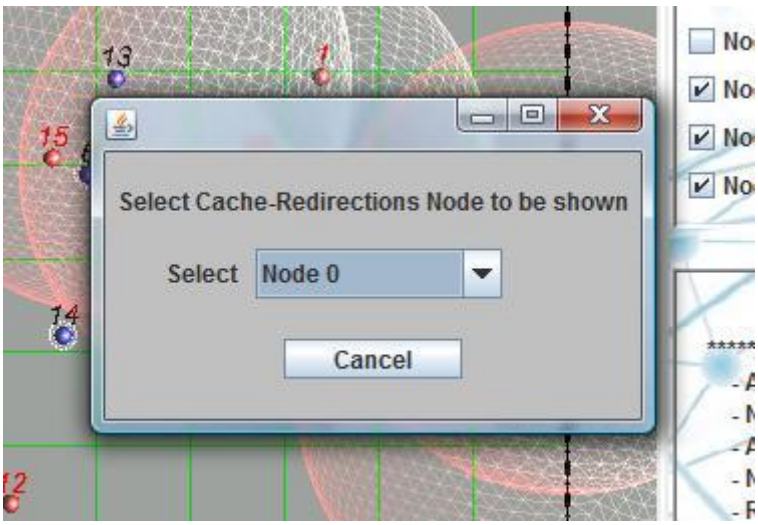


Figura 6.49. Selección del nodo cuya caché de redirecciones se pretende mostrar

Al igual que antes, en el borde superior de la ventana generada con el contenido de la caché de redirecciones aparece el nodo que se está consultando y el instante actual, instante para el que es válida la información mostrada.

En la parte inferior, vuelve a aparecer el número total de documentos que contiene la caché de redirecciones, indicando cuántos de ellos tienen información caducada (que vendrán representados con el indicativo "*Expired*"), y cuántos de ellos poseen una información acerca del instante de expiración igual a 0.0 (que vendrán representados con el indicativo "*Not Available*"). En este caso, sí aparecerán con cierta frecuencia entradas que no tengan información disponible del instante de expiración, debidos al propio funcionamiento del protocolo.

El contenido típico de una determinada caché de redirecciones es representado en la figura 6.50. En ella se observa, como para el caso de la consulta de caché local, una opción de búsqueda, que representa la búsqueda de la información de posición en la red de un determinado documento. Además, cada una de las entradas de esta memoria pueden estar representadas por una o dos filas, en función de si para ese documento se posee información de un solo registro o de los dos registros, GET y RESP.

La información que se obtiene de cada uno de los registros es la siguiente:

- "*Document ID*": identificador numérico del documento para el que se posee la información de su posición en la red.
- "*Register*": Tipo de registro cuya información viene representada en la fila actual, es decir, GET o RESP.

- "*Node Address*": Dirección o identificador del nodo que posee una copia válida del documento en su caché local.
- "*Distance (hops)*": Distancia, medida en saltos, a la que se encuentra la copia válida del documento cuya información está siendo representada en la actual entrada de caché de redirecciones.
- "*Expiration*": Instante a partir del cual la información acerca de la posición del documento se convierte en obsoleta. Cuando este instante sea inferior al instante de la animación actual, entonces para dichas entradas, como se ha comentado anteriormente, se mostrará el indicativo "*Expired*". Y si la información que se tiene almacenada acerca del instante de expiración equivale al valor 0.0, se mostrará el indicativo "*Not Available*".

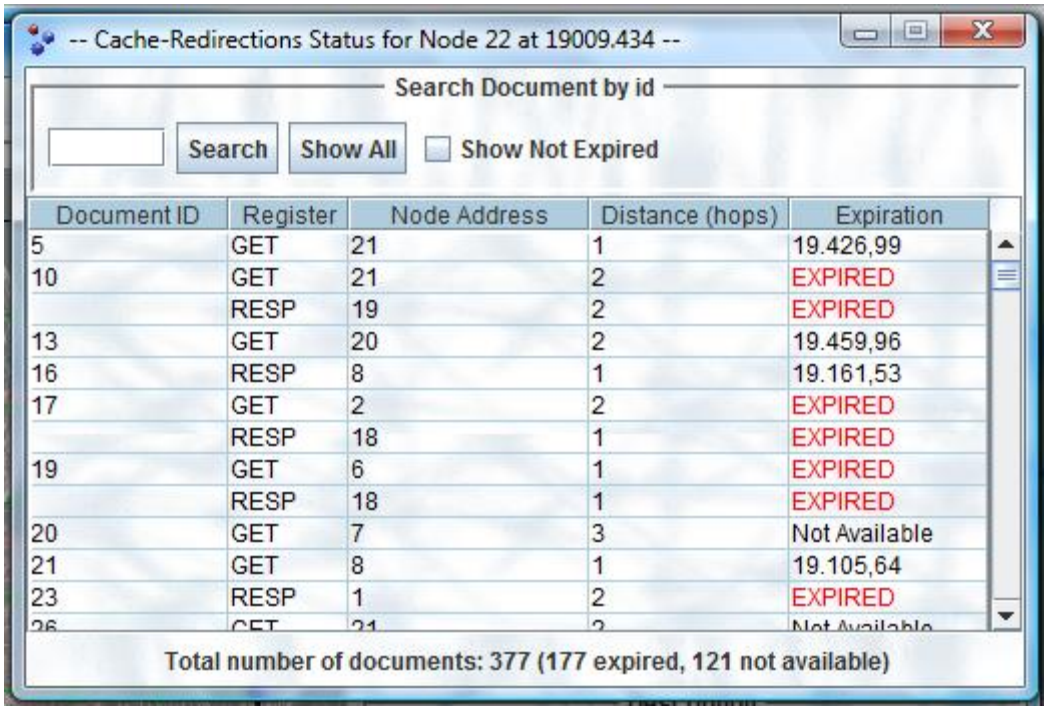


Figura 6.50. Contenido de una determinada caché de redirecciones

Como ocurría para las cachés locales, si se pretende visualizar únicamente aquellas entradas para las que, al menos, uno de los registros contiene información no obsoleta, entonces se elegirá el control "*Show Not Expired*", como se muestra en la figura 6.51.

Finalmente, para realizar la búsqueda de la información de la posición de un documento en la red, se introduce el identificador numérico del documento y se procede con el botón "*Search*". Si se encuentra información, se mostrará únicamente la entrada correspondiente como se observa en la figura 6.52, y si no, se indicará de la misma forma que para la caché local.

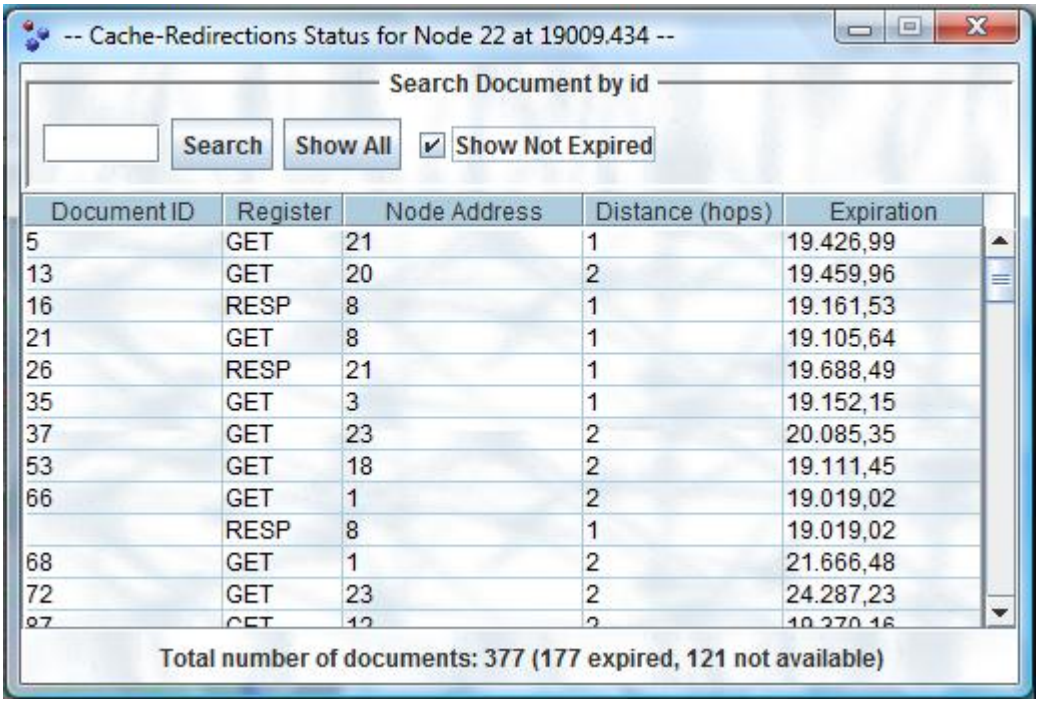


Figura 6.51. Documentos no expirados de una determinada caché de redirecciones

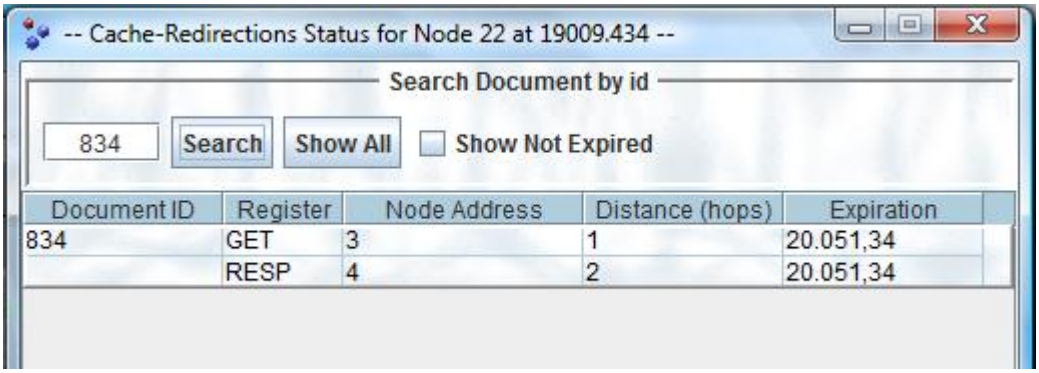


Figura 6.52. Documento encontrado en la caché de redirecciones

6.2.10.6. Menú 6: *Help*

El último menú, denominado "*Help*", presenta la opción para la visualización de información de la aplicación, como se muestra en la figura 6.53.

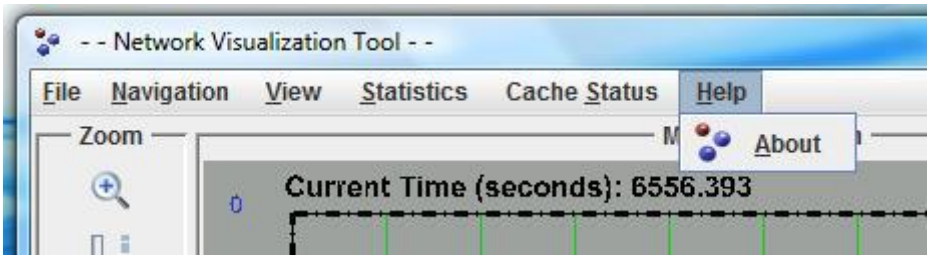


Figura 6.53. Menú *Help*

Al seleccionar la opción "*About*", se va a generar una ventana (figura 6.54), en la que la información mostrada contiene el nombre del departamento para el que se está realizando este proyecto, así como los nombres del tutor y de la autora del mismo.

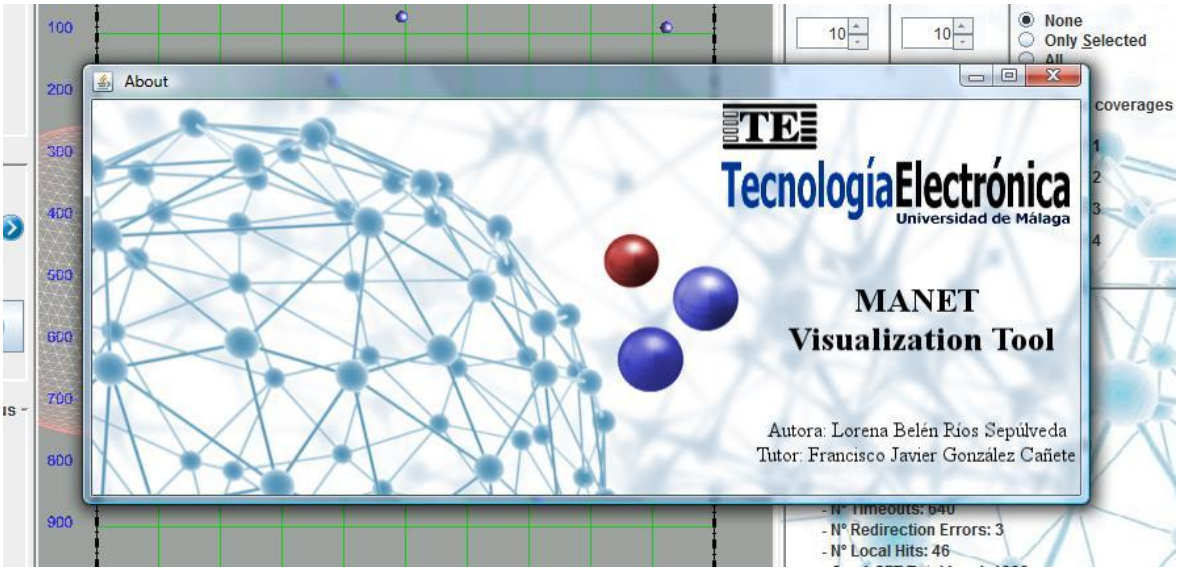


Figura 6.54. Menú *Help*: *About*

6.2.11. PANEL CENTRAL: ESCENARIO

El escenario es la parte fundamental de la aplicación, y por ello se ha decidido situar en la parte central de la ventana principal de la misma. En esta zona gráfica de la ventana, además de mostrar los nodos, sus coberturas, sus identificadores numéricos, sus halos, los mensajes enviados por la red, el instante actual de la animación, etc., se puede obtener información importante relativa a un nodo en particular, gracias al menú contextual, menú que aparece en la figura 6.55.

Para mostrar dicho menú, es necesario clicar con el botón derecho del ratón encima de la esfera que representa el nodo para el que se quiere obtener información. Esta información se podrá obtener independientemente de que sea un nodo que se encuentra seleccionado en dicho instante de tiempo o no.

Las diferentes opciones que aparecen en este menú contextual son:

- "*Show Statistics*": Muestra los estadísticos de ese nodo en el instante actual de la animación.
- "*Show Cache*": Muestra el contenido de la caché local de ese nodo en el instante actual de la animación.

- "*Show Redirection Cache*": Muestra del contenido de la caché de redirecciones de ese nodo en el instante actual de la animación.
- "*Show Coordinates*": Muestra las coordenadas en el escenario real que ocupa el nodo en el instante actual de la animación.
- "*Show Node Identifier*": Muestra el identificador del nodo por unos instantes si éste no está siendo mostrado.
- "*Show Coverage for Node*": Hace visible la cobertura del nodo, si no se estaba mostrando, o la oculta, en caso contrario.

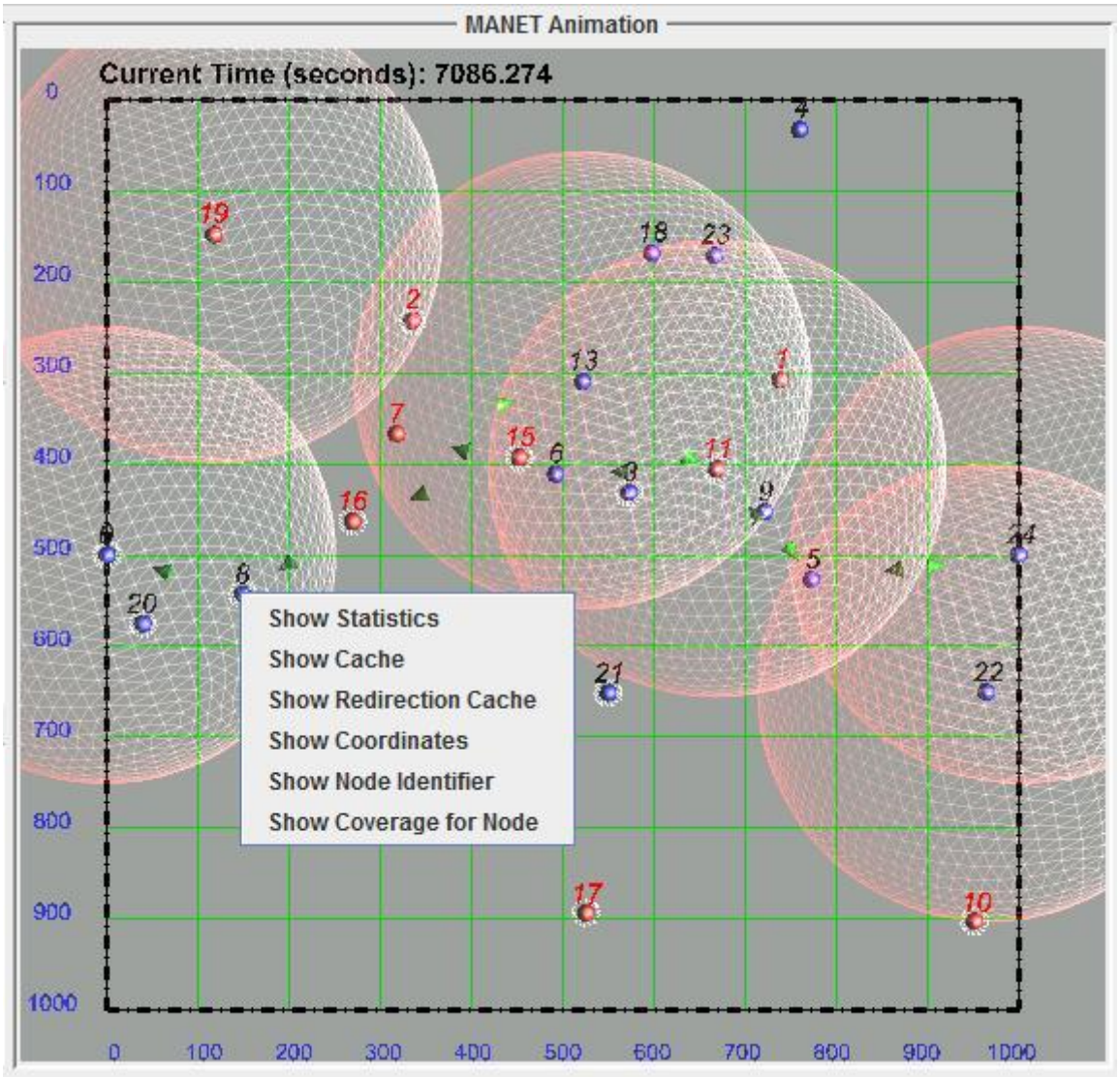


Figura 6.55. Menú contextual asociado a cada nodo del escenario

En el caso de solicitar las coordenadas reales del nodo, éstas serán representadas tal y como aparece en la figura 6.56.

Figura 6.56. Muestra de las coordenadas de un nodo en el escenario real

CAPÍTULO 7: Conclusiones y líneas futuras

Con este capítulo se concluye el análisis, el diseño y el desarrollo de la aplicación software que ha sido el objetivo este Proyecto Fin de Carrera. A través de las distintas etapas del desarrollo de este Proyecto se ha construido un software eficiente y de calidad que representa una novedosa aplicación cuya finalidad es la visualización de simulaciones de redes MANET con caché. Es muy interesante realizar un repaso para sintetizar las conclusiones más destacables, así como las posibles líneas de ampliación a través de las cuales se podría extender la funcionalidad de este software.

7.1. CONCLUSIONES

En los últimos años se ha producido un incremento considerable de los dispositivos móviles que tienen la posibilidad de conectarse a redes y, a la vez, interactuar con otros dispositivos. Debido a esto, ha surgido la necesidad en la comunidad científica de examinar y diseñar protocolos de red y aplicaciones para las redes móviles. La mayoría de este trabajo es realizado con la ayuda de los entornos de simulación de redes.

Como consecuencia del rápido crecimiento de las redes inalámbricas en general, se han requerido herramientas capaces de ofrecer de forma gráfica las conclusiones obtenidas a partir de los datos de simulaciones de redes. Uno de los simuladores de redes más extendidos es NS-2, un entorno muy popular utilizado para realizar simulaciones de redes MANET. Las necesidades de un entorno de visualización para los resultados de las simulaciones realizadas con NS-2 han constituido la base fundamental para la creación de este proyecto.

Un escenario típico de NS-2 contiene la configuración de la red y una serie de parámetros, como los patrones de tráfico y de movilidad de los nodos, que se codifican como entradas de datos, y cuando la simulación ha finalizado, se almacenan los resultados de las simulaciones en unos archivos de traza. Generalmente, los archivos de traza que son generados por las herramientas de simulación de redes contienen enormes cantidades de datos en formato texto, codificados según una especificación dada. Los archivos de trazas son utilizados tanto para realizar una evaluación estadística del rendimiento del protocolo bajo estudio como para realizar una depuración de errores. La primera actividad de

evaluación estadística del rendimiento examina el conjunto de trazas y calcula estimadores estadísticos para algunas propiedades del sistema como el retardo medio de los paquetes, la carga total de la red, etc. La segunda funcionalidad de depuración de errores se centra en estudiar e investigar cada uno de los eventos que forman parte de un archivo de trazas para comprender el comportamiento del protocolo. Este análisis es generalmente un desafío para el desarrollador del protocolo debido a que:

- debe leer un archivo de trazas y entender la semántica de cada uno de los eventos.
- debe mapear cada evento hacia un nodo de una red en una cierta posición geográfica, instante de tiempo, y una situación interna del nodo generalmente no incluida en el evento en cuestión.
- debe incluir el contexto definido por otros nodos que están posicionados en un entorno cercano, además de tener en consideración sus respectivos estados.

Por tanto, tras desarrollar nuevos protocolos de redes, los resultados de las simulaciones deberán ser analizados tanto estadísticamente como visualmente para verificar que se encuentran en concordancia con la hipótesis planteada. En el proceso de creación de un determinado protocolo de comunicaciones es necesario llevar a cabo una serie de pruebas que verifiquen el comportamiento esperado de los dispositivos conectados a dicha red, como son las pruebas de conformidad y las pruebas de interoperabilidad. Las pruebas de conformidad determinan si el sistema cumple la especificación (como por ejemplo, las especificaciones de redes expresadas en los RFC). Las pruebas de interoperabilidad verifican si los dispositivos son capaces de comunicarse a través del protocolo definido, aunque sean dispositivos de diferente naturaleza. Por todo esto, es muy importante realizar un análisis visual de los resultados de las simulaciones de redes.

Aunque la aplicación desarrollada es capaz de visualizar los archivos de trazas generados por cualquier simulador de redes (siempre que verifiquen el formato de mensajes presentado en el Apéndice A), los escenarios utilizados para la realización de las pruebas han sido generados con NS-2.

La aplicación para la visualización de redes creada en este proyecto constituye una herramienta muy útil para verificar el correcto comportamiento de los protocolos y validar la precisión de un modelo de movilidad, interpretando los archivos con la topología de los nodos utilizados para realizar las simulaciones. Es decir, hace posible estudiar incluso una nueva generación de patrones de movilidad más realistas y los comportamientos complejos de los nodos, siempre que verifiquen el formato requerido. Si nos centramos en el caso

particular del simulador de redes utilizado, NS-2, esta aplicación se puede emplear para que, tras realizar modificaciones en el protocolo estudiado, se puedan analizar los resultados, tanto de forma estadística como visualmente para comprobar que concuerdan con el análisis y las hipótesis realizadas.

Además, de cara a la necesidad planteada del estudio de redes MANET con caché local, este Proyecto ha generado una herramienta novedosa que se ajusta a dicho estudio, considerando las diversas estrategias de caché explicadas en el capítulo tercero. Mientras que algunos visualizadores dedican más esfuerzo a la parte gráfica, la aplicación desarrollada emplea adicionalmente gran parte de su esfuerzo a la visualización exacta de los eventos en los instantes temporales concretos del proceso de simulación. Es capaz de interpolar las posiciones intermedias que deben ocupar los nodos de forma automática, utilizando para ello únicamente los puntos origen y destino y la velocidad del movimiento. Como es necesario cargar los archivos inicialmente para la creación del escenario, realiza un post-procesado de la información introducida en forma de archivos de entrada, reduciendo así la sobrecarga.

Otra gran ventaja de esta aplicación de escritorio es su capacidad para desplazarse por la animación realizando pausas en aquellos eventos que estén relacionados con los nodos que se hayan indicado previamente. Esta funcionalidad implica la realización de un filtrado de los eventos que se hayan generado debido a la petición de un documento por un nodo de interés. Además, el usuario podrá moverse libremente por el transcurso de la animación a través de la barra horizontal o *slider* temporal, e incluso avanzar o retroceder hacia un instante determinado gracias a los saltos temporales o a los posicionamientos absolutos, indicando para ello los instantes de tiempo hasta el nivel de milisegundos. El usuario puede conocer la posición en el escenario real de cada nodo de la red.

También se pueden realizar movimientos por el escenario, así como usar las capacidades de *zoom*. Los colores y texturas empleados ayudan a detectar aquellos nodos que se encuentran seleccionados en cada momento, así como los nodos que están relacionados con las peticiones de documentos realizadas por otros nodos gracias a la incorporación de los halos. Finalmente, la muestra de las coberturas ayuda a detectar los nodos vecinos de cada nodo en un instante particular.

El usuario final de esta aplicación puede ver en ella no sólo una herramienta software que transforma los archivos de trazas en una completa animación del proceso de simulación, sino que este entorno visual ofrece una ventaja indiscutible en términos de

esfuerzo para realizar el análisis y depurar el protocolo de red empleado. Muestra la distribución o trazado de la red, posibilita el desplazamiento temporal a través de la animación, representa cada uno de los patrones de movilidad utilizados, realiza animaciones con la transferencia de los paquetes, calcula los parámetros estadísticos utilizados, muestra valores de las estructuras internas de los nodos como es el contenido de las cachés local y de redirecciones, etc. Todo esto, ofreciendo una interfaz gráfica sencilla y de diseño actual, favoreciendo su usabilidad. La aplicación también es capaz de generar informes en un formato muy extendido y estándar, como es PDF, con el resumen de los estadísticos de cada nodo y la comparativa de estos valores entre todos los nodos de la red, lo que dota a la aplicación de un valor añadido. Además, constituye una solución alternativa y atractiva respecto de otros visualizadores de redes por presentar las animaciones con gráficos en 3D, que aunque requieren de procesadores más potentes y de más memoria disponible, los rápidos avances tecnológicos en la actualidad imponen que este tipo de cuestiones no supongan un problema de importancia para esta aplicación.

Otra gran ventana de diseño de esta aplicación de escritorio es su carácter orientado a objetos, puesto que los sistemas orientados a objetos son más fáciles de mantener que los sistemas desarrollados con otras aproximaciones, por la característica fundamental de que los objetos son independientes. Cambiar la implementación de un objeto o agregarle servicios no debe afectar a los demás objetos del sistema, mientras se mantenga la interfaz entre ellos. Los objetos son componentes potencialmente reutilizables porque son encapsulamientos independientes de estado y operaciones. Los diseños se pueden desarrollar utilizando objetos creados en los diseños previos. Esto reduce los costes de diseño, programación y validación. La decisión tomada para la realización de un diseño cercano a la arquitectura software del MVC facilita la separación de la lógica de control, el manejo de los datos y la interfaz gráfica, además de favorecer las labores de mantenimiento y extensión del código. Esto implica que se establece una base arquitectónica sólida que sea flexible al cambio.

El lenguaje utilizado para desarrollar la aplicación ha sido Java por todas las ventajas que ofrece, como: es multiplataforma, requisito planteado en este Proyecto; tiene potentes APIs y librerías para el manejo de funciones gráficas, como la API para el desarrollo en 3D denominada Java3D; sólo requiere de la JVM para su ejecución, además de añadir Java3D a dicha máquina virtual por no estar integrada en ella por defecto, ambas fácilmente descargables a través de Internet; soporta la construcción de interfaces GUI a través de las librerías SWING y AWT; genera aplicaciones fácilmente desplegables a través de archivos

JAR; y presenta una amplia aceptación por parte de la comunidad académica y de investigación, facilitando el posible mantenimiento y extensión de su funcionalidad para que pueda llevarse a cabo en un futuro.

Con el fin de obtener un software de calidad duradera, se ha perseguido un desarrollo del software de forma eficiente, estableciendo claramente las diferentes fases del proceso de desarrollo de un software según la disciplina de la Ingeniería del Software. Estas distintas fases cubren los aspectos del análisis de requisitos, diseño, desarrollo y pruebas. Para favorecer un desarrollo que pueda adaptarse a las necesidades cambiantes de los usuarios y de la tecnología, para visualizar las diferentes vistas o perspectivas de la aplicación, y para ofrecer una documentación clara del código generado, se ha utilizado un modelado software, puesto que es fundamental para la producción de un buen software. Para ello se ha utilizado UML, un lenguaje muy potente y extendido que facilita la comunicación del comportamiento del sistema, consiguiendo un mejor entendimiento del funcionamiento del mismo y controlando, en la medida de lo posible, el riesgo. Los diagramas UML generados cubren las fases de análisis y diseño del software.

Finalmente, se puede concluir añadiendo que se ha perseguido el correcto funcionamiento de la aplicación, verificando cada uno de los más de 100 requisitos planteados durante la fase de análisis y generando un amplio banco de pruebas. Este banco de pruebas contiene más de 37 escenarios diversos, a través de los cuales se ha comprobado la capacidad multiplataforma, el funcionamiento de las diversas estrategias de caché planteadas en el capítulo tercero, y el consumo de la memoria, puesto que los archivos de entrada utilizados contienen, en ocasiones, más de 5 millones de líneas, lo que supone una cantidad de información a manejar muy elevada.

7.2. LÍNEAS FUTURAS

Como ocurre en la mayoría de aplicaciones generadas en diferentes ámbitos, tras el desarrollo completo de esta aplicación se han planteado numerosas ampliaciones de interés o líneas futuras:

- El protocolo estudiado en el presente Proyecto podría ser ampliado estableciendo una política de reemplazo en la caché de redirecciones, de la misma forma que se establece el manejo del contenido de la caché local. Incluso se podrían añadir nuevos eventos del archivo de salida de simulación *.txt* a interpretar. Tal y como se ha planteado el

software, la inclusión de nuevos mensajes se puede llevar a cabo con relativa poca inversión en software, ya que se deberían definir los nuevos formatos de mensajes, identificar a qué nodos están afectando estos mensajes y actualizar el valor de alguno de los estadísticos en el caso de que fuera necesario.

- Además de ampliar el protocolo de petición-respuesta empleado, se podrían generar nuevos protocolos realizando las oportunas definiciones de clases necesarias que adoptaran esos nuevos tipos de mensajes.
- Admitir nuevos formatos de eventos de movilidad de los nodos, puesto que los archivos de movilidad de los nodos son los generados por la herramienta NS-2. En la actualidad, existen otras herramientas que proveen diversos patrones de movilidad, como es el caso de BonnMotion [55], que es una herramienta de generación de escenarios de movilidad y de análisis. Esta herramienta, desarrollada por la Universidad de Bonn, es capaz de generar las trazas de movilidad en archivos con los formatos utilizados por algunos simuladores de redes, como NS-2, GloMoSim [56], Qualnet [56], e incluso en formato XML (*Extensible Markup Language*) siguiendo el esquema propuesto por el programa de investigación *Schwerpunktprogramm 1140* [57].
- Añadir nuevas funcionalidades y capacidades, como la posibilidad de seleccionar los paquetes que van por la red y poder consultar sus campos, incluyendo detalles del protocolo de encaminamiento utilizado. Incluso podría ser interesante implementar la visualización de datos internos al propio nodo, como las tablas de encaminamiento en aquellos protocolos en los que se utilicen. Además, sería de gran utilidad detectar los paquetes perdidos en la red, aspecto que no se puede descubrir en las simulaciones utilizadas.
- Añadir en los archivos de salida de las simulaciones aquellos paquetes enviados durante la búsqueda de rutas, implementando el software necesario para tal fin. Así, si se quisiera estudiar el protocolo de encaminamiento AODV, se podrían identificar los mensajes de búsqueda de ruta RREQ, RREP y RERR.
- Realizar capturas del escenario en un determinado instante de tiempo y presentarlas en un archivo externo a modo de imagen, e incluso almacenar en formato de vídeo el comportamiento del escenario entre dos instantes de tiempo facilitados.
- Añadir escenarios con coordenada z distinta de cero. La aplicación está desarrollada para que sea capaz de detectar nuevos valores de *z* distintos de cero, pero sería más interesante utilizar diferentes orografías de terrenos o perfiles de obstáculos. Con Java3D se pueden crear objetos, añadiendo texturas y geometrías y dibujando, por

ejemplo, desde montañas a edificios. En esos escenarios se podría explotar la funcionalidad aportada de la vista rotada del escenario, puesto que la aplicación permite introducirse dentro del propio escenario. En esos casos resultaría interesante añadir más movilidad a la cámara de visualización del escenario para la realización de giros diversos.

Bibliografía

- G. Booch, J. Rumbaugh y I. Jacobson, *El lenguaje Unificado de Modelado UML 2.0*. Madrid: Pearson Addison-Wesley, 2009.
- I. Sommerville, *Ingeniería del Software*. Madrid: Pearson Addison-Wesley, 2005.
- R. S. Pressman, *Ingeniería del Software, Un enfoque práctico*. Madrid: McGraw-Hill, 2002.
- S. K. Sarkar, T. G. Basavaraju y C. Puttamadappa, *Ad hoc mobile wireless networks: principles, protocols, and application.* New York: Auerbach Pubblications, 2008.
- M. Barbeau y E. Kranakis, *Principles of ad hoc networking.* Chichester, England: John Wiley & Sons Ltd., 2007.
- J. Friesen, *Beginning Java SE 6 Platform. From Novice to Professional*. New York: Apress, Springer-Verlag, 2007.
- J. Zukowski, *Java SE 6 Platform Revealed.* New York: Apress, Springer-Verlag, 2006.
- B. Eckel, *Piensa en Java.* Madrid: Prentice-Hall Pearson Educación, 2003.
- J. Zukowski, *Java 2 J2SE 1.4.* Madrid: Anaya Multimedia, 2003.
- A. Davison, *Pro Java 6 3D Game Development. Java 3D, JOGL, JInput and JOAL APIs.* New York: Apress, Springer-Verlag, 2007.
- D. Selman, *Java 3D Programming.* New York: Manning Publications, 2002. Libro electrónico.
- J. Carretero, P. de Miguel, F. García y F. Pérez, *Sistemas Operativos: una visión aplicada.* Madrid: McGraw-Hill, 2007.

Referencias

[1] E. M. Royer, y C-K., Toh, "A Review of Current Routing Protocols for *Ad Hoc* Mobile Wireless Networks". Dept. of Electrical And Computer Engineering, Univ. California, Santa Barbara y Dept. of Electrical And Computer Engineering, Georgia Institute of Technology, Atlanta: http://eprints.kfupm.edu.sa/20625/1/20625.pdf

[2] C. E. Perkins y P. Bhagwat; "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers"; ACM SIGCOMM, vol.24, no.4. Octubre, 1994.

[3] T. Clausen, y P., Jacquet, "Optimized Link State Routing Protocol (OLSR)". Internet Engineering Task Force Request for Comments (RFC) 3626, Octubre 2003.

[4] S. Murthy, y J.J. García-Luna-Aceves, "An Efficient Routing Protocol for Wireless Networks". Mobile Networks and Applications: special issue in mobile communications networks, vol. 1, no. 2,pp. 183-197, 1996.

[5] C. E. Perkins, E. M. Belding-Royer, y S. Das, "*Ad Hoc* On Demand Distance Vector (AODV) Routing". Internet Engineering Task Force Request for Comments (RFC) 3561, Julio 2003.

[6] D. B. Johnson, D. A. Maltz, y Y.C. Hu, "The Dynamic Source Routing Protocol (DSR) for Mobile *Ad Hoc* Networks for IPv4". Internet Engineering Task Force Request for Comments (RFC) 4728, Febrero 2007.

[7] V. Park, y S. Corson, "Temporally-Ordered Routing Algorithm (TORA)". Internet Engineering Task Force, Internet draft, Julio 2001.

[8] Z. J. Haas, M. R. Pearlman, y P. Samar, "The Zone Routing Protocol (ZRP) for *Ad Hoc* Networks". Internet Engineering Task Force, Internet draft, Julio 2002.

[9] F. J González-Cañete, E. Casilari, y A. Triviño-Cabrera, "Propuesta y evaluación de un esquema de caché para redes *ad hoc*", en *VIII Jornadas de Ingeniería Telemática (Jitel 2009)*, Septiembre 2009.

[10] F. J. González-Cañete, E. Casilari, y A. Triviño-Cabrera, "Cross-layer Interception Caching in MANETs", Dept. Tecnología Electrónica, Universidad of Málaga. Pendiente de publicación.

[11] Herramienta de simulación de redes NS-2: http://www.isi.edu/nsnam/ns/

[12] Extensiones de movilidad para redes inalámbricas:

http://www.monarch.cs.rice.edu/cmu-ns.html

[13] Proyecto Monarch:

http://www.cs.cmu.edu/afs/cs.cmu.edu/Web/People/dbj/monarch.html

[14] S. Kurkowski, T. Camp, y M. Colagrosso, "A Visualization and Animation Tool for NS-2 Wireless Simulations: iNSpect", Actas del 13º IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 503-506, 2005.

[15] Herramienta de visualización de redes iNSpect:

http://toilers.mines.edu/Public/NsInspect

[16] B. Scheuermann, M. Mauve, Busse, M., H. Füßler, M. Transier, y W. Effelsberg, "Huginn: A 3D Visualizer for Wireless ns2 Traces", Actas del 8º ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems, pp. 143-150, Montreal (Canadá), 2005.

[17] L. S. Nikolaos, "EXtensible Animator for Mobile Simulations: EXAMS", University of Patras, Grecia: http://livathinos.gr/EXAMS_overview.aspx?down=paper

[18] Herramienta de simulación de redes GloMoSim:

http://pcl.cs.ucla.edu/projects/glomosim/

[19] Tutoriales de Java SE: http://java.sun.com/docs/books/tutorial/

[20] Características de la tecnología Java: http://www.java.com/es/about/

[21] Visión general de Java SE: http://java.sun.com/javase/index.jsp

[22] Descargas de Java SE: http://java.sun.com/javase/downloads/index.jsp

[23] Especificaciones de Java:

http://jcp.org/aboutJava/communityprocess/final/jsr270/index.html

[24] Entorno Integrado de Desarrollo Netbeans: http://netbeans.org/

[25] Descarga de JDK y Netbeans:

http://java.sun.com/javase/downloads/widget/jdk_netbeans.jsp

[26] Tutorial de Netbeans:

http://java.sun.com/developer/onlineTraining/tools/netbeans_part1

[27] Formato PDF: http://www.cde.ua.es/cde/pdf.htm

[28] Librería para creación de PDF en Java iText: http://itextpdf.com/

[29] Librería de gráficos en Java jFreeChart: http://www.jfree.org/jfreechart/

[30] Librería miscelánea para Java jCommon: http://www.jfree.org/jcommon/

[31] Opiniones de clientes acerca de Java SE:

http://java.sun.com/javase/community/customers.jsp#wallace

[32] API para gráficos OpenGL: http://www.opengl.org/

[33] API para gráficos Directx:

http://www.microsoft.com/games/en-US/aboutGFW/pages/directx.aspx

[34] Juego en Java3D "*Tribal Trouble*": http://tribaltrouble.com/

[35] Juego en Java3D "*Puzzle Pirates*": http://www.puzzlepirates.com/

[36] Juego en Java3D "*Call of Juarez*": http://callofjuarez.es.ubi.com/demo/splash.php

[37] Juego en Java3D "*Star Wars Galaxies*":

http://starwarsgalaxies.station.sony.com/players/index.vm

[38] TIOBE *Programming Community Index*:

http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

[39] Java3D *Parent Project*: https://java3d.dev.java.net/

[40] JOGL API *Project*: https://jogl.dev.java.net/

[41] API Microsoft Direct3D:

http://msdn.microsoft.com/en-us/library/bb318764%28VS.85%29.aspx

[42] Especificación y guía del programador para JNI: http://java.sun.com/docs/books/jni/

[43] Java3D API: http://java.sun.com/javase/technologies/desktop/java3d/

[44] Tutorial de Java3D API: http://java.sun.com/developer/onlineTraining/java3d/

[45] Tutorial de Marc Greis de NS-2: http://www.isi.edu/nsnam/ns/tutorial/index.html

[46] E. Altman, y T. Jiménez, "NS Simulator for beginners", Universidad de los Andes, Mérida, Venezuela, y ESSI, Sophia-Antipolis, Francia, Diciembre 2003: http://www-sop.inria.fr/members/Eitan.Altman/COURS-NS/n3.pdf

[47] Licencia GPL: http://www.gnu.org/licenses/licenses.es.html#GPL

[48] Especificaciones y borradores de HTTP: http://www.w3.org/Protocols/Specs.html

[49] S. Shakkottai, T. S. Rappaport y P. C. Karlsson, "Cross-layer Design for Wireless Networks", Department of Electrical and Computer Engineering, Univ. Texas, Austin, y Telia Research AB, Suecia, junio 2003:

http://users.ece.utexas.edu/~shakkott/Pubs/cross-layer.pdf

[50] Tanenbaum, A., *Redes de computadoras.* Mexico: Pearson Addison-Wesley, 1997.

[51] Distribución exponencial: http://www.bioestadistica.uma.es/libro/node78.htm

[52] L. A. Adamic y B. A. Huberman, "Zipf's law and the Internet", Glottometrics, vol. 3, pp. 143-150, 2002.

[53] S. Lim, W.C. Lee, G. Cao y C.R. Das, "A novel caching scheme for improving Internet-based mobile ad hoc networks performance", Ad Hoc Networks, vol. 4, 225-239, 2006.

[54] L. Yin y G. Cao, "Supporting Cooperative Caching in Ad Hoc Networks". IEEE Transactions on Mobile Computing, Vol. 5, No. 1, pp.77- 89, 2006.

[55] Herramienta de generación de escenarios de movilidad BonnMotion:

http://net.cs.uni-bonn.de/wg/cs/applications/bonnmotion/

[56] Herramienta de simulación de redes Qualnet:

http://www.scalable-networks.com/products/qualnet/

[57] Programa de investigación “Schwerpunktprogramm140”:

http://www.tm.uka.de/forschung/SPP1140/

Apéndice A: Formato de los mensajes y de las cabeceras de los archivos

Los mensajes y las cabeceras que provienen del archivo de movilidad de los nodos *.pos* y del archivo de salida de la simulación con la actividad de la red *.txt* siguen el formato que se muestra en este apéndice. Este formato es el que es capaz de interpretar el visualizador y por esta razón, como se indicó en el capítulo introductorio, no existe la necesidad de que se utilice NS-2 como simulador de redes. De esta manera, se podría utilizar cualquier simulador de redes siempre que produjeran los eventos y las cabeceras con el formato correspondiente. En primer lugar, se van a presentar los mensajes según el tipo de información que representan, es decir, diferenciando los que indican la situación inicial y la movilidad de los nodos de los que se producen como consecuencia del funcionamiento de un protocolo. Hay que destacar que cada uno de estos mensajes deberá aparecer en líneas separadas, correspondiendo cada línea con un evento diferente. A continuación, se explicará qué información deberá aparecer en las cabeceras de los dos archivos de entrada a la aplicación.

Se va a expresar el formato de cada mensaje o cabecera añadiendo el texto que es fijo para ese tipo en particular, además de las expresiones **%d** y **%f** indicando aquellos lugares donde deba aparecer un número decimal o un número en punto flotante, respectivamente. Junto con las anteriores expresiones, exceptuando el caso de las cabeceras que únicamente contengan un campo, aparecerá un subíndice numérico de forma que se puedan identificar cada uno de los campos con contenido variable.

A.1. MENSAJES DEL ARCHIVO DE MOVILIDAD DE LOS NODOS (*.POS*)

Los mensajes que aparecen en el archivo de movilidad de los nodos siguen el formato estándar de NS-2. De hecho, el contenido de este archivo representa información de entrada para las simulaciones con NS-2 en escenarios con movilidad de nodos.

A.1.1. SITUACIÓN INICIAL DE LOS NODOS DEL ESCENARIO

Representa la posición inicial de un nodo de la red. En concreto, indica las coordenadas físicas del escenario real en metros que debe ocupar el nodo al arrancar la animación. Dichas coordenadas aparecerán indicadas por separado. Así, para cada nodo existirán tres

mensajes, uno a continuación del otro, de forma que se indiquen las 3 coordenadas en el espacio que debe ocupar. En el caso de que no aparezca información para alguna de las coordenadas, se considerará por defecto que vale 0. La coordenada "z" en estos escenarios simulados va a ser siempre 0.

Formato

- Para la coordenada "x": $node_(**%d**$_1$) set X_ **%f**$_2$
- Para la coordenada "y": $node_(**%d**$_1$) set Y_ **%f**$_2$
- Para la coordenada "z": $node_(**%d**$_1$) set Z_ **%f**$_2$

- 1. *idCurrentNode*: identificador del nodo cuya coordenada viene indicada.
- 2. *coordinate*: valor que debe tener la coordenada que viene indicada.

Ejemplos

Se va a mostrar a continuación un ejemplo para cada una de las coordenadas:

$node_(28) set X_ 125.000000000000

Indica que la coordenada "x" del nodo 28 debe ser, inicialmente, igual a 125 metros.

$node_(28) set Y_ 375.000000000000

Indica que la coordenada "y" del nodo 28 debe ser, inicialmente, igual a 375 metros.

$node_(28) set Z_ 0.000000000000

Indica que la coordenada "z" del nodo 28 debe ser, inicialmente, igual a 0 metros.

A.1.2. MOVILIDAD DE LOS NODOS EN EL ESCENARIO - CAMBIOS DE DIRECCIÓN

Representa el cambio de dirección que experimenta un nodo en un determinado instante de tiempo. En concreto, indica las coordenadas físicas del escenario real en metros hacia las que debe desplazarse el nodo siguiendo un movimiento rectilíneo a la velocidad que venga representada. Como en los escenarios generados se considera una altura del nodo igual a 0, la coordenada "z" no vendrá representada. Se debe tener en cuenta que cuando el nodo deba modificar su trayectoria, puede darse el caso de que esté tanto pausado como avanzando en otra dirección diferente e incluso a distinta velocidad.

Formato

$ns_ at **%f**$_1$"$node_(**%d**$_2$) setdest **%f**$_3$ **%f**$_4$ **%f**$_5$"

- 1. *currentTime*: instante de tiempo en segundos en el que se produce el cambio de dirección del nodo.
- 2. *idCurrentNode*: identificador del nodo que cambia de dirección.
- 3. *"x"-coordinate*: coordenada "x" que posee el punto destino del movimiento que debe realizar el nodo.
- 4. *"y"-coordinate*: coordenada "y" que posee el punto destino del movimiento que debe realizar el nodo.
- 5. *nodeSpeed*: velocidad que debe llevar el nodo en su movimiento rectilíneo hacia las coordenadas indicadas, representada en metros/segundo.

Ejemplo

$ns_ at 77.341803 "$node_(1) setdest 300.590826 513.985702 5.000000"

Indica que el nodo 1 debe cambiar su dirección en el instante 77.341803 (en segundos) desde la posición que esté ocupando hacia el punto del escenario real representado por las coordenadas "x" = 300.590826 e "y" = 513.985702, a una velocidad de 5 metros/segundo.

A.2. MENSAJES DEL ARCHIVO DE SIMULACIÓN (*.TXT*)

A.2.1. NIVEL DE APLICACIÓN – MENSAJES DE PETICIÓN

Los tres tipos de mensajes que se van a mostrar a continuación se dan en la capa de aplicación del nodo que realiza una determinada petición. La identificación para los mensajes de esta capa contiene las siglas CNA (*CacheNode Application*), puesto que la implementación utilizada genera un nivel de aplicación de la pila de protocolos que contiene una memoria caché local.

A.2.1.1. Acierto en caché local

Representa una petición GET de un documento realizada en un nodo para el que se tiene una copia en su caché local, es decir, se produce un acierto en caché. Se trata de un mensaje que, consecuentemente, no produce tráfico en la red, puesto que desde el mismo nivel de aplicación es servida la petición.

Formato

%f$_1$ %d$_2$ CNA - Local HIT - ID: **%d$_3$** - REQUEST: **%d$_4$**

- 1. *currentTime*: instante de tiempo en segundos en el que se produce la petición del documento y el acierto en caché local.
- 2. *idCurrentNode*: identificador del nodo que realiza la petición y produce el acierto en caché local.
- 3. *idDocumentRequested*: identificador del documento solicitado por el nodo en la petición GET.
- 4. *idCurrentRequest*: número que corresponde a la petición actual de entre todas las peticiones que se han producido en el nodo desde el inicio de la simulación. Hay que destacar que la primera petición que realice el nodo tendrá siempre el identificador 0.

Ejemplo

168.045208 15 CNA - Local HIT - ID: 4 - REQUEST: 4

Indica que el nodo 15 ha realizado su quinta petición (porque los identificadores de peticiones comienzan en 0) desde el inicio de la simulación, en el instante 168.045208 segundos, solicitando el documento con identificador 4 y para el que además tenía una copia válida del mismo en su caché local.

A.2.1.2. Petición GET

Representa la petición de un documento realizada en un nodo para el que no se tiene una copia en su caché local. Se trata del evento que desencadena el intercambio de mensajes en la red, tanto para la búsqueda de la ruta como para la retransmisión de la petición y de su respuesta.

Formato

%f$_1$ %d$_2$ CNA - **%d$_3$:%d$_4$** - send - GET - ID: **%d$_5$** - REQUEST: **%d$_6$**

- 1. *currentTime*: instante de tiempo en segundos en el que se produce la petición del documento.
- 2. *idCurrentNode*: identificador del nodo que realiza la petición.
- 3. *idSourceNode*: identificador del nodo que realiza la petición. Corresponde por tanto con el identificador del nodo que aparece en 2, es decir, el nodo actual.

- 4. *idDestinationNode*: identificador del nodo al que se le solicita el documento. Corresponderá con el identificador de un nodo servidor que contenga el documento solicitado.
- 5. *idDocumentRequested*: identificador del documento solicitado por el nodo.
- 6. *idCurrentRequest*: número de la petición actual de entre todas las que se han producido en el nodo desde el inicio de la simulación.

Ejemplo

166.680805 06 CNA - 06:00 - send - GET - ID: 24 - REQUEST: 5

Indica que el nodo 6 ha realizado su sexta petición desde el inicio de la simulación, en el instante 166.680805 segundos, solicitando el documento con identificador 24 y para el que además no tiene una copia válida del mismo en su caché local. El nodo al que se dirige esta petición es un el nodo servidor de datos con identificador 0.

A.2.1.3. *Timeout* de recepción

Representa un aviso para la repetición de la petición de un documento realizada con anterioridad en un nodo para el que no se tiene una copia válida en su caché local. Este mensaje se produce cuando se agota el temporizador en el nivel de aplicación del nodo, que es activado tras el envío de la petición del documento. Este temporizador es utilizado para detectar las posibles pérdidas de mensajes por la red.

Formato

%f$_1$ %d$_2$ CNA - Time out: **%d$_3$**

1. *currentTime*: instante de tiempo en segundos en el que se produce el *timeout*, que corresponde al instante de tiempo en el que se produce la petición de un cierto documento más un cierto *offset* fijado para la simulación.

2. i*dCurrentNode*: identificador del nodo que realiza la petición y que posteriormente experimenta un *timeout*.

3. *idDocumentRequested*: identificador del documento que fue solicitado con anterioridad y para el que no se ha obtenido una respuesta.

Ejemplo

168.481181 17 CNA - Time out: 30

Indica que el nodo 17 ha realizado la petición del documento con identificador igual a 30 con anterioridad y que en el instante 168.481181 (en segundos) se ha agotado su temporizador, indicándole que debe solicitar de nuevo el mismo documento puesto que aún no se ha recibido.

A.2.2. NIVEL DE APLICACIÓN – MENSAJES DE RECEPCIÓN EN LOS SERVIDORES

A.2.2.1. Recepción de una petición GET

Representa la recepción en el lado del servidor de la petición GET de un documento que ha sido realizada por un nodo cliente de la red. Una vez que el servidor pueda tramitar esta solicitud, enviará el documento solicitado si tiene una ruta activa hacia el nodo que lo solicitó.

Formato

%f$_1$ %d$_2$ CNA - **%d$_3$:%d$_4$** - recv - SERVER - ID: **%d$_5$**

- 1. *currentTime*: instante de tiempo en segundos en el que se produce la recepción de la petición.
- 2. *idCurrentNode*: identificador del nodo actual en el que se recibe la petición, que corresponde con un nodo servidor.
- 3. *idSourceNode*: identificador del nodo que realizó la petición y que es el origen del camino que ha recorrido ésta.
- 4. *idDestinationNode*: identificador del nodo al que se ha solicitado el documento, en este caso, el servidor, que además es el nodo actual.
- 5. *idDocumentRequested*: identificador del documento solicitado por el nodo que originó la petición que se acaba de recibir en este servidor y que debe ser devuelto como respuesta a la misma.

Ejemplo

174.805204 24 CNA - 15:24 - recv - SERVER - ID: 785

Indica que el nodo 24 ha recibido en el instante 174.805204 (en segundos) la petición realizada por el nodo 15, solicitando el documento con identificador 785.

A.2.2.2. Envío de la respuesta a una petición GET

Representa el envío de una respuesta en el lado del servidor a una petición GET recibida de un documento que ha sido realizada por un nodo. Una vez que el servidor pueda tramitar la solicitud recibida, enviará el documento solicitado si hay ruta hacia el nodo que lo solicitó. Además, en la misma respuesta se añadirán una serie de parámetros del documento, como su tiempo de vida o TTL y su tamaño en *bytes*.

Formato

%f$_1$ %d$_2$ CNA - **%d$_3$:%d$_4$** - send - SERVER - ID: **%d$_5$** - Size: **%d$_6$** - TTL: **%f$_7$**

- 1. *currentTime*: instante de tiempo en segundos en el que se produce el envío del documento.
- 2. *idCurrentNode*: identificador del nodo que envía la respuesta, el nodo actual, que corresponde con un nodo servidor.
- 3. *idSourceNode*: identificador del nodo al que se le solicita el documento, que corresponde con el nodo actual que envía este mensaje de respuesta.
- 4. *idDestinationNode*: identificador del nodo que realizó la petición, que corresponde con el destino de este mensaje de respuesta.
- 5. *idDocumentRequested*: identificador del documento solicitado por el nodo que originó la petición y que es servido en esta respuesta.
- 6. *sizeDocumentRequested*: tamaño en *bytes* del documento solicitado que va incluido en este mensaje de respuesta.
- 7. *ttlDocumentRequested*: representa el tiempo de expiración de la información contenida en el propio documento, es decir, el instante a partir del cual dicha información se vuelve obsoleta.

Ejemplo

174.805204 24 CNA - 24:15 - send - SERVER - ID: 785 - Size: 1000 - TTL: 5406.474275

Indica que el nodo 24 ha enviado en el instante 174.805204 (en segundos), como nodo origen del camino que seguirá este mensaje, la respuesta a la petición realizada por el destino de este camino, que es el nodo 15. El documento solicitado tiene identificador 785, tamaño 1000 *bytes* y su instante de expiración es el 5406.474275.

A.2.2.3. Retransmisión de un error de redirección en un servidor

Representa la retransmisión de un error recibido desde otro nodo indicando que se ha producido un error de redirección. El error de redirección se identifica como un error en el envío de un GET o petición por parte de un nodo. Este error se debe enviar hacia el nodo que solicitó el documento y que ha sufrido el error para que sea capaz de detectar el problema; de esta forma, podrá solicitar de nuevo el mismo documento. El error de redirección se va enrutando de nodo a nodo, y cada nodo intermedio se limita a propagar el error, pudiendo ser un nodo cliente de la red o bien un nodo servidor de la red que, para esta funcionalidad, haría el papel de cualquier otro nodo cliente de la red. Como se ha dicho anteriormente, este mensaje se envía creando rutas de nodo a nodo, es decir, rutas de un salto. Así, en los identificadores de los nodos origen y destino aparecen el origen y el destino del salto, respectivamente, y no aparece la ruta completa hacia y/o desde el nodo que solicitó el documento que ha sufrido el error de redirección.

Formato

%f$_1$ %d$_2$ CNA - **%d$_3$:%d$_4$** - recv - SERVER - ERROR_GET FORWARDED ID: **%d$_5$** - NODE: **%d$_6$**

- 1. *currentTime*: instante de tiempo en segundos en el que se produce la retransmisión del error de redirección recibido.
- 2. *idCurrentNode*: identificador del nodo actual que retransmite el error de redirección recibido, que corresponde con un nodo servidor.
- 3. *idSourceNode*: identificador del nodo desde donde se envía este mensaje de error, es decir, el nodo actual, puesto que este error se transmite de forma especial buscando una ruta para cada salto.
- 4. *idDestinationNode*: identificador del nodo destino hacia el que se envía este mensaje, que corresponde con un nodo que se encuentra a un salto de éste, ya que este error se transmite de forma especial buscando una ruta para cada salto.
- 5. *idDocumentRequested*: identificador del documento solicitado por el nodo que originó la petición GET que ha experimentado el error de redirección.
- 6. *idNode*: identificador del nodo que realizó la petición GET que ha experimentado el error de redirección.

Ejemplo

1261.570947 00 CNA - 00:12 - recv - SERVER - ERROR_GET FORWARDED ID: 4 - NODE: 21

Indica que el nodo 0, que es un nodo servidor, ha retransmitido en el instante 1261.570947 (en segundos) el error de redirección que ha sufrido la petición GET realizada por el nodo 21 para la solicitud del documento con identificador 4. Este error debe ser retransmitido desde este nodo origen 0 hasta el nodo destino 12, que se encuentra a un salto de este nodo.

A.2.3. NIVEL DE APLICACIÓN – MENSAJES DE RECEPCIÓN EN LOS CLIENTES

A.2.3.1. Recepción de una respuesta RESP

Representa la recepción de la respuesta a una petición GET que contiene el documento solicitado. En este mensaje se encuentra el documento en sí y una serie de parámetros acerca de cómo ha sido servida la petición, así como algunas características del documento. Así, por un lado nos encontramos con información del identificador del documento, su tamaño en *bytes* y su instante de expiración o TTL. Por otro lado, contiene la información acerca del instante en el que ocurrió la petición GET, para que se pueda conocer a cuáles de las peticiones corresponde esta respuesta, ya que se podría dar el caso de producir un *timeout*, solicitar de nuevo el mismo documento y que llegue la respuesta justo después de enviar por la red la segunda petición. Además, se incorpora el número de saltos que ha experimentado la petición del documento, incluidos los saltos que se podrían producir si se diera un error en la redirección y la posterior petición del mismo documento. Por último, aparece indicado si la petición ha sufrido un acierto de intercepción, un acierto de intercepción con recursos de encaminamiento, un acierto de redirección, o la combinación de algunos de ellos. Podría venir incluso indicado si el documento recibido no corresponde con el documento que este nodo espera recibir, o si la respuesta no corresponde a la última petición realizada de ese mismo documento.

Formato

%f$_1$ %d$_2$ CNA - **%d$_3$:%d$_4$** - recv - RESP - ID: **%d$_5$** - NHOPS: **%d$_6$** - Size: **%d$_7$** - TTL: **%f$_8$** - RTIME: **%f$_9$ [X]$_{10}$**

- 1. *currentTime*: instante de tiempo en segundos en el que se produce la recepción de este mensaje, es decir, del documento solicitado por este nodo.
- 2. *idCurrentNode*: identificador del nodo que solicitó el documento que acaba de recibir en este mensaje.
- 3. *idSourceNode*: identificador del nodo que ha enviado este mensaje de respuesta con el documento solicitado, que corresponde con el nodo que ha servido la petición, ya sea un nodo servidor de datos o un nodo cliente gracias a una intercepción o redirección.
- 4. *idDestinationNode*: identificador del nodo hacia el que iba dirigido este mensaje de respuesta, que corresponde con el nodo que solicitó el documento, es decir, el nodo actual.
- 5. *idDocumentRequested*: identificador del documento solicitado por este nodo y que acaba de recibir en este mensaje.
- 6. *nHops*: número de saltos que ha seguido la petición hasta que ha sido finalmente servida, es decir, desde el envío del mensaje GET hasta la recepción de este mensaje.
- 7. *sizeDocumentRequested*: tamaño en *bytes* del documento solicitado que va incluido en este mensaje de respuesta.
- 8. *ttlDocumentRequested*: representa el tiempo de expiración de la información contenida en el propio documento, es decir, el instante a partir del cual dicha información se vuelve obsoleta.
- 9. *rTime*: instante en el que este nodo realizó la petición que corresponde a la respuesta recibida.
- 10. *X*: corresponde a información acerca de si para servir la petición GET ha sido utilizado algún mecanismo de caché o si se trata de un documento no solicitado. Este campo puede aparecer sin información, esto es, vacío. Si fuera así, indica que la petición ha sido servida con el mecanismo por defecto, es decir, a través de una petición y respuesta por parte de un nodo servidor de datos. Las posibilidades que podrían aparecer son:
    - IH: *Interception Hit* – Acierto de Intercepción remota.
    - RH: *Redirection Hit* – Acierto de Redirección
    - RPH: *Routing Protocol Hit* – Acierto de Intercepción con recursos de encaminamiento.
    - NR: *Not Requested* – Documento no solicitado, o no corresponde con la petición que está esperando que sea servida.

Además, se puede dar el caso de que aparezcan varios de estos mecanismos, uno a continuación de otro.

Ejemplos

Se van a proponer algunos ejemplos de las varias posibilidades que se pueden encontrar:

8675.530541 02 CNA - 00:02 - recv - RESP - ID: 90 - NHOPS: 4 - Size: 1000 - TTL: 12053.040452 - RTIME: 8675.512706

Indica que el nodo 2, que es un nodo cliente, ha recibido en el instante 8675.530541 (en segundos) la respuesta a la petición GET que realizó este mismo nodo para la solicitud del documento con identificador 90. La petición GET ha sido servida sin el uso de ningún mecanismo de caché de los planteados, es decir, ha sido resuelta por un nodo servidor de datos. Esta respuesta ha sido enviada desde el nodo origen 0, que es un nodo servidor, hacia el nodo destino 2, que es este nodo actual, y contiene el documento solicitado e información sobre él, como su tamaño (1000 *bytes*) y su instante de expiración (12053.040452 segundos). Además indica que esta respuesta corresponde a la petición realizada por este nodo en el instante 8675.512706 y que ha sido servida con un total de 4 saltos.

7359.324079 13 CNA - 07:13 - recv - RESP - ID: 0 - NHOPS: 2 - Size: 1000 - TTL: 7493.457240 - RTIME: 7359.320571 IH

Indica que el nodo 13, que es un nodo cliente, ha recibido en el instante 7359.324079 (en segundos) la respuesta a la petición GET que realizó este mismo nodo para la solicitud del documento con identificador 0. La petición GET ha sido servida con el uso del mecanismo de intercepción remota estudiado, es decir, ha sido resuelta por un nodo cliente que se encontraba en la ruta desde este nodo 13 hacia el servidor que contiene el documento con identificador 0. Esta respuesta ha sido enviada desde el nodo origen 7, que es un nodo cliente, hacia el nodo destino 13, que es este nodo actual, y contiene el documento solicitado e información sobre él, como su tamaño (1000 *bytes*) y su instante de expiración (7493.457240 segundos). Además indica que esta respuesta corresponde a la petición realizada por este nodo en el instante 7359.320571 y que ha sido servida con un total de 2 saltos, lo que sugiere que el nodo cliente que ha interceptado la petición se encontraba a un salto del nodo que había solicitado el documento.

6773.119368 17 CNA - 14:17 - recv - RESP - ID: 12 - NHOPS: 6 - Size: 1000 - TTL: 7280.171796 - RTIME: 6773.076092 RH IH

Indica que el nodo 17, que es un nodo cliente, ha recibido en el instante 6773.119368 (en segundos) la respuesta a la petición GET que realizó este mismo nodo para la solicitud del documento con identificador 12. La petición GET ha sido servida con el uso de los mecanismos de redirección e intercepción remota estudiados. En este caso, se ha producido inicialmente una redirección de la petición hacia otro nodo, se ha enrutado hacia ese nodo, y en este camino ha sido interceptado por un nodo cliente que contenía una copia válida del documento con identificador 12. Esta respuesta ha sido enviada desde el nodo origen 14, que es el nodo cliente que ha realizado la intercepción, hacia el nodo destino 17, que es este nodo actual, y contiene el documento solicitado e información sobre él, como su tamaño (1000 *bytes*) y su instante de expiración (7280.171796 segundos). Además indica que esta respuesta corresponde a la petición realizada por este nodo en el instante 6773.076092 y que ha sido servida con un total de 6 saltos.

186.643980 04 CNA - 00:04 - recv - RESP - ID: 114 - NHOPS: 15 - Size: 1000 - TTL: 193.277114 - RTIME: 183.600690 NR

Indica que el nodo 4, que es un nodo cliente, ha recibido en el instante 186.643980 (en segundos) la respuesta a la petición GET que realizó este mismo nodo para la solicitud del documento con identificador 114. La petición GET ha sido servida sin el uso de ningún mecanismo de caché de los planteados, es decir, ha sido resuelta por un nodo servidor de datos. Esta respuesta ha sido enviada desde el nodo origen, que es un nodo servidor, hacia el nodo destino 4, que es este nodo actual, y contiene el documento solicitado e información sobre él, como su tamaño (1000 *bytes*) y su instante de expiración (193.277114 segundos). Además indica que esta respuesta corresponde a la petición realizada por este nodo en el instante 183.600690, pero no corresponde a la última petición que ha realizado de este mismo documento, así que es desechada. Finalmente, se indica que ha sido servido el documento con un total de 15 saltos.

A.2.3.2. Recepción de una respuesta con un documento que no se ha solicitado

Representa la recepción de un documento que no ha sido solicitado, o bien que no corresponde con la petición última que se ha realizado de ese mismo documento. Aparece justo tras la recepción de un documento en un nodo.

Formato

%f$_1$ %d$_2$ CNA - **%d$_3$:%d$_4$** - recv - CLIENT - NOT REQUESTED - **%d$_5$:%d$_6$**

- 1. *currentTime*: instante de tiempo en segundos en el que se produce este mensaje.
- 2. *idCurrentNode*: identificador del nodo actual que recibe el documento y detecta que no es el que estaba esperando, si esperaba alguno.
- 3. *idSourceNode*: identificador del nodo que envía este mensaje, que corresponde con el nodo actual.
- 4. *idDestinationNode*: identificador del nodo hacia el que se envía este mensaje, que corresponde con el nodo actual, puesto que en realidad no es un mensaje que se llegue a transmitir por la red, sino más bien un aviso en el propio nodo.
- 5. *idDocumentRequested*: identificador del documento que espera recibir el nodo.
- 6. *idDocumentReceived*: identificador del documento que ha recibido el nodo.

Ejemplo

186.643980 04 CNA - 04:04 - recv - CLIENT - NOT REQUESTED - 114:114

Indica que el nodo 4, que es un cliente, ha detectado en el instante 186.643980 (en segundos) que ha recibido el documento con identificador 114 correspondiente a una petición que no es la que este nodo esperaba que fuera servida.

A.2.3.3. Intercepción de una petición GET

Representa la intercepción de la petición GET de un documento. Es decir, se recibe un mensaje de petición GET que debe ser retransmitido al siguiente nodo en el camino, y se verifica que se tiene una copia válida del documento solicitado en su caché local. A este mecanismo se le denomina acierto de intercepción, y es uno de los mecanismos bajo estudio para este proyecto.

Formato

%f$_1$ %d$_2$ CNA - **%d$_3$:%d$_4$** - recv - CLIENT - INTERCEPTION HIT ID: **%d$_5$**

- 1. *currentTime*: instante de tiempo en segundos en el que se produce la intercepción.
- 2. *idCurrentNode*: identificador del nodo actual que recibe la petición GET y descubre que tiene una copia válida en su caché local antes de retransmitir la petición hacia el siguiente nodo en el camino.
- 3. *idSourceNode*: identificador del nodo que realizó la petición GET que está siendo interceptada.

- 4. *idDestinationNode*: identificador del nodo destino hacia el que se ha dirigido la petición, que podía ser un nodo servidor de datos o bien un nodo cliente en el caso de que se hubiera producido una redirección.
- 5. *idDocumentRequested*: identificador del documento que había sido solicitado en la petición GET interceptada y para el que se encuentra en el nodo actual una copia válida.

Ejemplo

7584.859532 19 CNA - 05:00 - recv - CLIENT - INTERCEPTION HIT ID: 2

Indica que el nodo 19, que es un nodo cliente, ha recibido en el instante 7584.859532 (en segundos) una petición GET realizada por el nodo 5 para que sea retransmitida al siguiente nodo hacia el nodo destino 0, y se ha verificado que se tiene una copia válida del documento con identificador 2 en su caché local. Entonces es capaz de responder a la petición sin necesidad de retransmitirla.

A.2.3.4. Intercepción de una petición GET con recursos de encaminamiento

Representa la intercepción con recursos de encaminamiento de la petición GET de un documento. Es decir, en la búsqueda de una ruta para el envío de un mensaje GET, ocurre que un nodo intermedio posee una copia válida del documento solicitado y se dirige la petición hasta ese nuevo nodo intermedio. Este mensaje indica la recepción de la petición en el nuevo destino, es decir, en el nodo intermedio, cuando se verifica que verdaderamente tiene una copia válida del documento solicitado en su caché, como se había estimado. Las siglas RP indican *Routing Protocol*, es decir, protocolo de encaminamiento, de forma que pueda ser diferenciado del acierto de intercepción presentado en el apartado anterior.

Formato

%f$_1$ %d$_2$ CNA - **%d$_3$:%d$_4$** - recv - CLIENT - RP INTERCEPTION HIT ID: **%d$_5$**

- 1. *currentTime*: instante de tiempo en segundos en el que se recibe este mensaje en el nodo actual, nodo hacia el que se ha dirigido la petición por tener una copia válida del documento solicitado en su caché.
- 2. *idCurrentNode*: identificador del nodo actual que recibe la petición GET y que es capaz de servirla puesto que tiene una copia válida del documento en caché local.

- 3. *idSourceNode*: identificador del nodo que realizó la petición GET que está siendo interceptada.
- 4. *idDestinationNode*: identificador del nodo destino hacia el que se ha dirigido la petición y que se trata del nodo cliente actual, puesto que se ha producido una intercepción con recursos de encaminamiento.
- 5. *idDocumentRequested*: identificador del documento que había sido solicitado en la petición GET interceptada y para la que se encuentra en el nodo actual una copia válida.

Ejemplo

8699.075789 08 CNA - 13:08 - recv - CLIENT - RP INTERCEPTION HIT ID: 14

Indica que el nodo 8, que es un nodo cliente, ha recibido en el instante 8699.075789 (en segundos) una petición GET realizada por el nodo 13, nodo origen de la petición. Este nodo 13 ha descubierto durante la búsqueda de ruta que el nodo 8 posee una copia válida del documento con identificador 14 en su caché local. Entonces, este nodo destino 8 es capaz de responder a la petición sin necesidad de retransmitirla.

A.2.3.5. Redirección de una petición GET

Representa la redirección de la petición GET de un documento. Es decir, cuando se produce el envío de un mensaje GET, el nodo que solicita el documento o bien un nodo intermedio en el camino hacia el servidor de datos redirige la petición hacia otro nodo. La redirección se realiza porque se considera que este nuevo nodo posee una copia válida del documento en su caché local gracias a la información que se almacena de la situación de algunos documentos diseminados por la red en la caché de redirecciones. Además, se debe cumplir que el nodo hacia el que se redirecciona está a menor distancia que el servidor de datos o nodo cliente hacia el que iba dirigida originalmente la petición. Cuando este nuevo nodo recibe la petición y tiene una copia válida del documento en su caché local se produce un acierto de redirección, y se manifiesta con este mensaje.

Formato

%f$_1$ %d$_2$ CNA - **%d$_3$:%d$_4$** - recv - CLIENT - REDIRECTION HIT ID: **%d$_5$**

- 1. *currentTime*: instante de tiempo en segundos en el que se recibe la petición en el nodo hacia el que se ha dirigido la petición por tener una copia válida del documento solicitado.

- 2. *idCurrentNode*: identificador del nodo actual que recibe la petición GET y que es capaz de servirla puesto que tiene una copia válida en caché local.
- 3. *idSourceNode*: identificador del nodo que realizó la petición GET que es redireccionada.
- 4. *idDestinationNode*: identificador del nodo destino hacia el que se ha redirigido la petición, que coincide por tanto con el nodo actual, y que posee una copia del documento solicitado, produciendo un acierto de redirección.
- 5. *idDocumentRequested*: identificador del documento que había sido solicitado en la petición GET redireccionada y para el que se encuentra en el nodo actual una copia válida.

Ejemplo

8701.462867 12 CNA - 10:12 - recv - CLIENT - REDIRECTION HIT ID: 57

Indica que el nodo 12, que es un nodo cliente, ha recibido en el instante 8701.462867 (en segundos) una petición GET realizada por el nodo 10, nodo origen de la petición. Este nodo 10 ha enviado la petición hacia un servidor de datos o hacia un nodo cliente si se tiene activado el mecanismo de intercepción con recursos de encaminamiento pero, o bien él mismo, o bien un nodo en el camino que estaba siguiendo, ha identificado que existía una copia válida del documento con identificador 57 en el nodo 12, pasando a ser el nuevo destino de la petición.

A.2.3.6. Error de redirección

Representa un error en la redirección de la petición GET de un documento. Es decir, cuando se produce el envío de un mensaje GET, el nodo que solicita el documento o bien un nodo intermedio en el camino hacia el servidor de datos o hacia un nodo cliente si se tiene activado el mecanismo de intercepción con recursos de encaminamiento, redirige la petición hacia otro nodo. Cuando este nuevo nodo, que es un nodo cliente, recibe la petición pero no tiene una copia válida del documento en su caché local, se produce un error de redirección, y se manifiesta con este mensaje. A continuación, este nodo propagará por la red hacia el nodo que solicitó el documento un mensaje de error.

Formato

%f$_1$ %d$_2$ CNA - **%d$_3$:%d$_4$** - recv - CLIENT - REDIRECTION ERROR ID: **%d$_5$**

- 1. *currentTime*: instante de tiempo en segundos en el que se recibe la petición en el nodo hacia el que se ha redireccionado la petición y se detecta que no se tiene una copia válida del documento solicitado.
- 2. *idCurrentNode*: identificador del nodo actual que recibe la petición GET y que no tiene una copia válida del documento solicitado en su caché local.
- 3. *idSourceNode*: identificador del nodo que realizó la petición GET que es redireccionada erróneamente.
- 4. *idDestinationNode*: identificador del nodo destino hacia el que se ha redirigido la petición, que coincide por tanto con el nodo actual, y que no posee una copia del documento solicitado, produciendo un error de redirección.
- 5. *idDocumentRequested*: identificador del documento que había sido solicitado en la petición GET redireccionada y para el que no se encuentra en el nodo actual una copia válida.

Ejemplo

1715.428590 04 CNA - 18:04 - recv - CLIENT - REDIRECTION ERROR ID: 31

Indica que el nodo 4, que es un nodo cliente, ha recibido en el instante 1715.428590 (en segundos) una petición GET realizada por el nodo 18, nodo origen de la petición. Este nodo 18 ha enviado la petición hacia el servidor de datos o hacia otro nodo cliente si se tiene activado el mecanismo de intercepción con recursos de encaminamiento pero, o bien él mismo, o bien un nodo en el camino hacia el servidor o nodo cliente, ha identificado que existía una copia válida del documento con identificador 31 en el nodo 4, pasando a ser el nuevo destino de la petición. Pero al recibir la petición, el nodo 4 no posee una copia válida del documento, por lo que se produce el error de redirección.

A.2.3.7. Recepción de un error de redirección en el nodo originario de la petición

Representa la recepción de un error en la redirección de la petición GET del documento solicitado por este nodo cliente. Es decir, cuando se produce el envío de un mensaje GET, este nodo que solicita el documento o bien un nodo intermedio en el camino hacia el servidor de datos o hacia un nodo cliente si se tiene activado el mecanismo de intercepción con recursos de encaminamiento, redirige la petición hacia otro nodo. Cuando este nuevo nodo, que es un nodo cliente, recibe la petición pero no tiene una copia válida del documento en su caché local se produce un error de redirección, que se retransmite hacia el nodo que solicitó el documento. A través de este mensaje se manifiesta cuándo es recibido

este error en el nodo que realizó la petición, e implica además la transmisión de una nueva petición del mismo documento por la red.

Formato

%f$_1$ %d$_2$ CNA - **%d$_3$:%d$_4$** - recv - CLIENT - ERROR_GET RECEIVED ID: **%d$_5$**

- 1. *currentTime*: instante de tiempo en segundos en el que se recibe el error de redirección y se produce la retransmisión de la petición del mismo documento.
- 2. *idCurrentNode:* identificador del nodo actual que realizó la petición GET y que ha recibido el error de redirección.
- 3. *idSourceNode*: identificador del nodo que realizó la petición GET que ha sido redireccionada erróneamente, que corresponde con el nodo actual.
- 4. *idDestinationNode*: identificador del nodo destino hacia el que se debe dirigir la nueva petición GET tras la recepción de error de redirección.
- 5. *idDocumentRequested*: identificador del documento que había sido solicitado en la petición GET y que ha sufrido el error de redirección.

Ejemplo

3746.094766 16 CNA - 16:24 - recv - CLIENT - ERROR_GET RECEIVED ID: 17

Indica que el nodo 16, que es un nodo cliente, ha recibido en el instante 3746.094766 (en segundos) un error de redirección sufrido en la petición GET que realizó del documento con identificador 17. Tras la indicación del error, el nodo actual transmitirá una nueva petición del mismo documento hacia el nodo 24, en este caso.

A.2.3.8. Retransmisión de un error de redirección en un cliente

Representa la retransmisión de un error recibido desde otro nodo indicando error de redirección. El error de redirección se identifica como un error en la realización de un GET o petición por parte de un nodo. Este error se debe enviar hacia el nodo que solicitó el documento para que sea capaz de detectar el problema y pueda solicitar de nuevo el mismo documento. En este caso, el error de redirección se va enrutando de nodo a nodo, y cada nodo intermedio se limita a propagar el error, pudiendo ser un nodo cliente de la red o bien un nodo servidor de la red que, para esta funcionalidad, haría el papel de cualquier otro nodo cliente de la red. Como se ha dicho anteriormente, este mensaje se envía creando rutas de nodo a nodo, es decir, rutas de un salto. Así, en los identificadores de los nodos origen y destino aparecen el origen y el destino del salto, respectivamente, y no aparece la

ruta completa hacia el nodo que realizó la petición del documento que ha experimentado el error de redirección.

Formato

%f$_1$ %d$_2$ CNA - **%d$_3$:%d$_4$** - recv - CLIENT - ERROR_GET FORWARDED ID: **%d$_5$** - NODE: **%d$_6$**

- 1. *currentTime*: instante de tiempo en segundos en el que se produce la retransmisión del error de redirección recibido.
- 2. *idCurrentNode*: identificador del nodo que retransmite el error de redirección recibido, que corresponde con un nodo cliente, ya que si fuera un servidor se utilizaría el mensaje definido en el apartado A.2.2.3.
- 3. i*dSourceNode*: identificador del nodo actual que produce este mensaje indicando la recepción de un error de redirección que debe ser retransmitido hacia el siguiente nodo.
- 4. *idDestinationNode*: identificador del nodo hacia el que va dirigido este mensaje, que corresponde con un nodo que se encuentra a un salto de éste, ya que este error se transmite de forma especial buscando una ruta para cada salto.
- 5. *idDocumentRequested*: identificador del documento solicitado por el nodo que originó la petición GET que ha experimentado el error de redirección.
- 6. *idNode*: identificador del nodo que realizó la petición GET que ha experimentado el error de redirección.

Ejemplo

3746.082752 17 CNA - 17:15 - recv - CLIENT - ERROR_GET FORWARDED ID: 19 - NODE: 16

Indica que el nodo 17, que es un cliente, ha retransmitido en el instante 3746.082752 (en segundos) el error de redirección que ha experimentado la petición GET realizada por el nodo 16 para la solicitud del documento con identificador 19. Este error debe ser retransmitido desde este nodo origen 17 hasta el nodo destino 15, que se encuentra a un salto de éste.

A.2.3.9. Inicio de estadísticos. Fin de calentamiento de las cachés

Representa el instante a partir del cual se considera que las cachés locales de los nodos tienen la información suficiente para presentar un funcionamiento normal en la red. La

forma de implementarlo se basa en fijar un porcentaje de las peticiones realizadas por un nodo de forma que su caché contenga un cierto número de documentos. A partir de ese momento, para cada nodo se comenzarán a calcular sus estadísticos. Los valores estadísticos serán nulos para instantes inferiores al momento en que se produce este mensaje en un nodo dado, y se podrá identificar este período transitorio con la indicación "*Warming Up Cache*".

Formato

%f$_1$ %d$_2$ CNA - Reset Statistics

- 1. *currentTime*: instante de tiempo en segundos en el que se produce este reinicio de estadísticos.
- 2. *idCurrentNode*: identificador del nodo donde se tendrán en cuenta los estadísticos.

Ejemplo

4001.049680 01 CNA - Reset Statistics

Indica que el nodo 1, ha empezado a calcular en el instante 4001.049680 (en segundos) el valor de sus estadísticos ya que ha completado un cierto porcentaje del total de peticiones que debe realizar a lo largo de la simulación, emulando así el calentamiento de su caché local y lograr obtener un escenario más real.

A.2.4. NIVEL DE PROTOCOLO DE ENRUTAMIENTO

En este apartado se describen los mensajes que suceden en la capa de enlace y, más concretamente, son mensajes que pertenecen a la subcapa encargada de la realización del enrutamiento. Por ello todos estos mensajes aparecen con la etiqueta RP (*Routing Protocol*).

A.2.4.1. Recepción de un mensaje

Representa la recepción de un mensaje, ya sea desde las capas inferiores por ser proveniente del medio físico o desde las capas superiores, como la capa de aplicación, donde se realizan las peticiones de documentos y se encuentran la caché local y la caché de redirecciones. Este mensaje es capaz de identificar si se refiere a la recepción de un mensaje debido a una petición, a una respuesta o a un error.

Formato

%f$_1$ %d$_2$ RP - **%d$_3$:%d$_4$** - recv - METHOD: **%d$_5$** - ID: **%d$_6$** - NODE: **%d$_7$**

- 1. *currentTime*: instante de tiempo en segundos en el que se produce la recepción de este mensaje.
- 2. *idCurrentNode*: identificador del nodo que ha recibido este mensaje en su capa de enlace.
- 3. *idSourceNode*: identificador que corresponde con el nodo que ha solicitado un documento si se trata la recepción de una petición, con el nodo que ha servido una petición si se trata de la recepción de una respuesta, o con el nodo origen del salto que está siguiendo un aviso de error en el caso de producirse un error de redirección, ya que se vio que en ese caso la ruta seguida se obtenía nodo a nodo.
- 4. *idDestinationNode*: identificador del nodo hacia el que se está dirigiendo la petición o respuesta incluida en este mensaje, que corresponde con el nodo al que se le solicita un documento si se trata de la recepción de una petición, con el nodo que ha realizado una petición si se trata de la recepción de una respuesta, o con el nodo destino del salto que está siguiendo un error en el caso de producirse un error de redirección, ya que se vio que en ese caso la ruta seguido se obtenía nodo a nodo.
- 5. *method*: indica si se trata de un mensaje de una petición GET si tiene valor 0, si es un mensaje de una respuesta RESP si tiene valor 1, o si se trata de un mensaje de un error de redirección si tiene valor 2.
- 6. *idDocumentRequested*: identificador del documento solicitado por la petición GET que ha causado la generación de este mensaje.
- 7. *idNode*: identificador del nodo que ha realizado la petición GET que ha causado la generación de este mensaje.

Ejemplos

Se van a proponer algunos ejemplos de las varias posibilidades que se pueden encontrar:

16968.497143 04 RP - 23:00 - recv - METHOD: 0 - ID: 0 - NODE: 23

Indica que el nodo 4 ha recibido en el instante 16968.497143 (en segundos) la retransmisión de la petición GET del documento con identificador 0 que ha realizado el nodo 23. Además, indica que el origen de esta petición es el nodo 23, que es el nodo que solicitó el documento, y que el destino es el nodo 0, ya sea un servidor de datos o bien un

nodo cliente que es capaz de servir el documento si se ha producido una intercepción con recursos de encaminamiento o una redirección.

1261.623379 18 RP - 00:21 - recv - METHOD: 1 - ID: 89 - NODE: 21

Indica que el nodo 18 ha recibido en el instante 1261.623379 (en segundos) la retransmisión de la respuesta RESP a una petición GET del documento con identificador 89 que ha realizado el nodo 21. Además, indica que el origen de esta respuesta es el nodo 0, que puede ser o un servidor de datos o bien un nodo cliente que ha sido capaz de servir el documento si se ha producido una intercepción con recursos de encaminamiento o una redirección, y que el destino hacia el que se envía es el nodo 21, que es el nodo que realizó la petición.

2311.888884 03 RP - 03:20 - recv - METHOD: 2 - ID: 50 - NODE: 02

Indica que el nodo 3 ha recibido en el instante 2311.888884 (en segundos) la retransmisión de un error de redirección producido en una petición GET del documento con identificador 50 que ha realizado el nodo 2. Además, a diferencia de los dos casos anteriores, durante la retransmisión del error, el origen y destino del mismo se van obteniendo salto a salto. Debido a esto, el nodo origen que busca ruta para enviar el error es con el nodo actual, el nodo 3, y el nodo hacia el que se retransmitirá el error es el nodo 20. Este error se irá retransmitiendo por la red hasta que sea recibido en el nodo 2.

A.2.4.2. Redirección de una petición a nivel RP

Representa la redirección de un mensaje de petición GET que se acaba de recibir en un nodo y para el que se ha estimado que se tiene una copia en otro nodo diferente al destino de la petición a una distancia en saltos inferior. Esta estimación se realiza gracias a la consulta en la caché de redirecciones a nivel de aplicación. Además, la redirección se realizará únicamente si existe ruta entre el nodo actual y el nodo al que se redirige la petición. La redirección se puede realizar tanto en el mismo nodo que realiza la petición GET antes de enviarla por la red o bien en un nodo intermedio del camino que siga la petición hacia su destino.

Formato

%f$_1$ %d$_2$ RP - **%d$_3$:%d$_4$** - Redirection Only With Route - ID: **%d$_5$** - DEST: **%d$_6$**

- 1. *currentTime*: instante de tiempo en segundos en el que se produce la redirección de la petición GET.

- 2. *idCurrentNode*: identificador del nodo que ha recibido este mensaje en su capa de enlace.
- 3. *idSourceNode*: identificador del nodo que ha originado la petición GET que va a ser redireccionada.
- 4. *idDestinationNode*: identificador del nodo destino hacia el que se estaba dirigiendo la petición GET, que corresponde con el nodo que en principio iba a servir la petición, ya fuera un nodo servidor de datos o bien un nodo cliente si se hubiera producido una intercepción con recursos de encaminamiento.
- 5. *idDocumentRequested*: identificador del documento solicitado por la petición GET que ha causado la generación de este mensaje.
- 6. *idNode*: identificador del nuevo nodo destino hacia el que se ha redireccionado la petición GET que ha causado la generación de este mensaje.

Ejemplos

Se van a proponer algunos ejemplos de las varias posibilidades que se pueden encontrar:

34.493217 17 RP - 17:24 - Redirection Only With Route - ID: 91 - DEST: 16

Indica que el nodo 17 ha recibido en el instante 34.493217 (en segundos) la petición GET a transmitir del documento con identificador 91 que ha realizado el nodo origen 17 y que debía transmitir, en un principio, hacia el nodo destino 24. En este caso la redirección se produce en el mismo nodo que origina la petición, y se realiza gracias a que en su caché de redirecciones viene indicado que el nodo 16 posee una copia válida en su caché local del documento 91, por lo que se redirige la petición hacia él, siempre y cuando exista ruta desde este nodo 17.

219.372092 17 RP - 15:00 - Redirection Only With Route - ID: 0 - DEST: 9

Indica que el nodo 17 ha recibido en el instante 219.372092 (en segundos) la petición GET a retransmitir del documento con identificador 0 que ha realizado el nodo origen 15 y que debía transmitir, en un principio, hacia el nodo destino 0, ya fuera un nodo servidor de datos o un nodo cliente si se hubiera producido una intercepción con recursos de encaminamiento. En este caso la redirección se produce en un nodo intermedio hacia el que se ha enrutado la petición, y se realiza gracias a que en su caché de redirecciones viene indicado que el nodo 9 posee una copia válida en su caché local del documento 0, por lo que se redirige la petición hacia él, siempre y cuando exista ruta desde este nodo 17.

A.2.4.3. Eliminación de mensajes del *buffer* debidos a una respuesta de ruta

Representa la situación en que el protocolo de encaminamiento implementado recibe un mensaje de respuesta de ruta (en el caso de AODV un RREP) notificando que se posee una ruta hacia un determinado nodo. Cuando consulta en su *buffer* de almacenamiento si existen mensajes que debían ser transmitidos hacia ese mismo nodo destino, estos son eliminados, desechando los que estén caducados y cursando por la ruta obtenida aquellos que aún no hayan expirado. Se tendrá un mensaje de este tipo para cada uno que no pueda ser retransmitido por estar caducado.

Formato

%f$_1$ %d$_2$ RP - BUFFER TIMEOUT Reply: **%f$_3$** - ID: **%d$_4$ - %d$_5$:%d$_6$**

- 1. *currentTime*: instante de tiempo en segundos en el que se produce este mensaje.
- 2. *idCurrentNode*: identificador del nodo actual.
- 3. *replyInstant*: instante en el que se produjo la petición GET para la que no se tenía ruta al retransmitir por la red, ya fuera durante el envío de la petición como de la respuesta.
- 4. *idDocument*: identificador del documento solicitado por la petición GET que ha causado la generación de este mensaje.
- 5. *idSourceNode*: identificador del nodo que ha realizado la petición GET que ha originado la creación de este mensaje.
- 6. *idDestinationNode*: identificador del nodo hacia el que iba dirigida la petición GET que ha originado la creación de este mensaje.

Ejemplo

24.559327 00 RP - BUFFER TIMEOUT Reply: 21.478519 - ID: 10 - 81:00

Indica que el nodo 0 ha recibido en el instante 24.559327 (en segundos) un mensaje de respuesta de ruta hacia el nodo destino 0 (como se ha utilizado AODV sería un RREP), por lo que posee una ruta válida hasta él. A continuación, ha consultado su *buffer* a nivel RP para eliminar aquellos mensajes que debería haber retransmitido hacia el nodo 0 y para los que no tenía una ruta válida, desechando los que estén caducados y enviando aquellos que no están expirados. En particular, este mensaje indica que se ha eliminado la petición/respuesta del documento con identificador 10 que había sido generada en el instante 21.478519 por el nodo origen 81 y que iba dirigida al nodo destino 0.

A.2.4.4. Eliminación de mensajes del *buffer* debidos a una petición de ruta

Representa la situación en que el protocolo de encaminamiento implementado recibe un mensaje de petición de ruta (en el caso de AODV un RREQ). Entonces, por razones de eficiencia, se pasa a consultar en su *buffer* de almacenamiento si existen mensajes que no pudieron ser transmitidos hacia el mismo nodo para el que se pide una ruta, eliminando aquellos que estén ya caducados. Se tendrá un mensaje de este tipo para cada uno de los que sean eliminados.

Formato

%f$_1$ %d$_2$ RP - BUFFER TIMEOUT Request: **%f$_3$** - ID: **%d$_4$ - %d$_5$:%d$_6$**

- 1. *currentTime*: instante de tiempo en segundos en el que se produce este mensaje.
- 2. *idCurrentNode*: identificador del nodo actual.
- 3. *requestInstant*: instante en el que se produjo la petición GET para la que no se tenía ruta al retransmitir por la red, ya fuera durante el envío de la petición como de la respuesta.
- 4. *idDocument*: identificador del documento solicitado por la petición GET que ha causado la generación de este mensaje.
- 5. *idSourceNode*: identificador del nodo que ha realizado la petición GET que ha originado la creación de este mensaje.
- 6. *idDestinationNode*: identificador del nodo al que va dirigido la petición GET que ha originado la creación de este mensaje.

Ejemplo

15956.521189 13 RP - BUFFER TIMEOUT Request: 15955.082039 - ID: 34 - 13:0

Indica que el nodo 13 ha recibido en el instante 15956.521189 (en segundos) un mensaje de petición de ruta hacia el nodo destino 0 (como se ha utilizado AODV sería un RREQ). A continuación, ha consultado su *buffer* a nivel RP para eliminar aquellos mensajes que debería haber retransmitido hacia el nodo 0 y para los que no tenía una ruta válida, eliminando los que estén caducados. En particular, este mensaje indica que se ha eliminado la petición/respuesta del documento con identificador 34 que había sido generada en el instante 15955.082039 por el nodo origen 13 y que iba dirigida al nodo destino 0.

A.2.4.5. Retransmisión de un mensaje

Representa la retransmisión de un mensaje hacia el nodo destino que aparece en el propio mensaje. Este mensaje es capaz de identificar si se refiere a la recepción de un mensaje debido a una petición, a una respuesta o a un error. En este mensaje no aparece la ruta completa desde el nodo origen al destino hacia el que va la petición o respuesta, sino que aparece el nodo origen y destino del salto que va a realizar el mensaje. Así, el nodo actual corresponderá siempre con el nodo origen, y el nodo destino corresponderá con el nodo hacia el que será enviado este mensaje, facilitando así la comprensión de los envíos por la red.

Formato

%f$_1$ %d$_2$ RP - **%d$_3$:%d$_4$** - FORWARD - METHOD: **%d$_5$** - ID: **%d$_6$** - NODE: **%d$_7$**

- 1. *currentTime*: instante de tiempo en segundos en el que se produce la retransmisión de este mensaje.
- 2. *idCurrentNode*: identificador del nodo que está indicando este mensaje en su capa de enlace.
- 3. *idSourceNode*: identificador del nodo que va a enviar este mensaje, que corresponde con el nodo actual.
- 4. *idDestinationNode*: identificador del nodo al que se va a enviar este mensaje.
- 5. *method*: indica si se trata de un mensaje de una petición GET si tiene valor 0, si es un mensaje de una respuesta RESP si tiene valor 1, o si se trata de un mensaje de un error de redirección si tiene valor 2.
- 6. *idDocumentRequested*: identificador del documento solicitado por la petición GET que ha causado la generación de este mensaje.
- 7. *idNode*: identificador del nodo que ha realizado la petición GET que ha causado la generación de este mensaje.

Ejemplos

Se van a proponer algunos ejemplos de las varias posibilidades que se pueden encontrar:

2.033935 02 RP - 02:99 - FORWARD - METHOD: 0 - ID: 867 - NODE: 58

Indica que el nodo 2 ha retransmitido en el instante 2.033935 (en segundos) la petición GET del documento con identificador 867 que ha realizado el nodo 58. El nodo 2 podría

ser un cliente o un servidor de datos. Además, indica que el nodo al que se envía este mensaje es el nodo 99.

2.157893 90 RP - 90:45 - FORWARD - METHOD: 1 - ID: 2 - NODE: 05

Indica que el nodo 90 ha retransmitido en el instante 2.157893 (en segundos) la respuesta RESP a una petición GET del documento con identificador 2 que ha realizado el nodo 5. El nodo 90 podría ser un cliente o un servidor de datos. Además, indica que el nodo al que se envía este mensaje es el nodo 45.

1106.084400 80 RP - 80:27 - FORWARD - METHOD: 2 - ID: 107 - NODE: 47

Indica que el nodo 80 ha retransmitido en el instante 1106.0844 (en segundos) el error de redirección que se ha producido en una petición GET del documento con identificador 107 que ha realizado el nodo 47. El nodo 80 podría ser un cliente o un servidor de datos. Además, indica que el nodo al que se envía este mensaje de error es el nodo 27.

A.2.4.6. Intercepción con recursos de encaminamiento de una petición a nivel RP

Representa la intercepción con recursos de encaminamiento presentada en el capítulo 3 de un mensaje de petición GET que acaba de realizar en un nodo y para el que se ha detectado, durante la petición de una ruta hacia el nodo destino, que se tiene una copia válida en otro nodo diferente al destino de la petición a una distancia en saltos inferior. Por tanto este tipo de intercepción sólo se puede dar en el mismo nodo que realiza la petición, a diferencia del otro mecanismo de intercepción remota.

Formato

%f$_1$ %d$_2$ RP – RP Interception - ID: **%d$_3$** - DEST: **%d$_4$**

- 1. *currentTime*: instante de tiempo en segundos en el que se produce la intercepción con recursos de encaminamiento de la petición GET.
- 2. *idCurrentNode*: identificador del nodo que ha indicado en su subcapa de enrutamiento que ha detectado durante la búsqueda de ruta que un nodo cliente más cercano que el nodo destino puede servir su petición.
- 3. *idDocumentRequested*: identificador del documento solicitado por la petición GET que ha causado la generación de este mensaje.

- 4. *idNode*: identificador del nodo hacia el que se debe enviar la petición GET en lugar de su destino original porque tiene una copia válida del documento solicitado en su caché local y se encuentra a menor distancia en saltos.

Ejemplo

97.759592 14 RP - RP Interception - ID: 3 - DEST: 20

Indica que el nodo 14 ha realizado la búsqueda de ruta para enviar petición GET del documento con identificador 3 que ha realizado este mismo nodo. Además, indica que el nuevo destino de la petición es el nodo 20 puesto que ha detectado, en el instante 97.759592 (en segundos), que posee una copia válida de ese documento con identificador 3 y que se encuentra a menor distancia del destino original de la petición.

A.2.5. NIVEL DE APLICACIÓN – MENSAJES DE CACHÉ LOCAL

Los mensajes que se van a mostrar a continuación se dan en la capa de aplicación del nodo y se utilizan para actualizar el contenido de la caché local, que se modificará según la política de reemplazo que esté implementando. La identificación para los mensajes de esta capa contiene las siglas CACHE para poder diferenciarlos del siguiente grupo de mensajes relativos a la caché de redirecciones, que se presentarán más adelante.

A.2.5.1. Inserción de un documento en la caché local

Representa la inserción de un documento en la caché local en la posición indicada y con los parámetros característicos que aparecen en el mensaje. Se trata de un mensaje que, consecuentemente, no produce tráfico en la red.

Formato

$\%f_1$ $\%d_2$ CACHE - INSERT **$\%d_3$: $\%d_4$ $\%d_5$ $\%d_6$ $\%f_7$ $\%f_8$**

- 1. *currentTime*: instante de tiempo en segundos en el que se produce la inserción en caché local.
- 2. *idCurrentNode*: identificador del nodo actual que actualiza su caché local.
- 3. *position*: posición de la memoria caché local donde será insertado el documento. Esto implicará que el resto de documentos que haya en las siguientes posiciones quedarán desplazados una posición.
- 4. *idDocument*: identificador del documento que va a ser almacenado en caché local.

- 5. *sizeDocument*: tamaño en *bytes* del documento que va a ser almacenado en caché local.
- 6. *frequency*: número de veces que ha sido accedido el documento que va a ser almacenado en caché local. Para este mensaje de inserción en caché, la frecuencia inicial viene fijada a 1, que indica la consulta actual de ese documento.
- 7. *evaluationPolicy*: valoración dependiente de la política de reemplazo utilizada.
- 8. *ttlDocument*: representa el tiempo de expiración de la información contenida en el propio documento, es decir, el instante a partir del cual dicha información se vuelve obsoleta.

Ejemplo

1106.365822 95 CACHE - INSERT 0: 462 1000 1 0.000000 3698.160804

Indica que en el instante 1106.365822 (en segundos) de la simulación, el nodo 95 inserta en la posición 0 el documento con identificador 462, que tiene un tamaño en *bytes* de 1000, una frecuencia de acceso de 1, una valoración según la política de reemplazo de valor 0.0, y que expira en el instante 3698.160804, instante a partir del cual la información del documento se considera obsoleta.

A.2.5.2. Cambio de posición de un documento en la caché local

Representa el movimiento o cambio de posición de un documento en la caché local desde su antigua posición hasta la posición indicada. El mensaje añade además algunos de los parámetros característicos que aparecen en el mensaje. Se trata de un mensaje que, consecuentemente, no produce tráfico en la red.

Formato

%f$_1$ %d$_2$ CACHE – MOVE **%d$_3$-%d$_4$: %d$_5$ %d$_6$ %d$_7$ %f$_8$ %f$_9$**

- 1. *currentTime*: instante de tiempo en segundos en el que se produce el cambio de posición del documento en caché local.
- 2. *idCurrentNode*: identificador del nodo actual que actualiza su caché local.
- 3. *oldPosition*: posición de la caché donde se encuentra inicialmente el documento que se debe desplazar.
- 4. *newPosition*: nueva posición de la memoria caché local donde será insertado el documento. Esto implicará que el resto de documentos que haya en las siguientes posiciones quedarán desplazados una posición.

- 5. *idDocument*: identificador del documento que va a ser desplazado en la caché local.
- 6. *sizeDocument*: tamaño en *bytes* del documento que va a ser desplazado en la caché local.
- 7. *frequency*: número de veces que ha sido accedido el documento que va a ser desplazado en la caché local.
- 8. *evaluationPolicy*: valoración dependiente de la política de reemplazo utilizada.
- 9. *ttlDocument*: representa el tiempo de expiración de la información contenida en el propio documento, es decir, el instante a partir del cual dicha información se vuelve obsoleta.

Ejemplo

1106.389164 66 CACHE - MOVE 5-0: 4 1000 2 0.000000 3274.652323

Indica que en el instante 1106.389164 (en segundos) de la simulación, el nodo 66 desplaza hacia la posición 0 el documento con identificador 4 que se encontraba en la posición inicial 5, que tiene un tamaño en *bytes* de 1000, una frecuencia de acceso de 2, una valoración según la política de reemplazo de valor 0.0, y que expira en el instante 3274.652323, instante a partir del cual la información del documento se considera obsoleta.

A.2.5.3. Borrado de un documento de la caché local

Representa la eliminación de un documento que se encuentra en una determinada posición de la caché local. Se trata de un mensaje que, consecuentemente, no produce tráfico en la red.

Formato

%f$_1$ %d$_2$ CACHE – DELETE **%d$_3$: %d$_4$**

- 1. *currentTime*: instante de tiempo en segundos en el que se produce la eliminación del documento en caché local.
- 2. *idCurrentNode*: identificador del nodo actual que actualiza su caché local.
- 3. *position*: posición de la caché donde se encuentra el documento que se debe eliminar.
- 4. *idDocument*: identificador del documento que va a ser eliminado de la caché local.

Ejemplo

1108.594588 87 CACHE - DELETE 95: 410

Indica que en el instante 1108.594588 (en segundos) de la simulación, el nodo 87 elimina de la posición 95 el documento con identificador 410.

A.2.6. NIVEL DE APLICACIÓN – MENSAJES DE CACHÉ DE REDIRECCIONES

Los mensajes que se van a mostrar a continuación se dan en la capa de aplicación del nodo y se utilizan para actualizar el contenido de la caché de redirecciones utilizada para determinar la distancia a la que se encuentra un determinado documento en la red al utilizar el mecanismo de redirección estudiado. La identificación para los mensajes de esta capa contiene las siglas CACHEDEST. Además, esta caché se caracteriza porque se modela como una tabla en la que se guarda la información de dónde se encuentran los documentos en la red, y cada entrada se encuentra dividida en dos registros. El primer registro, denominado registro GET, contiene la información acerca del identificador del nodo que pidió el documento, el número de saltos al que está y el instante de expiración o caducidad del documento. El segundo registro, denominado registro RESP, contiene la información acerca del identificador del nodo que sirvió el documento siempre que sea un nodo cliente (puesto que de antemano los nodos conocen en qué servidor se encuentran los documentos), el número de saltos al que está y el instante de expiración o caducidad del documento.

A.2.6.1. Inserción de la información de un documento en el registro GET de la caché de redirecciones

Representa la inserción en la caché de redirecciones de la información de la situación de un documento en la red gracias a la información que lleva la petición GET de dicho documento. Se trata de un mensaje que, consecuentemente, no produce tráfico en la red.

Formato

%f$_1$ %d$_2$ CACHEDEST - INSERT GET **%d$_3$: %d$_4$ %d$_5$ %f$_6$**

- 1. *currentTime*: instante de tiempo en segundos en el que se produce la inserción en el registro GET en la caché de redirecciones.
- 2. *idCurrentNode*: identificador del nodo actual que actualiza su caché de redirecciones.

- 3. *idDocument*: identificador del documento cuya información de su situación va a ser almacenada en la caché de redirecciones.
- 4. *idNode*: identificador del nodo que ha solicitado el documento y que tendrá una copia del mismo en su caché local.
- 5. *nHops*: distancia en número de saltos a la que se encuentra el nodo que solicitó el documento a través de una petición GET y que por tanto lo tendrá en su caché local.
- 6. *ttlDocument*: representa el tiempo de expiración inicial que se asigna a la información contenida sobre el propio documento. Este valor inicial será siempre 0, pudiendo ser modificado posteriormente cuando este nodo obtenga información del TTL del documento si llega a retransmitir la respuesta a la petición, ya que dicha respuesta contiene al propio documento.

Ejemplo

15954.313373 22 CACHEDEST - INSERT GET 637: 20 2 0.0

Indica que en el instante 15954.313373 (en segundos) de la simulación, el nodo 22 añade en el registro GET destinado a la información del documento con identificador 637 los datos de su localización. Así, el documento se encontrará en el nodo con identificador 20 que se encuentra a 2 saltos, y que el TTL inicial será 0.0.

A.2.6.2. Inserción de la información de un documento en el registro RESP de la caché de redirecciones

Representa la inserción en la caché de redirecciones de la información de la situación de un documento en la red gracias a la información que lleva la respuesta RESP a una petición GET de dicho documento. Se trata de un mensaje que, consecuentemente, no produce tráfico en la red.

Formato

%f$_1$ %d$_2$ CACHEDEST - INSERT RESP **%d$_3$: %d$_4$ %d$_5$ %f$_6$**

- 1. *currentTime*: instante de tiempo en segundos en el que se produce la inserción en el registro RESP en la caché de redirecciones.
- 2. *idCurrentNode*: identificador del nodo actual que actualiza su caché de redirecciones.

- 3. *idDocument*: identificador del documento cuya información de su situación va a ser almacenada en la caché de redirecciones.
- 4. *idNode*: identificador del nodo que ha servido el documento y que tiene una copia del mismo en su caché local.
- 5. *nHops*: distancia en número de saltos a la que se encuentra el nodo que sirvió el documento a través de una respuesta a una petición GET y que por tanto lo tendrá en su caché local.
- 6. *ttlDocument*: representa el tiempo de expiración de la información contenida sobre el propio documento, es decir, el instante a partir del cual dicha información se vuelve obsoleta.

Ejemplo

15960.959771 05 CACHEDEST - INSERT RESP 883: 07 1 16437.550566

Indica que en el instante 15960.959771 (en segundos) de la simulación, el nodo 5 añade en el registro RESP destinado a la información del documento con identificador 883 los datos de su localización. Así, el documento se encuentra en el nodo con identificador 7 que se encuentra a 1 salto, pero que esta información dejará de ser válida a partir del instante 16437.550566 (en segundos).

A.2.6.3. Modificación de la información de expiración de un documento en su registro GET de la caché de redirecciones

Representa la modificación en la caché de redirecciones de la información del instante de expiración del registro GET de un documento en un nodo, que ocurre cuando un nodo retransmite la respuesta a una petición GET que contiene el documento solicitado. Se trata de un mensaje que, consecuentemente, no produce tráfico en la red.

Formato

%f$_1$ %d$_2$ CACHEDEST – MODIFY GET **%d$_3$: %f$_4$**

- 1. *currentTime*: instante de tiempo en segundos en el que se produce la modificación del registro GET en la caché de redirecciones.
- 2. *idCurrentNode*: identificador del nodo actual que actualiza su caché de redirecciones.
- 3. *idDocument*: identificador del documento cuya información de su tiempo de expiración va a ser modificada en la caché de redirecciones.

- 4. *newTtlDocument*: nuevo tiempo de vida del documento que debe ser actualizado en la caché de redirecciones.

Ejemplo

15979.664533 10 CACHEDEST - MODIFY GET 164: 16414.141688

Indica que en el instante 15979.664533 (en segundos) de la simulación, el nodo 10 modifica en el registro GET destinado a la información del documento con identificador 164 los datos de su instante de expiración por el nuevo valor 16414.141688 (en segundos).

A.2.6.4. Borrado del registro GET de una entrada de la caché de redirecciones

Representa el borrado del contenido del registro GET de la entrada relativa a la información de un documento. Se trata de un mensaje que, consecuentemente, no produce tráfico en la red.

Formato

%f$_1$ %d$_2$ CACHEDEST – DELETE GET **%d$_3$**

- 1. *currentTime*: instante de tiempo en segundos en el que se produce el borrado del registro GET en la caché de redirecciones.
- 2. *idCurrentNode*: identificador del nodo actual que actualiza su caché de redirecciones.
- 3. *idDocument*: identificador del documento cuya información de su registro GET va a ser eliminada.

Ejemplo

15984.302095 16 CACHEDEST - DELETE GET 65

Indica que en el instante 15984.302095 (en segundos) de la simulación, el nodo 16 elimina el contenido del registro GET destinado a la información del documento con identificador 65.

A.2.6.5. Borrado del registro RESP de una entrada de la caché de redirecciones

Representa el borrado del contenido del registro RESP de la entrada relativa a la información de un documento. Se trata de un mensaje que, consecuentemente, no produce tráfico en la red.

Formato

%f$_1$ %d$_2$ CACHEDEST – DELETE RESP **%d$_3$**

- 1. *currentTime*: instante de tiempo en segundos en el que se produce el borrado del registro RESP en la caché de redirecciones.
- 2. *idCurrentNode*: identificador del nodo actual que actualiza su caché de redirecciones.
- 3. *idDocument*: identificador del documento cuya información de su registro RESP va a ser eliminada.

Ejemplo

15999.420773 21 CACHEDEST - DELETE RESP 3

Indica que en el instante 15999.420773 (en segundos) de la simulación, el nodo 21 elimina el contenido del registro RESP destinado a la información del documento con identificador 3.

A.2.6.6. Borrado de una entrada de la caché de redirecciones

Representa el borrado del contenido de la entrada relativa a un cierto documento de la caché de redirecciones, lo que implica la eliminación de la información almacenada en sus registros GET y RESP. Se trata de un mensaje que, consecuentemente, no produce tráfico en la red.

Formato

%f$_1$ %d$_2$ CACHEDEST – DELETE ALL **%d$_3$**

- 1. *currentTime*: instante de tiempo en segundos en el que se produce el borrado de la entrada en la caché de redirecciones.
- 2. *idCurrentNode*: identificador del nodo actual que actualiza su caché de redirecciones.

- 3. *idDocument*: identificador del documento cuya información de sus registros GET y RESP va a ser eliminada.

Ejemplo

16001.080334 05 CACHEDEST - DELETE ALL 20

Indica que en el instante 16001.080334 (en segundos) de la simulación, el nodo 5 elimina el contenido de los registros GET y RESP destinados a la información del documento con identificador 20.

A.3. FORMATO DE LAS CABECERAS DE LOS ARCHIVOS

Además de los eventos que provienen de los archivos de entrada de la simulación, tanto eventos de mensajes como eventos de movilidad de los nodos, se exige que determinados valores necesarios para el funcionamiento de la aplicación sean introducidos en dichos ficheros de entrada con el formato que se va a indicar a continuación. Estos valores son necesarios para poder verificar la integridad de los datos que debería manejar la aplicación.

Así, en el archivo de salida de la simulación *.txt* se exige la introducción de los siguientes elementos en líneas diferentes:

- El número total de nodos que interviene en la simulación, considerando tanto los nodos clientes como los nodos servidores de datos. Para introducir este valor es necesario utilizar el formato siguiente, utilizando un número decimal:

  "num_nodes is set **%d**"

- La duración total de la simulación, en segundos. Para introducir este valor es necesario utilizar el formato siguiente, utilizando un número decimal:

  "simulation duration: **%d**"

- El radio de cobertura de los nodos del escenario, en metros. Es importante destacar que este valor no es obligatorio como en los dos casos anteriores, aunque si se desea fijar una cobertura de nodos diferente a 250 metros de radio, que es el valor que la aplicación toma por defecto, se deberá establecer un nuevo valor de esta manera. Para introducir este valor es necesario utilizar el formato siguiente, utilizando un número decimal o en punto flotante:

"coverage radius: **%f**"

Por otra parte, en el archivo de movilidad de los nodos *.pos* se exige la introducción de los siguientes elementos en líneas diferentes:

- El número total de nodos que interviene en la simulación de la movilidad de los nodos, considerando tanto los nodos clientes como los nodos servidores de datos. Para introducir este valor es necesario utilizar el formato siguiente, utilizando un número decimal:

  "nodes: **%d**"

- La duración total de la simulación de la movilidad de los nodos, en segundos. Para introducir este valor es necesario utilizar el formato siguiente, utilizando un número decimal:

  "simulation duration: **%d**"

- Las dimensiones del escenario utilizado para la simulación, para que el software pueda crear las líneas de separación y la rejilla, además de ajustar el *zoom* inicial para que quede el escenario virtual quede completamente dentro de la zona de la ventana principal de la aplicación destinada a ello. Para introducir estos valores es necesario utilizar el formato siguiente, utilizando un número en punto flotante:

  "max x: $\mathbf{\%f_1}$, max y: $\mathbf{\%f_2}$"