UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA EN INFORMÁTICA

PROYECTO FIN DE CARRERA

Extensión de NCTUns 5.0 para simular el entorno inalámbrico y desarrollo del visor de mensajes intercambiados para EVIGEN

Alumna: Lorena González Manzano
Profesor Tutor: José María de Fuentes García-Romero de Tejada

OCTUBRE, 2010

Agradecimientos

Previamente a la descripción de este proyecto es indispensable mencionar a todas aquellas personas que, incondicionalmente, me han ofrecido su apoyo y su comprensión a lo largo de todo el desarrollo.

Antes de comenzar agradecer a todos los lectores de este documento el haber dedicado unas horas de su tiempo en cada una de las páginas.

Primeramente debo agradecer a mi familia, especialmente a mis padres, José y Marivir, y a mi hermana, Cristina, el haberme aguantado y apoyado en cada momento, porque no hay duda que el estrés y la tensión hacen que determinadas situaciones sean tan difíciles que sólo con su ayuda se puedan superar.

Me gustaría hacer una mención muy especial, a mis abuelos, Felisa y Alejandro, por todo el cariño y fuerza que me han dado cada día y que espero que, por muchos años, me sigan dando.

Por otra parte, agradecer a Sergio el haber desarrollado su proyecto en cooperación con el presentado en este documento.

Finalmente, no puedo dejar de mencionar a José María (Chema), tutor de este proyecto, sin el cual no se podría haber llevado a cabo y al que he de agradecer todo el entusiasmo, dedicación, rigidez y ánimo prestando.

Resumen

Hoy en día son innumerables los avances tecnológicos, incluso han llegado hasta al campo automovilístico, pero el desarrollo tecnológico en este ámbito es más pausado y costoso que en muchos otros. Una de las principales causas de tales hechos es el coste en la realización de dispositivos innovadores, cuya integración en un vehículo está asociada a un elevado coste económico. Por otro lado, también hay que tener presente la infraestructura en la que realizar las pruebas de cada una de las innovaciones propuestas, ya que la creación del entorno adecuado dista de ser gratuita.

Motivados por el coste de desarrollo de nuevo avances en el mundo del automóvil, se presentan los simuladores, facilitando la creación y posterior experimentación con distintos vehículos, distintos entornos y distintos modelos de comportamiento, de tráfico, etc. Existen gran variedad de simuladores que proporcionan extensas posibilidades de juego, pero no en todos se puede simular aquello que se desee.

El presente proyecto, atendiendo a la falta de flexibilidad en muchos simuladores, se basa en la implementación del protocolo EVIGEN para integrarlo como nuevo modelo de comportamiento de un simulador, en concreto, en el simulador NCTUns 5.0 (1).

Considerando todo lo comentado en párrafos anteriores, los objetivos de este proyecto son:

- Implementación del protocolo EVIGEN, entorno inalámbrico, como nuevo modelo de comportamiento incorporado al simulador NCTUns 5.0 (1).
- Desarrollo para la visualización de los mensajes intercambiados en el protocolo EVIGEN.

Índice de contenidos

Índice de tablas

Índice de ilustraciones

1 Introducción

Actualmente muchos son los adelantos tecnológicos realizados en todo tipo de campos, la tecnología inunda casas, colegios, empresas, etc. ¿Quién no dispone de internet? o ¿quién no ha realizado alguna vez fotos con un teléfono móvil?, todo en nuestro alrededor está bañado por aires tecnológicos.

Sin embargo, a pesar de todos los grandes avances existentes y otros muchos en proceso de desarrollo, los vehículos han permanecido al margen de gran parte de todo ello, aunque sí es cierto que desde los comienzos del automóvil muchas han sido las mejoras realizadas. Tras la creación del coche de caballos, de vapor y de motor de combustión impulsado por aceite, llega el vehículo de motor impulsado por gasolina, el cual, aunque con considerables mejoras, se corresponde con el coche que conocemos hoy en día.

Del mismo modo, aunque de forma mucho más abrumadora, ha sido el avance producido en las tecnologías de la información, cada día se promueven nuevas creaciones, como el desarrollo de un ratón táctil o la implantación de un nuevo buscador que aumenta la seguridad, como ocurrió con Google SSL.

Por otro lado, aunque son indiscutibles los grandes progresos realizados en el mundo de las tecnologías de la información, ¡ahora toca el turno de los vehículos!. Los motivos son completamente sustanciales ya que se precisa mejorar muchos aspectos relacionados con la conducción, siendo unos de los prioritarios, el aumento de la seguridad. Muchas de las muertes producidas en carretera podrían haberse evitado si el vehículo hubiese facilitado información sobre el estado de la misma, advirtiendo de una carretera deslizante por la abundancia de agua o de un vehículo estacionado por avería. Además, también podría informarse, no sólo de datos relativos a la seguridad vial o al tráfico, sino también datos de carácter puramente informativo como por ejemplo la proximidad de un centro de ocio o de un restaurante.

No obstante, la creación de un dispositivo para un automóvil requiere una gran inversión, tanto en coste de infraestructura como en coste asociado al entorno de desarrollo. Es por este motivo por el que muchas de las propuestas, por ejemplo el establecimiento en cada vehículo de un pequeño chip que permita la identificación del mismo, no son llevadas a cabo, sino que únicamente se desarrollan de forma experimental. De hecho, muchas de las propuestas son realizadas por los investigadores de las universidades, mediante la utilización de simuladores que faciliten y reduzcan el coste asociado.

Considerando lo comentado en relación con el campo automovilístico, los simuladores son esenciales en el mundo empírico, de modo que permiten probar modelos de comportamiento, de tráfico, etc., también denominados protocolos, que establecidos en vehículos reales, serían excesivamente costosos. Sin embargo, existen gran cantidad de simuladores, unos orientados a la creación de modelos de movimiento, otros enfocados en la generación de modelos de tráfico, etc., además de que unos son de libre

distribución y otros comerciales. Por ello, no todos los simuladores proporcionan las mismas funcionalidades y, por el mismo motivo, no todos permiten la integración del protocolo deseado.

Cierto es que la utilización de simuladores es una buena práctica pero realizar ensayos sobre la adecuación de un protocolo en distintos escenarios, pero no es tarea fácil. Por un lado, se requiere el completo conocimiento del funcionamiento del simulador escogido y, tras esto, se necesita que dicho simulador permita llevar a cabo la implantación del desarrollo deseado.

En conclusión, muchos de los simuladores existentes en la actualidad aportan grandes funcionalidades y son fáciles de utilizar, sin embargo, la integración de cualquier tipo de protocolo no es siempre viable, siendo éste el principal objetivo del presente proyecto, la integración del protocolo EVIGEN como nuevo modelo de comportamiento asociado al simulador NCTUns 5.0 (1).

1.1 Objetivos del presente proyecto

El proyecto que se desea realizar consta del análisis y estudio del simulador NCTUns 5.0 (1) para la inclusión del protocolo EVIGEN (el cual se explica en la sección 2.2) dentro de los modelos de comportamiento proporcionados por el mismo. Además, se pretende la creación de un desarrollo que posibilite la visualización de los mensajes intercambiados en dicho protocolo.

Las funcionalidades que permiten la simulación de EVIGEN son demasiado amplias como para realizar una completa implementación del mismo en este proyecto. Por este motivo, el presente proyecto se desarrolla en paralelo a otro complementario (2).

La Ilustración 1 refleja la funcionalidad global que se pretende conseguir a través de ambos proyectos, integración de EVIGEN en el simulador NCTUns 5.0 (1), visualización de los mensajes intercambiados y generación de las estadísticas asociadas al funcionamiento de dicho protocolo. En concreto, en el presente proyecto se desarrollan los módulos sombreados de morado, correspondientes a parte de la integración del protocolo EVIGEN y a la visualización de los mensajes, mientras que en el complementario se abordan los módulos destacados en color gris, asociados a la integración de la parte restante del protocolo, al desarrollo de un módulo criptográfico y a la creación de estadísticas basadas en los resultados obtenidos.

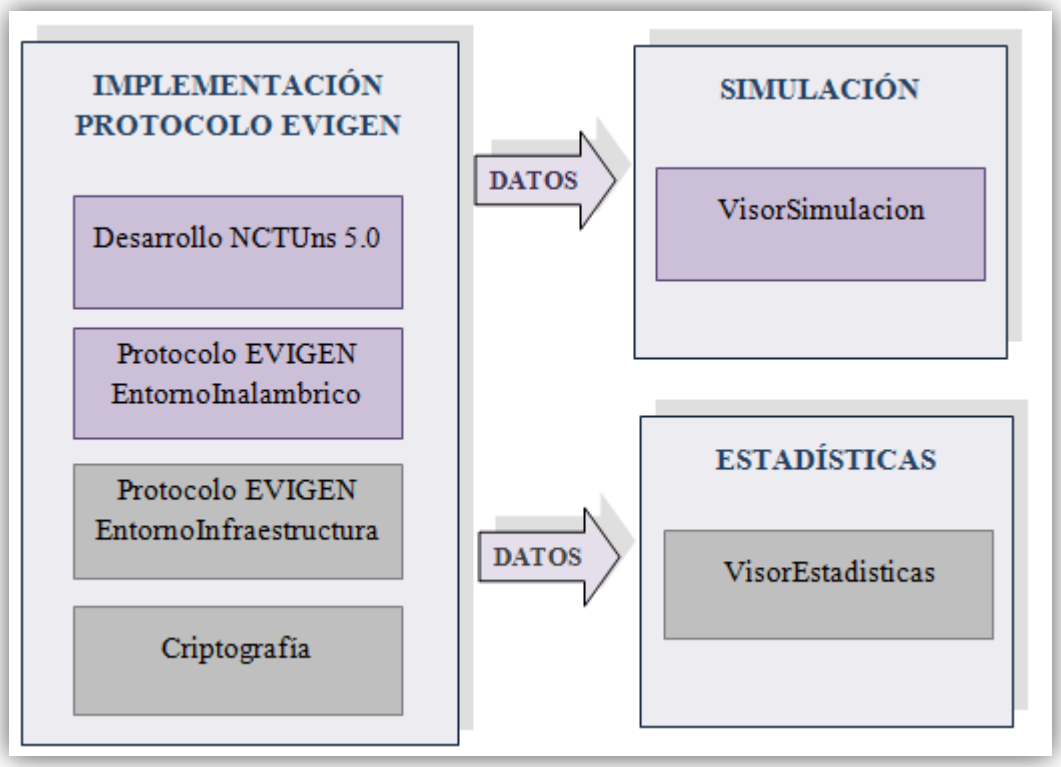


Ilustración 1: extensión simulador NCTUns, objetivos.

Del lado de la implementación del protocolo EVIGEN se va a realizar la integración de dicho protocolo en dos módulos, "ProtocoloEVIGEN EntornoInalámbrico" y

"ProtocoloEVIGEN EntornoInfraestructura", así como la asociación de secciones previamente implementadas en el simulador NCTUns 5.0(1), "Desarrollo NCTUns 5.0".

Asimismo, para el correcto desarrollo del protocolo es imprescindible hacer uso del módulo "Criptografia" (desarrollado en el proyecto complementario (2)), en el que se proporcionan operaciones criptográficas aplicables a los mensajes intercambiados en EVIGEN.

Por otro parte, se desean visualizar los mensajes intercambiados en el protocolo para su posterior análisis, módulo "VisorSimulacion".

Finalmente, subrayar la existencia de un flujo de datos desde la propia implementación del protocolo EVIGEN al desarrollo creado para la visualización de los mensajes intercambiados, de modo que dichos mensajes puedan ser representados.

Considerando lo mencionado, los objetivos de este proyecto son:

- Implementación del protocolo EVIGEN, entorno inalámbrico, como nuevo modelo de comportamiento incorporado al simulador NCTUns 5.0 (1).
- Desarrollo para la visualización de los mensajes intercambiados en el protocolo EVIGEN.

1.2 Organización del presente documento

Este documento se compone de ocho secciones y cinco anexos complementarios (3):

1. Introducción: en esta sección se describe la motivación del presente proyecto y se presentan los objetivos perseguidos.

2. Contexto: redes vehiculares y protocolo EVIGEN: en esta sección se describen las principales características de las redes vehiculares y se desarrolla el funcionamiento del protocolo EVIGEN en su totalidad.

3. Análisis: en esta sección se estudia la situación actual y los elementos disponibles en la actualidad para abordar cada uno de los objetivos planteados. Además, se establece la arquitectura del sistema, detallando los componentes requeridos. Asimismo, se especifican los requisitos del sistema y el plan de pruebas que garantice el cumplimiento de cada uno de dichos requisitos.

4. Diseño: en esta sección se especifican todos los componentes del sistema, profundizando en el diseño de clases y funciones o métodos comprendidos en cada una de ellas. Del mismo modo, se detallan los diagramas de secuencia oportunos para conseguir comprender las interacciones efectuadas entre las clases desarrolladas.

5. Implementación: en esta sección se exponen los detalles de implementación y los resultados de las pruebas de aceptación.

6. Conclusiones y líneas futuras: en esta sección se exponen las conclusiones obtenidas tras la finalización del presente proyecto y las vías que aún pueden ser exploradas y desarrolladas en proyectos futuros.

7. Bibliografía y referencias: listado de las fuentes utilizadas a lo largo del desarrollo de este proyecto.

8. Glosario de términos: descripción de las definiciones y acrónimos, característicos de este proyecto, los cuales pueden ser consultados a modo informativo.

En al documento asociado (3) al presente proyecto se sitúan los siguientes anexos:

1. ANEXO I: Pruebas de aceptación: especificación de las pruebas de aceptación que verifican la satisfacción de los requisitos.

2. ANEXO II: Evaluación del sistema: exposición de simulaciones realizadas para evaluar el correcto funcionamiento de la extensión de NCTUns para la simulación del protocolo EVIGEN y del sistema de visión de mensajes de la simulación.

3. ANEXO III: Gestión del proyecto: en esta sección se realiza la descripción de la planificación del proyecto, así como de las herramientas disponibles para llevarlo a cabo y los costes asociados al mismo.

4. ANEXO IV: implantación del software: descripción del proceso de arranque y puesta en marcha del sistema.

5. ANEXO V: Análisis de seguridad: especificación del análisis de seguridad de aspectos relativos al intercambio de mensajes del protocolo.

2 Contexto: redes vehiculares y protocolo EVIGEN

Especificados los objetivos del proyecto y teniendo en cuenta que el protocolo es utilizado para simular un determinado comportamiento mediante la comunicación entre distintos vehículos y en distintos entornos, lo cual comprende una red vehicular, en esta sección se exponen los fundamentos básicos de las redes vehiculares (sección 2.1) y la descripción del protocolo EVIGEN en su totalidad (sección 2.2), incluyendo las partes "ProtocoloEVIGEN EntornoInalambrico" y "ProtocoloEVIGEN EnotnoInfraestructura" presentadas en la sección de objetivos (sección 1.1).

Previamente al desarrollo de la sección, es imprescindible señalar que ha sido realizada conjuntamente con el autor del proyecto complementario (2).

2.1 Redes vehiculares

Dado que el protocolo EVIGEN se desarrolla sobre una red vehicular, en esta sección se describen los fundamentos de este tipo de redes de comunicación.

Principalmente, en una red vehicular se distinguen dos tipos de nodos, los vehículos, nodos móviles, y las RSU (*Road Side Units*), nodos fijos. Por un lado, los vehículos constan de un pequeño hardware integrado, denominado OBU (*On Board Side Unit*), encargado de la comunicación *wireless* con otros vehículos. Por otro lado, las RSU, dispuestas en puntos críticos a lo largo de las carreteras, se responsabilizan del envío de información hacia y entre los vehículos.

Además, en este tipo de infraestructuras también pueden existir otro tipo de nodos, como son las autoridades de certificación. Uno de los papeles fundamentales de entidades como éstas es el envío de CRL a las distintas RSU, para que estas últimas sean las encargadas de reenviarlas a los vehículos y, de este modo, aumentar la seguridad puesto que se descartan los mensajes procedentes de un vehículo con certificado revocado.

Otro de los aspectos a considerar en las redes vehiculares es la recepción de la información del entorno, de modo que tanto vehículos como RSU dispongan de información de su entorno próximo. Esto se realiza mediante el envío de mensajes de *beaconing* a los vehículos o RSU que se encuentran situados en un determinado radio de cercanía, lo cual queda reflejado en la Ilustración 2.

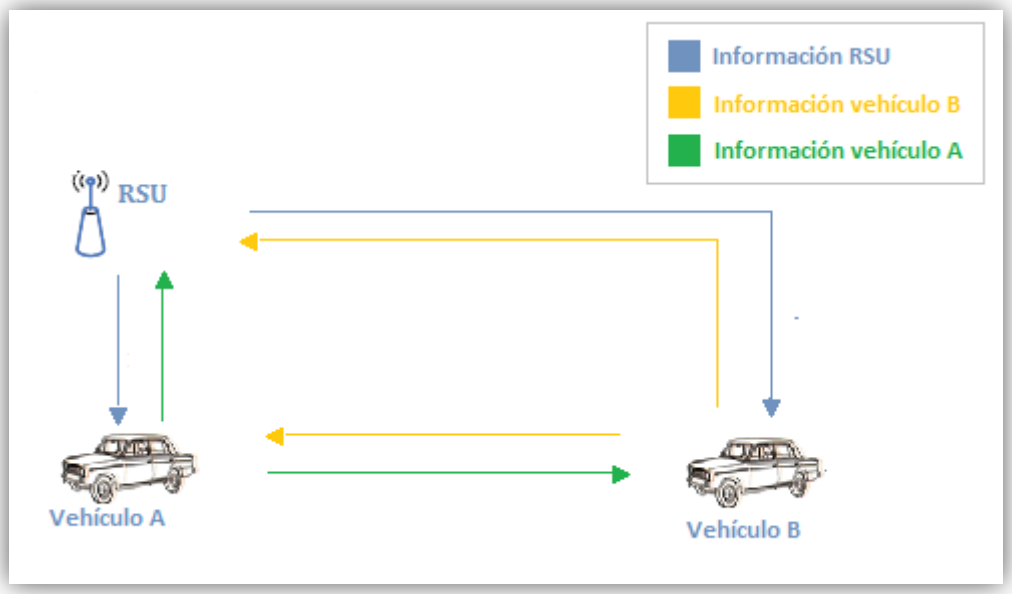


Ilustración 2: envío mensajes de *beaconing*.

2.2 Descripción del protocolo EVIGEN implementado en este proyecto

El propósito del protocolo EVIGEN (cuyo nombre procede de *EVIdence GENeration*) es el de de permitir la creación de pruebas electrónicas en el contexto del procedimiento sancionador que permitan describir el comportamiento del vehículo en el momento al que se refiere la sanción. Para conseguir este objetivo, la aproximación de EVIGEN se basa en la comunicación del vehículo sancionado con otros cercanos, los cuales puedan emitir una descripción ("testimonio") de dicho comportamiento. Como ejemplo de aplicación, este protocolo se podría ejecutar si un vehículo recibiera una notificación de sanción por exceso de velocidad, el cual es el enfoque en el que se basa la descripción del protocolo y el que sustenta el desarrollo de este proyecto.

2.2.1 Roles participantes

Teniendo en cuenta los distintos nodos que pueden existir en una red vehicular (introducidos en la sección 2.1), en EVIGEN se identifican distintos roles para cada uno de ellos. En particular los nodos móviles pueden tomar tres roles distintos: Requester, Witness y NoEquipped, mientras que los nodos fijos se corresponden con RSU, AC y DGT. Es conveniente indicar que un vehículo, en un primer momento, sólo puede ser Witness o NoEquipped, pasando, conjuntamente, a tomar el rol de Requester en el momento de recepción de un mensaje procedente de una RSU, en este caso correspondiente con una notificación de sanción.

Previamente a la descripción del protocolo y con el propósito de facilitar la comprensión del mismo, se presentan las funciones asociadas a cada uno de los roles:

- Requester: encargado de cuestionar a los Witness, que se encuentren dentro de su radio de cercanía, sobre la velocidad a la que iba en un determinado momento. Tras recibir y procesar los testimonios recibidos, ha de construir la evidencia y, en caso de que la velocidad resultante sea inferior a la que fue sancionado, debe enviar dicha evidencia a la RSU más cercana para que posteriormente sea recibida en la DGT.
- Witness: encargado de crear los testimonios de acuerdo con una solicitud realizada por un Requester. El testimonio creado puede ser enviado al propio Requester o, en caso de no encontrarse éste lo suficientemente cerca, se enviaría a nodos con rol NoEquipped considerados cercanos o, de no ser posible, a nodos con rol RSU. Teóricamente, se corresponde con un nodo con sensores capaces de detectar los acontecimientos ocurridos en un determinado radio.
- NoEquipped: encargado de realizar reenvíos de testimonios para que el Requester pertinente pueda recibirlo. En caso de no poder enviarlo al Requester, por no encontrarse cerca, se enviaría a nodos con rol NoEquipped considerados cercanos o, de no ser posible, a nodos con rol RSU. Teóricamente, representa a aquellos vehículos que no disponen de información sensorial útil para el

desarrollo del protocolo pero que, sin embargo, están equipados con una unidad de comunicaciones a bordo (OBU).

- RSU: encargada de la detección de infracciones cometidas por vehículos, nodos móviles, y de su notificación a la DGT. Además, actúan de intermediarios de los mensajes que transmiten los distintos participantes del protocolo. Teóricamente representan a las unidades que se encuentran bajo el control del gobierno, situadas a lo largo de las carreteras.
- DGT: encargada de la emisión de las sanciones de los vehículos, de la verificación de las evidencias que demuestran la inocencia de los mismos, y de determinar los vehículos cuyo certificado se debe revocar. Teóricamente representa una autoridad legal.
- AC: encargada de la gestión de la CRL y de su transmisión a las RSU. Teóricamente asume el rol de la autoridad de certificación emisora de los certificados de cada una de las entidades del protocolo.

2.2.2 Descripción básica de los mensajes intercambiados

Con el propósito de clarificar el intercambio de mensajes producido, la descripción del protocolo se va a realizar mediante la creación del entorno más simple, el cual se presenta posteriormente, en la Ilustración 3.

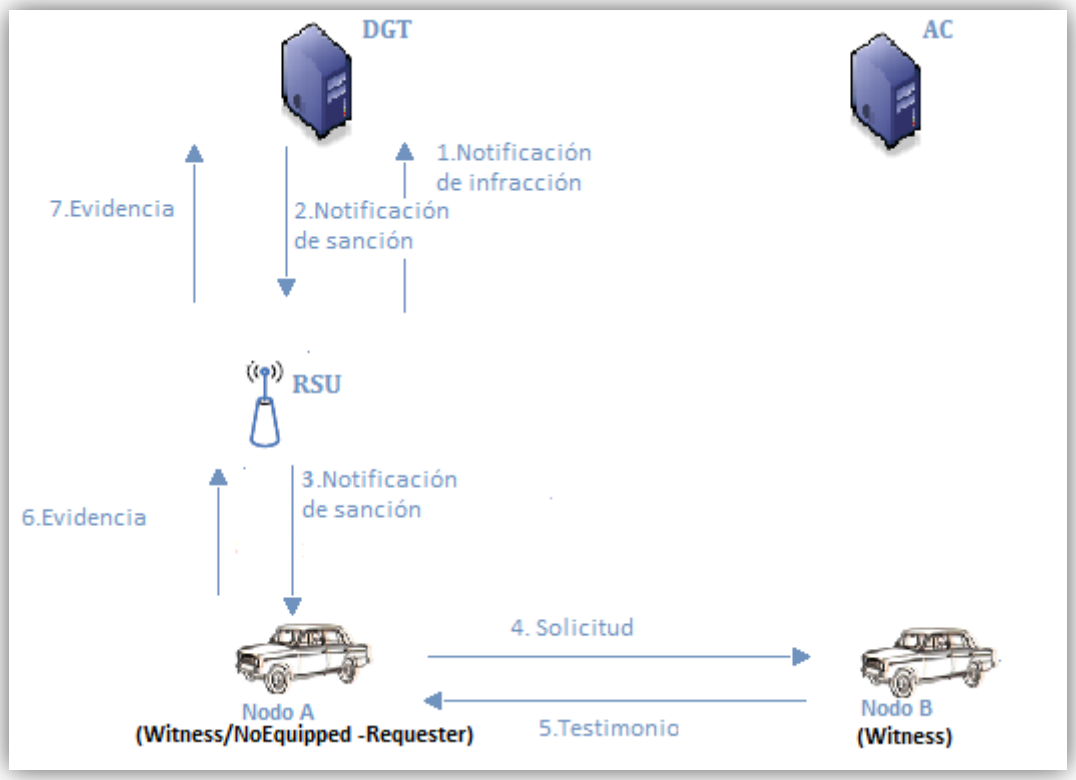


Ilustración 3: descripción básica del protocolo EVIGEN.

La descripción, de acuerdo con los mensajes presentados en la Ilustración 3, es la siguiente (4):

1. La RSU detecta que un vehículo, en adelante nodo A, circula por encima del límite de velocidad establecido, notificándoselo a la DGT.

2. La DGT genera la notificación correspondiente y se la envía a la RSU para que ésta realice el envío al nodo oportuno.

3. La RSU envía la notificación de la sanción al nodo A.

4. El nodo A, al cual ha asumido el rol de Requester, envía una solicitud a un vehículo Witness, en adelante nodo B, para obtener un testimonio y poder revocar la sanción.

5. El nodo B, Witness, responde al nodo A con un testimonio. (En caso de haberse desplazado lo suficientemente lejos como para que el mensaje no pudiese ser recibido por el nodo A, se debería realizar un reenvío a nodos NoEquipped o RSU que se consideren cercanos).

6. El nodo A crea la evidencia asociada al testimonio recibido y la envía a la RSU.

7. La RSU realiza el envío a la DGT.

2.2.3 Escenario alternativo

Conocido el funcionamiento principal del protocolo, se presenta un escenario que permanece al margen del protocolo pero cuyo conocimiento es necesario para la correcta comprensión del desarrollo presentado.

Este escenario se corresponde con lo que sucede si la DGT no recibe la evidencia asociada a una notificación de sanción enviada o si dicha evidencia se considera no fiable, lo cual puede ocurrir por distintos motivos. Por una parte, si alguno de los mensajes intercambiados en el protocolo se pierde (considerar el envío vía *wireless*), la evidencia no podría ser enviada. Otra posible circunstancia es que un vehículo no envíe una evidencia no favorable aunque el protocolo se desarrollase adecuadamente, lo cual se asume. Finalmente, también puede suceder que un vehículo construya una evidencia no fiable, enviándola a la DGT para poder evitar la sanción.

La descripción de este escenario se realiza bajo la suposición de que se produce un fallo en la recepción de uno de los mensajes del protocolo y, por ello, la evidencia no puede ser enviada a la DGT. El desarrollo se muestra posteriormente, en la Ilustración 4.

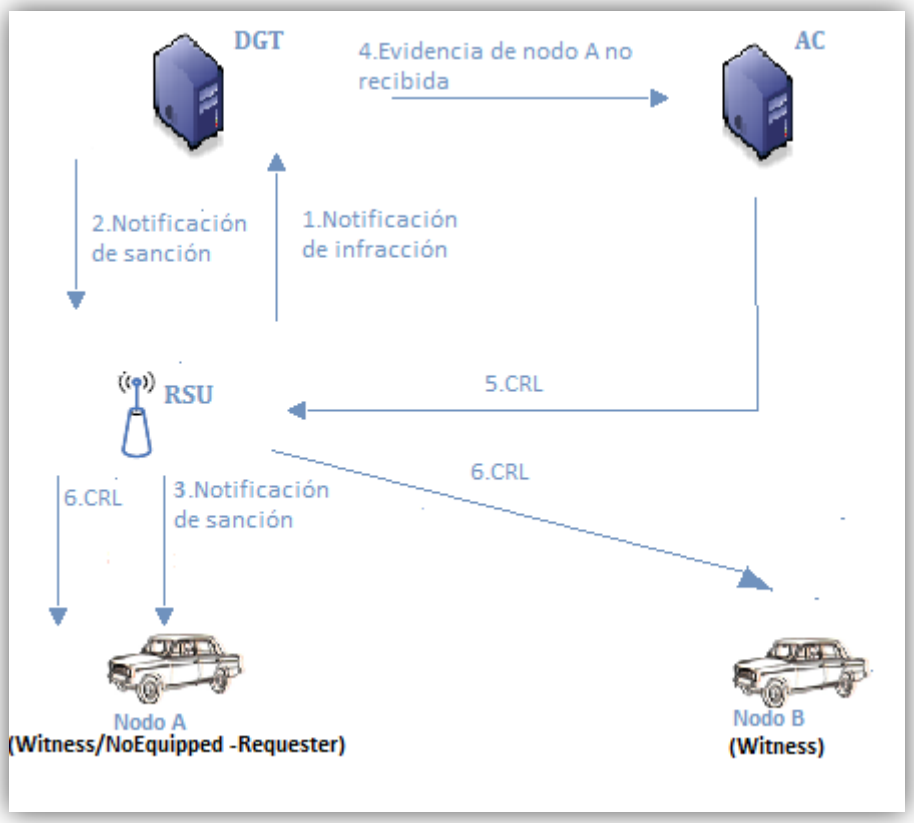


Ilustración 4: escenario alternativo vinculado al desarrollo del protocolo EVIGEN.

La descripción es la siguiente:

1. La RSU detecta que un vehículo, en adelante nodo A, circula por encima del límite de velocidad establecido, notificándoselo a la DGT.

2. La DGT genera la notificación correspondiente y se la envía a la RSU para que ésta realice el envío al nodo oportuno.

3. La RSU envía la notificación de la sanción al nodo A.

4. Pasado un tiempo, la DGT no recibe respuesta y le comunica a la AC la falta de la recepción de la evidencia procedente del nodo A.

5. La AC revoca el certificado del nodo A, introduciéndolo en la CRL y enviándola a la RSU para propagar la modificación realizada.

6. La RSU envía, a todos los nodos que permanezcan en su radio de cercanía, la CRL recibida.

3 Análisis

En esta sección se analiza el problema que se pretende resolver, es decir, la simulación del protocolo EVIGEN mediante NCTUns 5.0. Para ello, se presenta en primer lugar la descripción detallada de los objetivos a alcanzar (sección 3.1). La sección posterior (sección 3.2) aborda el análisis de la situación actual, mostrándose el estudio del simulador NCTUns 5.0 (1). En la tercera sección (sección 3.3) se presenta la arquitectura preliminar establecida. Posteriormente, se establecen las tecnologías impuestas (sección 3.4) para, en la sección siguiente, realizar un estudio tecnológico de todas las tecnologías que pueden ser utilizadas (sección 3.5). Seguidamente, se presenta la arquitectura definitiva de acuerdo a los objetivos del proyecto (sección 3.6). Tras el análisis tecnológico realizado, se especifican las tecnologías escogidas (sección 3.7). Posteriormente, se muestran los casos de uso (sección 3.8), para en la sección siguiente (sección 3.9) establecer los requisitos del software. Por último, se describe el diseño del plan de pruebas para garantizar la satisfacción de los requisitos (sección 3.10).

3.1 Descripción de los objetivos del proyecto

Establecidos de forma global los objetivos del presente proyecto (sección 1.1), en esta sección se profundiza cada uno de ellos.

3.1.1 Descripción de la implementación del protocolo EVIGEN: mensajes intercambiados

Descrito el funcionamiento del protocolo, en este apartado se especifica la parte a realizar en el presente proyecto, la cual se corresponde al desarrollo de las funcionalidades efectuadas por los nodos con rol Requester, Witness y NoEquipped. Además, se presenta el contenido de cada uno de los mensajes intercambiados.

Con el fin de simplificar la presentación de los mensajes se va a utilizar la notación expuesta posteriormente, Tabla 1.

Notación	Significado
$Cert_X$	Certificado del nodo X.
E_{KpubX}	Cifrado con la clave pública del nodo X.
$posX_X$	Posición, en m, en el eje X de la pantalla de simulación en la que se encuentra el nodo X.
$posY_X$	Posición, en m, en el eje Y de la pantalla de simulación en la que se encuentra el nodo X.
S_{KpvX}	Firma con la clave privada del nodo X.
tipoV	Tipo de rol que está desempeñando un nodo en un determinado momento, puede ser W para Witness u O para NoEquipped. El rol de Requester no se contempla puesto que puede ser ejercido por ambos al recibir una notificación de sanción.
v_X	Velocidad, en m/sg, a la que circula el nodo X.
iV_X	Intervalo de velocidad [X1.X2] al que circulaba el nodo X.
v_F	Velocidad, en m/sg, final correspondiente con la intersección de los intervalos recibidos y la media del intervalo resultante.
+	Operador de concatenación.

Tabla 1: notación mensajes protocolo.

Primeramente, indicar que a lo largo de toda una simulación se intercambian mensajes de *beaconing* y, tal y como se indica en la sección 2.2.3, mensajes con la CRL, los cuales no se incluyen como parte del protocolo pero que serán considerados como tal. El contenido de dichos mensajes es el siguiente:

- Mensaje *beaconing*: contiene la información de estado de cada nodo. Posteriormente se presenta el mensaje:

$$Mensaje\_beaconing = id_X + posX_X + posY_X + v_X + tipoV$$

- Mensaje CRL: contiene la lista de nodos cuyo certificado ha sido revocado. En la siguiente línea se presenta el mensaje:

$$Mensaje\_CRL = Cert_{CA} + S_{KpvCA}(listaIdCoches) + listaIdCoches + TTL$$

Por otra parte, para realizar la descripción de los mensajes participantes en el protocolo EVIGEN se asume el siguiente escenario de ejecución, Ilustración 5. Es necesario indicar que con el nombre "ProtocoloEVIGEN EntornoInfraestructura" se referencia la parte desarrollada en el proyecto complementario a este (2), mencionada en la sección 1.1.

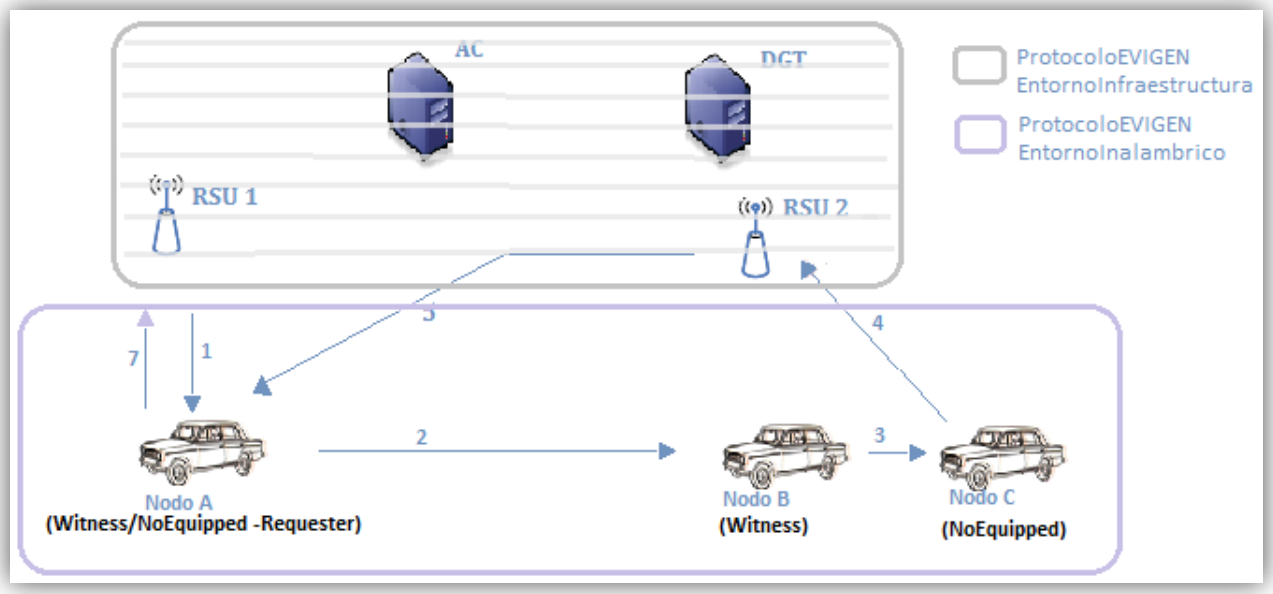


Ilustración 5: ejemplo protocolo EVIGEN.

Observando la Ilustración 5 hay que subrayar que el mensaje 4 se podría propagar tanto a RSU 2 como a RSU 1 pero se presenta un escenario simplificado para que los mensajes intercambiados en la ejecución del protocolo se comprendan con mayor facilidad.

El protocolo EVIGEN se desarrolla, en términos generales, de la siguiente forma (4):

1. Por medio de una RSU, un vehículo (en adelante nodo A), Witness o NoEquipped, recibe una notificación de sanción, convirtiéndose a su vez en Requester y verificando que la sanción recibida es válida. Dicha notificación contiene lo siguiente:

$$Notificaci\acute{o}n =$$
$$Cert_{DGT} + S_{KpvDGT}(id_A + fecha + vMulta + id_{RSU}) + (id_A + fecha + vMulta + id_{RSU})$$

2. El nodo A solicita a otro vehículo (en adelante nodo B), con rol Witness, la creación de un testimonio en el que se indique la velocidad a la que circulaba. Esta solicitud se enviará, por tanto, a todos los nodos situados en una determinada área geográfica cercana al solicitante. La solicitud consta de lo siguiente:

$$Solicitud = Cert_A + S_{KpvA}(id_A + TTL1 + marcaT1) + (id_A + TTL1 + marcaT1)$$

3. El nodo B verifica la solicitud, tras lo cual debería realizar el envío del testimonio al Requester, nodo A. Sin embargo, como circulaban a distintas velocidades, el nodo A no es alcanzable y el testimonio se envía a un vehículo NoEquipped (en adelante nodo C), al cual sí se considera cercano. Tras verifica la corrección de la solicitud, el testimonio a crear consta de lo siguiente:

$$\underline{\underline{ticket}} = E_{KpubCA}(Cert_B + S_{KpvB}(claveAES) + claveAES)$$

$$\underline{\underline{condTesti}} = E_{KAES}(v_A)$$

$$Testimonio = E_{KpubA}(hash(id_B + TTL2 + marcaT2) + hash(\underline{\underline{ticket}} + \underline{\underline{condTesti}}) + \underline{\underline{ticket}} + \underline{\underline{condTesti}} + v_A) + (id_B + TTL2 + marcaT2)$$

4. El nodo C reenvía el testimonio a una RSU, puesto que es la única cercana.

$$Testimonio = E_{KpubA}(hash(id_B + TTL2 + marcaT2) + hash(\underline{\underline{ticket}} + \underline{\underline{condTesti}}) + \underline{\underline{ticket}} + \underline{\underline{condTesti}} + v_A) + (id_C + TTL3 + marcaT3)$$

5. La RSU reenvía el testimonio al Requester que lo solicitó. El testimonio reenviado es el siguiente:

$$Testimonio = E_{KpubA}(hash(id_B + TTL2 + marcaT2) + hash(\underline{\underline{ticket}} + \underline{\underline{condTesti}}) + \underline{\underline{ticket}} + \underline{\underline{condTesti}} + v_A) + (id_{RSU} + TTL4 + marcaT4)$$

6. El nodo A, Requester, crea la evidencia. Una vez verificada la validez del testimonio recibido (o recibidos, puesto que se puede recibir más de uno por cada solicitud) se crea la evidencia, que consta de lo siguiente:

$$\underline{ticketsCondTesti} = \underline{\underline{ticket}}1 + \underline{\underline{condTesti}}1(+\underline{\underline{ticket}}2 + \underline{\underline{condTesti}}2 + ... + \underline{\underline{ticketN}} + \underline{\underline{condTestiN}})$$
$$Evidencia = Cert_A + S_{KpvA}(v_F) + v_F + \underline{ticketsCondTesti}$$

7. El nodo A envía la evidencia a la RSU más cercana para que sea entregada a la DGT.

3.1.2 Descripción del desarrollo de la visualización de los mensajes del protocolo

Una vez descrito el protocolo EVIGEN y con el objetivo de simplificar el análisis de su funcionamiento, se especifican las bases del desarrollo que permita visualizar los mensajes intercambiados en la simulación del mismo. Además, se deben visualizar los datos que, por limitaciones del simulador NCTUns 5.0 (1), no pueden ser mostrados.

Por un lado, se necesita observar los mensajes intercambiados a lo largo de una simulación, detallando los datos contenidos en cada uno de ellos. Teniendo en cuenta el alcance de este proyecto, estos mensajes serán los numerados con 1, 2, 3, 4, 5, 6 y 7 en la explicación del protocolo (sección 3.1.1). Asimismo, teniendo en cuenta la gran cantidad de mensajes que pueden ser intercambiados, se ofrece la posibilidad de limpiar la pantalla en cualquier momento para que el usuario supervisor de la simulación pueda visualizar los mensajes con mayor comodidad.

Por otro lado, asumiendo la existencia de una lista de identificadores de nodo cuyo certificado está revocado (CRL), es adecuado poder comprobar los identificadores incluidos en ella puesto que, de este modo, se puede conocer el motivo por el que unos mensajes son respondidos y otros, por el contrario, son descartados.

Por otra parte, se tiene que mostrar, en cada paso de la simulación, la información de estado correspondiente a cada uno de los vehículos. Se estima conveniente poder conocer el identificador, la posición, la velocidad y el tipo de vehículo/rol, así como la información de estado del vecindario asociado a cada nodo, en cada instante de la simulación.

Finalmente, para poder analizar los mensajes intercambiados y poder verificar que los contenidos son los adecuados, se precisa almacenar los mensajes en un fichero con formato PDF.

3.2 Análisis de la situación actual

Tal y como ya se ha mencionado, este proyecto se ha de integrar con el simulador NCTUns 5.0 (1) y, por ello, es imprescindible el estudio y el análisis del mismo para, posteriormente, poder resolver las necesidades especificadas, a saber, cómo emplearlo para simular el protocolo EVIGEN y cómo mostrar los mensajes intercambiados en dicho protocolo. Hay que subrayar la realización de un análisis interno, quedando fuera del ámbito de estudio la creación de entornos de simulación puesto que para ello se puede hacer uso del manual de usuario del simulador NCTUns 5.0 (5).

Para comenzar, es muy importante tener en cuenta que este simulador se ejecuta sobre un núcleo desarrollado especialmente para su ejecución, de modo que las direcciones de red no se corresponden con las establecidas comúnmente.

A grandes rasgos, esta aplicación está compuesta por cuatro componentes claramente diferenciados cuya relación se muestra en la Ilustración 7. El propósito de cada uno de ellos es el siguiente:

- ***Dispatcher***: su función principal es la asignación de un determinado *Coordinator* a la *GUI* para que realice la simulación del mismo.
- ***GUI***: también denominado cliente, responsable de proporcionar las herramientas necesarias para que un usuario pueda crear y visualizar una simulación.

  En relación con este componente se debe tener en cuenta que, a pesar de que NCTUns 5.0 es un simulador de libre distribución, no se proporciona la implementación de la interfaz gráfica, cuya imagen se muestra en la Ilustración 6.

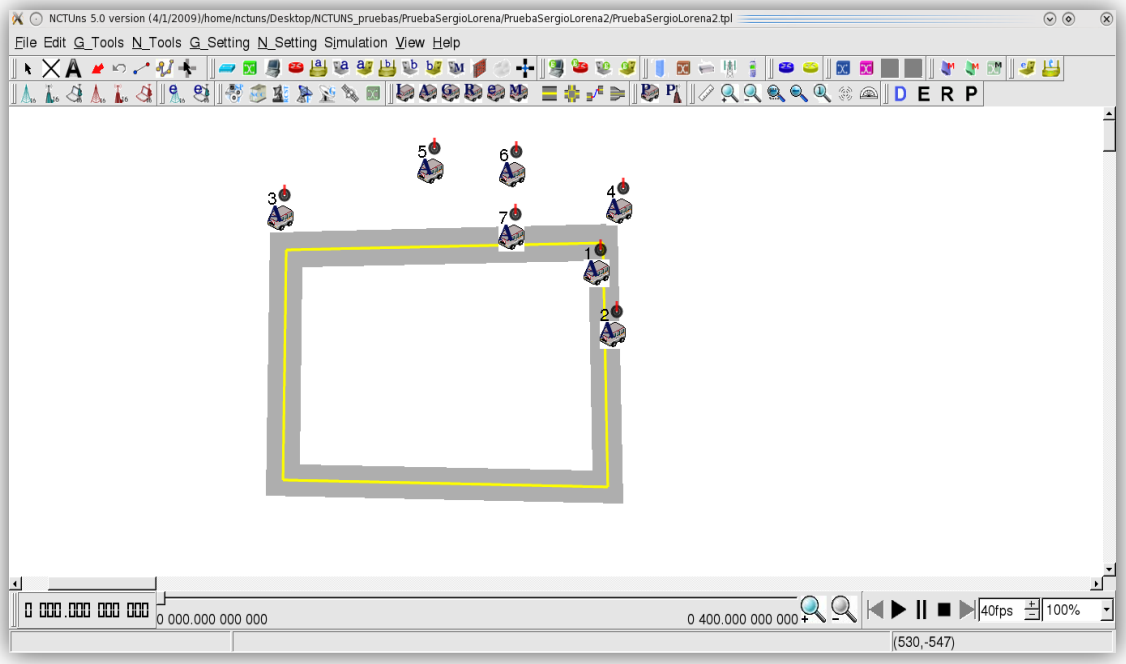


Ilustración 6: GUI simulador NCTUns 5.0.

- ***Coordinator***: encargado de la transmisión de datos desde y hacia la *GUI*, de modo que se observen por pantalla los resultados de una simulación realizada. Además, se relacionará con el *SimulationEngine* para efectuar las operaciones y crear los objetos solicitados por la *GUI*.
- ***SimulationEngine***: responsable de la realización de las operaciones necesarias para efectuar la simulación solicitada.

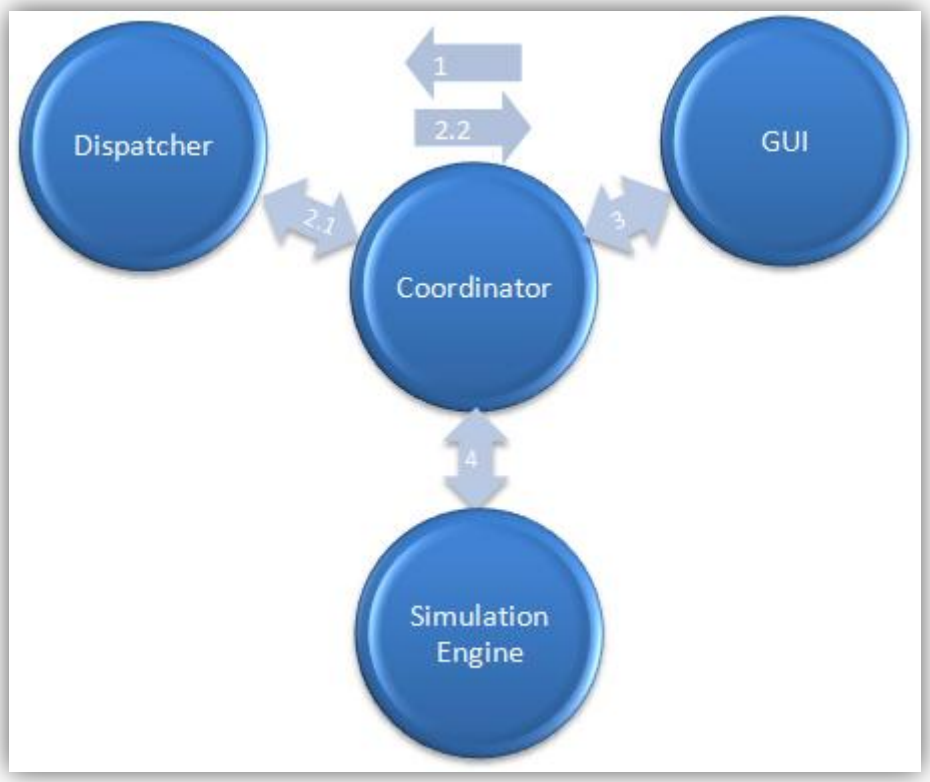


Ilustración 7: estructura NCTUNS 5.0.

De acuerdo con el diagrama presentado, el flujo de ejecución es el siguiente:

1. Desde la *GUI* se indica al *Dispatcher* el comienzo, desconexión o reconexión de una simulación.
2. Operaciones efectuadas por el *Dispatcher*.
   2.1. Se indica a un *Coordinator* que atienda una petición proveniente de la *GUI*, de modo que el *Dispatcher* establezca a dicho *Coordinator* como ocupado.
   2.2. Se indica al cliente el puerto y la dirección del *Coordinator* que va a atender su simulación.
3. Establecimiento de la conexión entre *Coordinator* y la *GUI*, de modo que por esta conexión se realicen peticiones desde la simulación ejecutada en la *GUI* al *Coordinator* junto con los comandos de parada, pausa, continuación y aborto de una ejecución. De igual modo, de *Coordinator* a *GUI* se indicará información relacionada con el progreso de la simulación.

4. Establecimiento de la conexión entre el *Coordinator* y el *SimulationEngine*. Por un lado, el *Coordinator* solicita al *SimulationEngine* los comandos de gestión de la simulación indicados por la *GUI*. Mientras que el *SimulationEngine* proporciona respuesta a las solicitudes requeridas así como realiza las actualizaciones adecuadas a la información que alberga.

Otra de las consideraciones a realizar, es el funcionamiento interno de una simulación cuya ejecución se realiza en el componente *GUI*. En cada una de las simulaciones participan una serie de nodos, vehículos en este caso, a los que se les asocia un determinado programa, el cual proporciona un comportamiento concreto para cada uno de los nodos, como puede ser un vehículo roto o un vehículo que únicamente se desplaza hacia abajo.

Teniendo en cuenta lo comentado, la comunicación realizada entre cada uno de los programas y el componente *SimulationEngine* se basa en el establecimiento de una conexión TCP por la que cada nodo se comunica con este componente para, posteriormente, solicitar datos del entorno y actuar de acuerdo con el funcionamiento establecido. Todo lo mencionado se ejemplifica en la Ilustración 8.

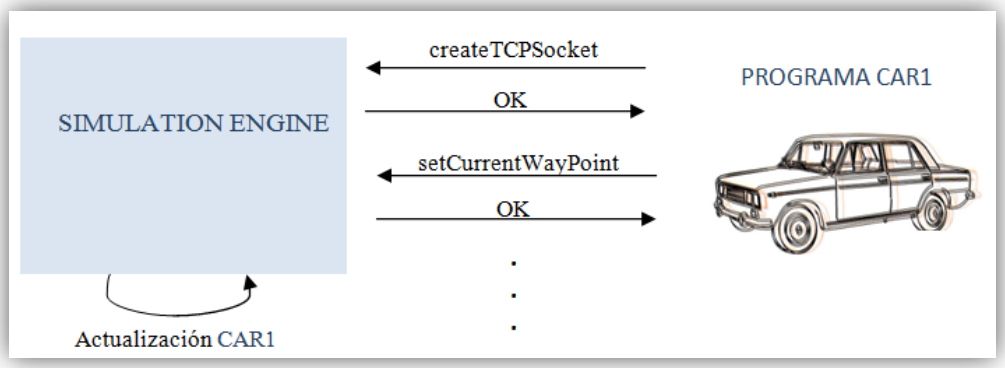


Ilustración 8: funcionamiento NCTUNS 5.0.

Además, para la creación de las comunicaciones y los envíos de información entre nodos, es muy importante considerar, tal y como se mencionó al comienzo de la sección, que las direcciones de red no se corresponden con las comúnmente utilizadas. Por tanto, hay que tener presente que la dirección para el envío masivo de información entre nodos, dirección de *broadcast*, se corresponde con la 1.0.1.255.

Estudiado todo el sistema, es necesario profundizar en el *SimulationEngine* puesto que es el componente en el que, tras este estudio analítico, parece necesaria una mayor intervención para lograr los objetivos de este proyecto. Las siguientes imágenes, Ilustración 9 e Ilustración 10, muestran la estructura interna de este componente.

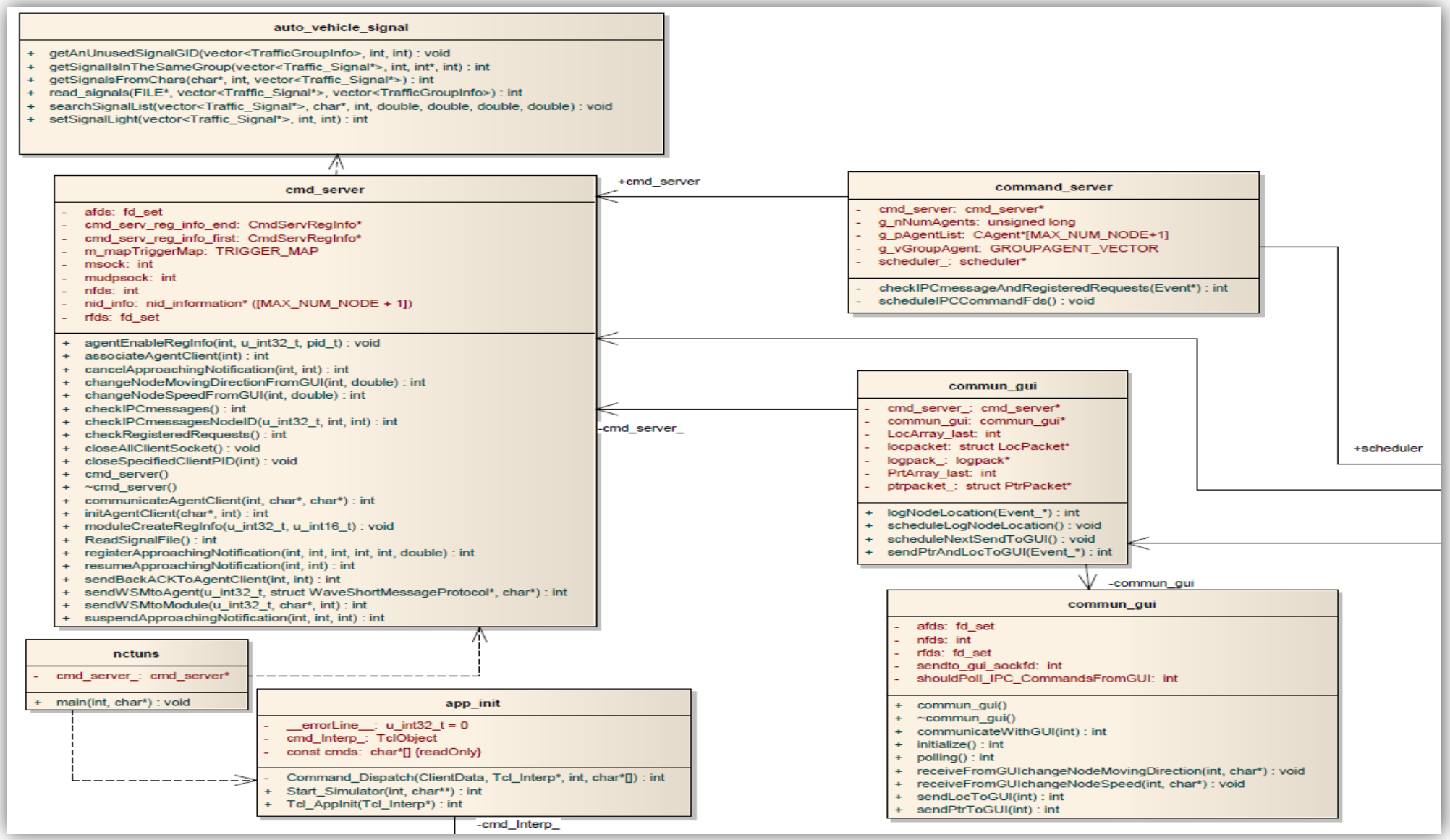


Ilustración 9: diagrama 1 SimulationEngine.

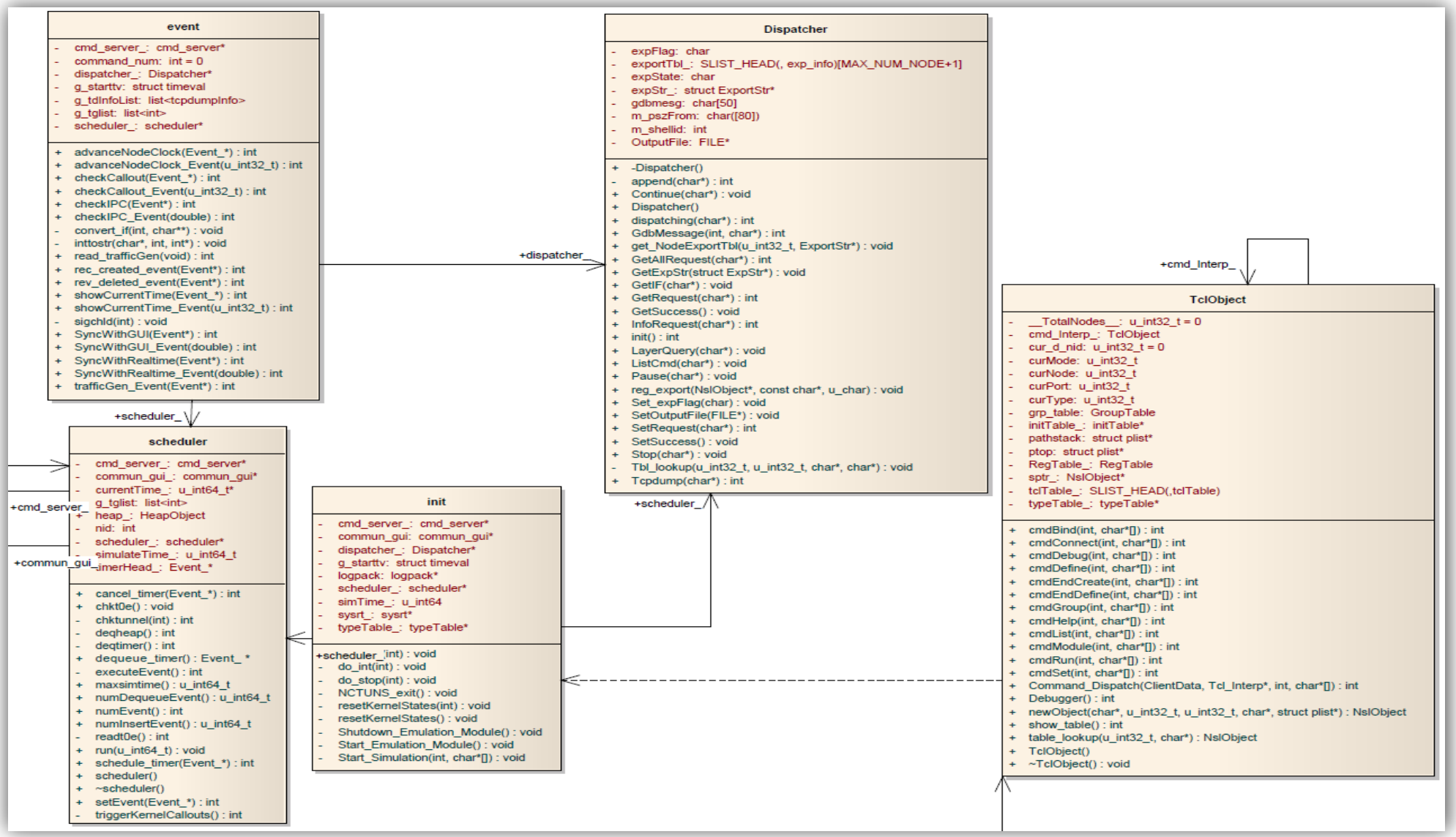


Ilustración 10: diagrama 2 SimulationEngine.

Brevemente, el funcionamiento de este componente es el siguiente:

1. Se registran todos los módulos de los diferentes protocolos que pueden ser utilizados (clase "NCTUns") y se lee el fichero de configuración de la simulación correspondiente (clase "app_init"), en el cual se encuentran campos como la velocidad del nodo o la posición inicial. Dicho fichero es específico para cada simulación y se crea al guardar cada una de las simulaciones, su extensión es *.xtpl*.
2. Se ejecutan una serie de comandos encargados de construir el entorno en el que se va a ejecutar la simulación (clase "tclObject"). Entre estos comandos se puede destacar la construcción de los nodos, como son los vehículos.
3. Se procede a la ejecución de las operaciones que definen el funcionamiento de la simulación.

   Para comenzar, se realiza la asignación de direcciones a los diferentes nodos de la interfaz de simulación y se crea un evento encargado de lanzar un proceso independiente para ejecutar un programa correspondiente por cada uno de dichos nodos. Esto es una de las funcionen principales para la puesta en marcha de la simulación (clase "event").

   Después se crean otros eventos para que fluya la comunicación entre el *Coordinator*, el *Dispatcher* y el *SimulationEngine*.

   En último lugar, se activa el planificador para comenzar la ejecución de la simulación y encargarse de la gestión y ejecución de los eventos (clase "scheduler").

   Todo lo descrito en este punto se desarrolla dentro de la clase "init".

Tras finalizar el estudio, las conclusiones más relevantes que han sido obtenidas son:

- No se proporciona la implementación de la interfaz gráfica del simulador.
- En el fichero de configuración *.xtpl* se dispone de campos pertenecientes a las características de cada nodo, como son la velocidad o la posición inicial.
- Una simulación está compuesta por una serie de nodos, que en este caso coinciden con vehículos, a los que se les asocia un programa correspondiente con un comportamiento a simular. Además, es en la clase "event" en la que se efectúa la lectura de cada uno de los programas.
- El simulador NCTUns 5.0 (1) se ejecuta en un núcleo especialmente desarrollado para su ejecución. Por tanto, la dirección de *broadcast* no se corresponde con 255.255.255.255, sino que se ha realizado una implementación en la que la dirección asociada a este propósito es la 1.0.1.255.

3.3 Arquitectura preliminar

En secciones anteriores se ha presentado el protocolo EVIGEN (sección 3.1.1), el cual constituye la base de este proyecto, y el estudio del simulador NCTUns 5.0 (1) (sección 3.2) con el que se pretende simularlo. Por tanto, en este punto del análisis se puede efectuar una arquitectura preliminar del sistema a desarrollar, lo cual servirá de punto de partida para el posterior diseño.

Teniendo en cuenta que los dos objetivos de este proyecto (sección 1.1) afectan a partes distintas del simulador, en esta sección se presentan ambas cuestiones de manera separada. La sección 3.3.1 aborda el desarrollo de la visualización de los mensajes intercambiados en las simulaciones del protocolo y la sección 3.3.2 presenta la arquitectura preliminar para la creación del protocolo de intercambio de mensajes entre los distintos tipos de nodos. Sin embargo, entre ambas partes hay un flujo de datos, el cual se representa en la Ilustración 11.

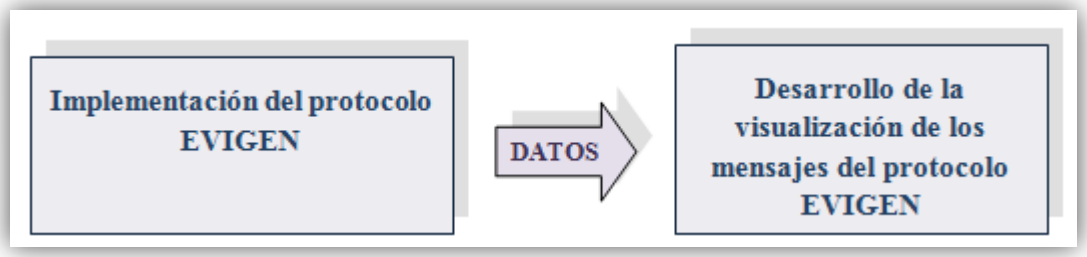


Ilustración 11: definición preliminar del flujo de datos EVIGEN-visualización de mensajes.

3.3.1 Arquitectura preliminar para el desarrollo de la visualización de los mensajes del protocolo

Considerando lo comentado en la sección 3.2, en la que se indicaba que el código de la *GUI* no es proporcionado, hace imposible la modificación o ampliación de la interfaz del simulador NCTUns 5.0 (1). Por tanto, se establece que el desarrollo de la visión de los mensajes intercambiados en el protocolo EVIGEN se va a llevar a cabo mediante la creación de un sistema independiente, denominado, a partir de este momento, sistema de visión de mensajes de la simulación.

La Ilustración 12 muestra la arquitectura preliminar establecida.

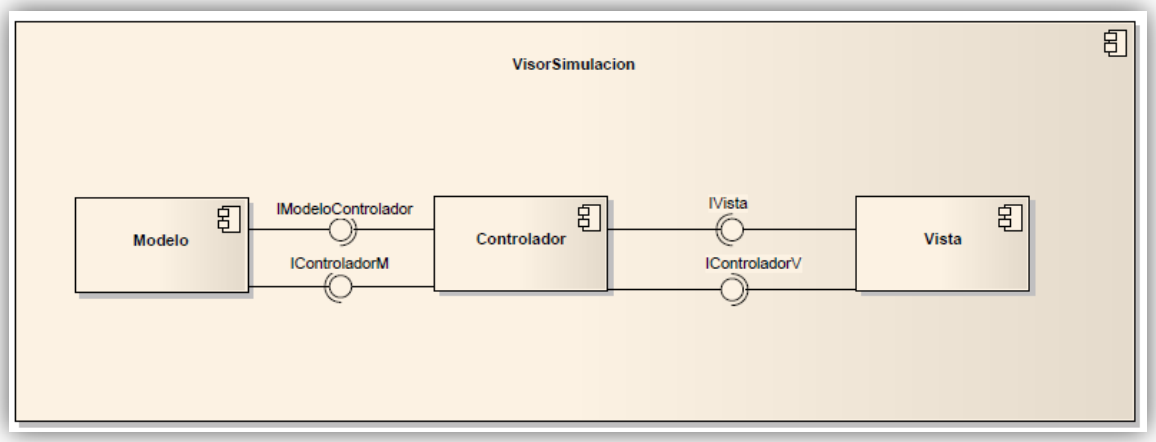


Ilustración 12: arquitectura preliminar del sistema de visión de mensajes de la simulación.

Se aprecia la existencia de tres componentes que se corresponden con el uso del patrón modelo-vista-controlador (MVC) (6) puesto que es un patrón que permite la independencia de componentes y se adecúa en gran medida a las tres partes en las que se ha escogido dividir el sistema, presentación de la información, control de la información y almacén de la información.

Por un lado, el *Modelo* incluye la recepción de información proveniente del simulador y referente a los mensajes intercambiados, los nodos que intervienen en la simulación, los nodos con certificado revocado y la información de estado de cada uno de ellos (velocidad, localización, identificador y tipo de vehículo/rol). Además, será el encargado de proporcionar al *Controlador* toda la información que se le solicite.

Por otra parte, el *Controlador* será el componente encargado del control de información y, por ello, en él se establecerán condiciones que garanticen que todo lo que se solicite desde la *Vista* no contenga errores o, de lo contrario, notificar el error para que se actúe en consecuencia. Asimismo, solicitará respuesta al *Modelo* para las peticiones realizadas por la *Vista*.

Finalmente, la *Vista* será la encargada de presentar al usuario la información referente a los mensajes intercambiados, a los nodos con certificado revocado y al estado de cada uno de los nodos partícipes en la simulación.

3.3.2 Arquitectura preliminar de la implementación del protocolo EVIGEN

Teniendo en cuenta las características del protocolo (sección 3.1.1) y la estructura fundamental del simulador (sección 3.2), en esta sección se describen los aspectos fundamentales de la adaptación necesaria para la integración del protocolo EVIGEN en dicho simulador y siendo denominada, a partir de este momento, extensión de NCTUns para la simulación del protocolo EVIGEN.

En este diagrama, Ilustración 13, se presenta la parte correspondiente con la implementación del protocolo.

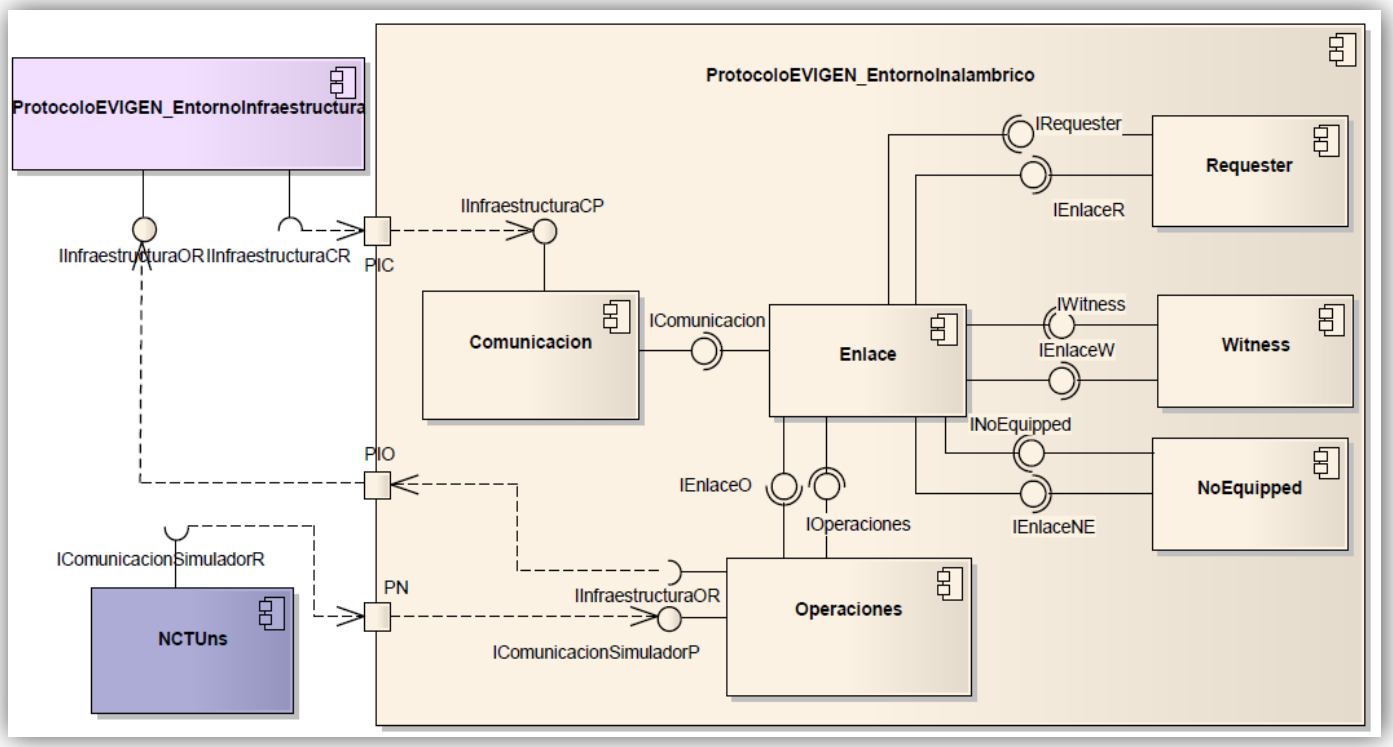


Ilustración 13: arquitectura preliminar de la extensión de NCTUns para la simulación del protocolo EVIGEN.

Se pueden apreciar tres componentes, *NCTUns*, correspondiente con los componentes del simulador presentados en la sección 3.2, *ProtocoloEVIGEN_EntornoInfraestructura*, relativo a la implementación del proyecto complementario (2) y *ProtocoloEVIGEN_EntornoInalambrico*, donde se lleva a cabo el desarrollo del protocolo EVIGEN vinculado a este proyecto.

Primeramente, en la parte derecha de *ProtocoloEVIGEN_EntornoInalambrico* se presentan los componentes correspondiente con los roles existentes en el protocolo (componentes *Requester*, *Witness*, *NoEquipped*) en donde se realizarán las acciones propias de cada uno de ellos, mientras que las operaciones comunes a ambos se incluirán en el componente *Operaciones*. Además, se dispone del componente enlazador, *Enlace*, para realizar el acceso a las operaciones comunes como pueden ser inserciones en listas o comprobaciones de datos, y, también, para acceder a las operaciones de envío y recepción de mensajes, realizadas en el componente *Comunicacion*, parte de las cuales corresponderán con la entrega de mensajes al sistema visor de mensajes de la simulación.

Por otra parte, dado que hay que efectuar la integración con el simulador NCTUns 5.0 (1) se hace uso del componente *NCTUns*, señalado en azul y mencionado en los objetivos con el nombre "Desarrollo NCTUns 5.0" (sección 1.1). Por tanto, a este componente se solicitan datos necesarios para la realización de determinadas operaciones como es la creación de la movilidad de los vehículos.

Por último, se aprecia *ProtocoloEVIGEN_EntornoInfraestructura*, representándose la interacción entre el desarrollo del protocolo realizado en este proyecto y el desarrollo del protocolo efectuado en el complementario (2). De este modo, se ofrecen datos para facilitar el cumplimiento de los requisitos establecidos en dicho proyecto y se obtienen datos para poder representar correctamente todos los mensajes intercambiados y satisfacer los objetivos establecidos en el presente proyecto.

3.4 Análisis de las tecnologías impuestas

Dado que los artefactos software que se desarrollen en este proyecto, como respuesta informática a las necesidades planteadas, deben relacionarse con un producto software preexistente (en concreto, el simulador NCTUns en su versión 5.0), es necesario analizar los condicionantes tecnológicos que vienen impuestos por esta circunstancia.

Es importante tener en cuenta que uno de los requisitos para el funcionamiento del simulador NCTUns 5.0 (1) es el de ejecutarse en el sistema operativo Fedora (7) y preferiblemente Fedora10 para evitar problemas de incompatibilidades. Además, el simulador está desarrollado en el lenguaje de programación C++ y la conexión entre los nodos, correspondientes con los vehículos, se realiza mediante el uso de sockets UDP no bloqueantes.

3.5 Estudio tecnológico

En esta sección se presentan distintas tecnologías necesarias para conseguir los objetivos establecidos. La exposición se divide en el estudio de las tecnologías aplicables al sistema de visión de mensajes de la simulación y las aplicables al desarrollo de la extensión de NCTUns para la simulación del protocolo EVIGEN.

El contenido de este estudio queda resumido en forma tabular en la Tabla 2, presentada posteriormente.

Sistema de visión de mensajes de la simulación	
Necesidad tecnológica	**Alternativas Posibles**
Lenguajes de programación aplicables al sistema de visión de mensajes de la simulación.	Java
	C#
	C++
	HTML
Librerías para la creación de la interfaz gráfica del sistema de visión de mensajes de la simulación.	Java Swing (8)
	Windows Forms (9)
	GTK+ (10)
Tecnologías aplicables para la creación de ficheros PDF.	iText (11)
	iTextSharp (12)
Extensión de NCTUns para la simulación del protocolo EVIGEN	
Necesidad tecnológica	**Alternativas Posibles**
Tecnologías aplicables para sistemas distribuidos.	Java RMI
	AXIS (13)
	CORBA (14)
	Sockets
Tecnologías aplicables a la ejecución en paralelo, comunicación y sincronización.	Tuberías
	Paso de mensajes
	Ficheros
	Variables de memoria compartida
	Mutex
	Variables condicionales
Tecnologías para el almacenamiento de los elementos configurables.	Ficheros
	Base de datos

Tabla 2: entorno tecnológico estudiado.

Previamente al comienzo del análisis, es conveniente indicar que asumiendo la existencia de herramientas y tecnología de libre distribución, lo suficientemente potentes como para realizar un buen desarrollo, así como la necesidad de no incrementar el presupuesto establecido en el proyecto, se va a descartar el uso de herramientas y tecnologías comerciales.

3.5.1 Tecnologías aplicables al sistema de visión de mensajes de la simulación

En esta sección se mostrarán lenguajes en los que es posible realizar la implementación del sistema de visión de mensajes de la simulación, las posibles librerías para el desarrollo de la interfaz gráfica del sistema y las librerías viables para la creación de ficheros PDF.

3.5.1.1 Lenguajes de programación aplicables al sistema de visión de mensajes de la simulación

Para comenzar, los lenguajes más adecuados escogidos para la realización de este sistema son C++. C# o Java, así como HTML para la creación de páginas web que puedan ser incluidas como soporte. Todos ellos son estudiados en los siguientes párrafos.

Por un lado se encuentra el lenguaje C++ en el que se puede programar por medio de aplicaciones como Microsoft Visual Studio (15), que proporcionan buenas herramientas para crear interfaces gráficas. Por otro lado, hay que considerar que ha de ser compilado en el sistema en el que se vaya a ejecutar y, de este modo, maximizar las prestaciones ofrecidas por cada plataforma (Windows, Mac, Solaris, etc.).

Otra opción viable es la utilización de C#. La sintaxis es similar a C/C++ y está orientado a objetos. Además, es un lenguaje similar a Java, desarrollado para la plataforma .NET de Microsoft, que proporciona ciertos mecanismos para facilitar la programación funcional.

El lenguaje Java también es otra alternativa para la creación de interfaces gráficas. Una de las principales ventajas es la cantidad de documentación que puede ser encontrada para el desarrollo de cualquier aplicación, no sólo es más sencillo encontrar manuales sino que se dispone de un completo *Javadoc* que proporciona ayuda en el momento deseado. No obstante, es un paradigma orientado a objetos que no puede ser combinado con otros como el de la programación procedural. Finalmente indicar que por ser compilado para una JVM es independiente de la plataforma, pudiendo ser ejecutado en muchos entornos aunque sin obtener el máximo rendimiento.

Otros de los lenguajes, utilizados como base para la creación de páginas web, es HTML, el cual puede ser integrado en múltiples sistemas para presentar cierta información que requiera ser actualizada o como soporte o medio de contacto para los usuarios de la aplicación con los desarrolladores del sistema.

Existen otros muchos lenguajes de programación pero quedan fuera del ámbito de este proyecto ya que, de acuerdo con el propósito a conseguir, no son lenguajes que se consideren tan adecuados como los citados.

3.5.1.2 Librerías para la creación de la interfaz gráfica del sistema de visión de mensajes de la simulación

Por otra parte, en cuanto a la librería que puede ser escogida para la creación de la interfaz gráfica, las opciones más adecuadas son Java Swing (8),Windows Forms (9) o GTK+ (10), analizadas posteriormente.

En lo referente a Java Swing (8) hay que indicar que es una biblioteca, independiente de la plataforma, que se utiliza para la creación de interfaces gráficas en lenguaje Java. Asimismo, es de uso libre y posee documentación de ayuda y soporte, pero es gracias a la ayuda de editores como el proporcionado por NetBeans (16) lo que simplifica su utilización.

En cuanto a GTK+ (10), es una librería multiplataforma viable para Windows, Linux y Mac, que permite programar en lenguajes de programación como Java, C++, C, PHP, etc. Además, es de libre distribución y, de igual modo que Java Swing (8), se puede hacer uso de editores como NetBeans (16) para facilitar la implementación.

En lo referente a Windows Forms (9), señalar que es una aplicación para crear aplicaciones gráficas por medio de la plataforma .NET. Del mismo modo que Java Swing (8), dispone de manuales e información de soporte, así como de editores para simplificar el desarrollo de las interfaces, en este caso un ejemplo sería Microsoft Visual Studio (15).

3.5.1.3 Tecnologías aplicables para la creación de ficheros PDF

Teniendo en cuenta la necesidad de crear ficheros PDF en los que se presentan los mensajes intercambiados en una simulación, se han de analizar las librerías existentes para la realización de este objetivo. Las posibilidades ofrecidas, consideradas más adecuadas, son principalmente dos, la librería itext (11) o bien la librería itextSharp (12).

Las funcionalidades incluidas en ambas librerías son prácticamente idénticas, ambas son sencillas y ofrecen un gran número de ejemplos y manuales para poder aprender con facilidad la sintaxis y las funciones a utilizar.

Sin embargo, la diferencia entre ambas es que itext (11) puede ser utilizada siempre que el lenguaje en el que se incluya sea Java, mientras que itextSharp (12) debe ser utilizada si el lenguaje en el que se hace uso de ella es C#.

3.5.2 Tecnologías aplicables de la extensión de NCTUns para la simulación del protocolo EVIGEN

Por otro lado, para el desarrollo del protocolo EVIGEN se estudian las tecnologías que permitan el intercambio de datos entre dos programas, independientes, en ejecución. Además, en el protocolo se debe considerar la existencia de distintos procesos ejecutándose simultáneamente por lo que es necesario analizar las tecnologías que permitan la sincronización y comunicación de los mismos. Asimismo, el protocolo requiere elementos configurables y, por este motivo, es imprescindible escoger un modo de almacenamiento.

3.5.2.1 Tecnologías aplicables para sistemas distribuidos

Tal y como se expuso en la sección 3.2, dado que no se dispone del código de la interfaz de NCTUns 5.0, el visor de mensajes debe desarrollarse de forma independiente al simulador. Por este motivo, es imprescindible analizar las tecnologías existentes para la creación de aplicaciones distribuidas.

En este caso se han de realizar intercambios de mensajes desde el protocolo desarrollado en el simulador NCTUns 5.0 (1) al sistema de visión de mensajes la simulación. Finalmente, teniendo en cuenta lo mencionado y sin haber establecido los lenguajes a utilizar, se analizarán las formas más comunes de intercambio de mensajes en aplicaciones distribuidas: Java RMI, AXIS (13), CORBA (14) y sockets.

Para comenzar, Java RMI proporciona un modo sencillo de invocación de métodos remotos exclusivamente para ser utilizado en Java.

Por un lado, AXIS (13) es una implementación de SOAP, el cual es un protocolo basado en XML para realizar la comunicación y el intercambio de información por medio de HTTP, soportado tanto para Java como para C++. Además, unas de sus fortalezas son la interoperatibilidad y la extensibilidad, debido a la utilización de estándares y protocolos abiertos. No obstante, una de sus debilidades es el rendimiento proporcionado ya que los paquetes se han de encapsular en SOAP y, posteriormente, ser enviados vía HTTP.

Por otra parte, CORBA (14) es un modo de comunicación de aplicaciones desarrolladas en distintos lenguajes de programación, lo cual es una de sus ventajas. Además, es un modo de programación en el que únicamente se realizan llamadas teniendo en cuenta el nombre de los servicios sin necesidad de establecer puertos, de hecho, ofrece un conjunto de servicios de alto nivel. No obstante, no es uno de los modos más sencillos de programar y, por ello, requiere el manejo del lenguaje IDL y el correcto conocimiento del estándar para que la programación sea fluida. Asimismo, es un mecanismo que proporciona un rendimiento relativamente lento, en comparación con otros como los sockets, debido a la sobrecarga producida por las cabeceras.

Finalmente, los sockets son otro de los modos de programación distribuida, rápidos y sencillos pero que, por el contrario, requieren programar a bajo nivel. Por tanto, es

necesario analizar los tipos de sockets que podrían ser utilizados teniendo en cuenta el tipo de transporte. Por un lado, se puede utilizar el protocolo TCP de modo que las comunicaciones sean seguras y se garantice que los datos llegan al extremo en el que se solicitan. Por otra parte, la utilización de sockets con UDP implicaría considerar que los mensajes pudiesen no ser recibidos en el extremo deseado. Además, es conveniente indicar que los sockets también pueden ser bloqueantes o no bloqueantes, en el primer caso se espera hasta que llega información por el socket y en el segundo, únicamente se comprueba si un dato ha sido o no recibido.

3.5.2.2 Tecnologías aplicables a la ejecución en paralelo

Otra de las consideraciones a realizar es que en este sistema se van a ejecutar, simultáneamente, tantos nodos, internamente denominados procesos, como se deseen. Esto conduce a la necesidad de paralelizar las tareas realizadas por cada proceso, ya que de lo contrario el protocolo sería muy lento, creándose múltiples hilos por proceso para poder ejecutar dichas tareas. Por ello, hay que analizar los lenguajes en los que puede ser permisible crearlos y los modos de sincronización y comunicación existentes.

Recordando que el simulador está implementado en el lenguaje C++, se establece que el lenguaje C sea el utilizado para la creación de los hilos. Sin embargo, dicha elección requiere la programación a bajo nivel, apareciendo limitaciones como la implementación de la encapsulación o el polimorfismo mediante variables estáticas o estructuras. No obstante, son precisamente tales limitaciones, así como otras, las que provocan que pueda ser un lenguaje utilizado para maximizar la eficiencia de los sistemas.

Por otro lado, para la comunicación y sincronización de procesos se encuentran las tuberías y el paso de mensajes. En cuanto a las tuberías, señalar que se realiza la comunicación de modo que el flujo de datos sea unidireccional. Y, en cuanto a paso de mensajes, es otro mecanismo en el que se necesita realizar, explícitamente, las operaciones de recepción y envío, y en el que los procesos pueden estar en la misma o en distinta máquina.

En lo referente a mecanismos que únicamente garantizan la comunicación, se encuentran los ficheros y las variables de memoria compartida. Los ficheros, por su parte, son un modo sencillo de comunicación en el que es muy importante considerar las escrituras, en el caso de ser concurrentes. Por otro lado, las variables de memoria compartida son un recurso cómodo para realizar la comunicación y compartición de recursos entre procesos que se ejecutan en una misma máquina, sin embargo, al igual que ocurría en el caso de los ficheros es conveniente establecer mecanismos de sincronización adecuados.

En lo relacionado con los mecanismos únicamente de sincronización se encuentran los semáforos, los *mutex* y las variables condicionales. Por una parte, los semáforos son generalmente utilizados para sistemas con memoria compartida y a cada semáforo se le

asocia una cola de procesos. Por otro lado, los *mutex* y las variables condicionales son mecanismos sencillos para la sincronización de procesos ligeros cuyas funciones son únicamente bloquear y desbloquear la zona en la que se establezcan.

3.5.2.3 Tecnologías para el almacenamiento de los elementos configurables

De igual modo que sucede en otros sistemas, se han de almacenar elementos para posteriormente ser cargados en el sistema. Con el fin de conseguir este propósito se puede escoger, principalmente, entre realizar el almacenamiento en un fichero o en una base de datos.

Por un lado, la elección de un fichero como método de almacenamiento proporciona grandes ventajas como son la facilidad de manejo de las operaciones de ficheros y la posibilidad de elección del modo de inserción y obtención de los elementos puesto que se pueden almacenar de forma secuencial, con saltos de línea, etc. Además, no es necesaria la instalación de ningún sistema ni el aprendizaje de ningún tipo de sintaxis. Sin embargo, hay que tener presente que sólo es adecuado si la cantidad de información a manipular no es muy elevada y si el control de concurrencia en el acceso se puede realizar de manera sencilla.

Por otro lado, la elección de una base de datos facilita la gestión de los datos puesto que se proporcionan funciones para el acceso directo y la manipulación de los mismos. Asimismo, se pueden establecer distintas políticas de control de concurrencia, dependiendo del uso requerido. No obstante, en contraste con al almacenamiento mediante ficheros, es indispensable conocer el lenguaje utilizado en la base de datos escogida.

3.6 Arquitectura definitiva de alto nivel

Estudiadas las tecnologías y teniendo una visión de los objetivos a realizar, en esta sección se presenta, en profundidad, la arquitectura sobre la que se va a diseñar el sistema. Además, al igual que en ocasiones anteriores, esta sección se divide en la arquitectura definitiva del sistema de visión de mensajes de la simulación y la arquitectura definitiva de la extensión de NCTUns para la simulación del protocolo EVIGEN.

Asimismo, tal y como se indicó en la arquitectura preliminar, y recordando el flujo de datos existente entre el simulador NCTUns y el sistema de visión de mensajes de la simulación, se muestra la Ilustración 14, equivalente a la presentada en la sección 3.3.

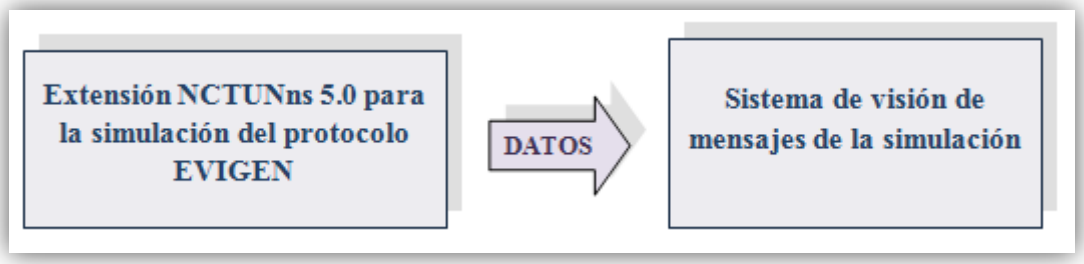


Ilustración 14: definición definitiva del flujo de datos EVIGEN-visualización de mensajes.

3.6.1 Arquitectura definitiva sistema de visión de mensajes de la simulación

Para la realización de este componente, correspondiente con el sistema visor de la simulación, se va a hacer uso del patrón MVC (6), tal y como se comentó en la arquitectura preliminar y como se ilustra en la siguiente imagen, Ilustración 15.

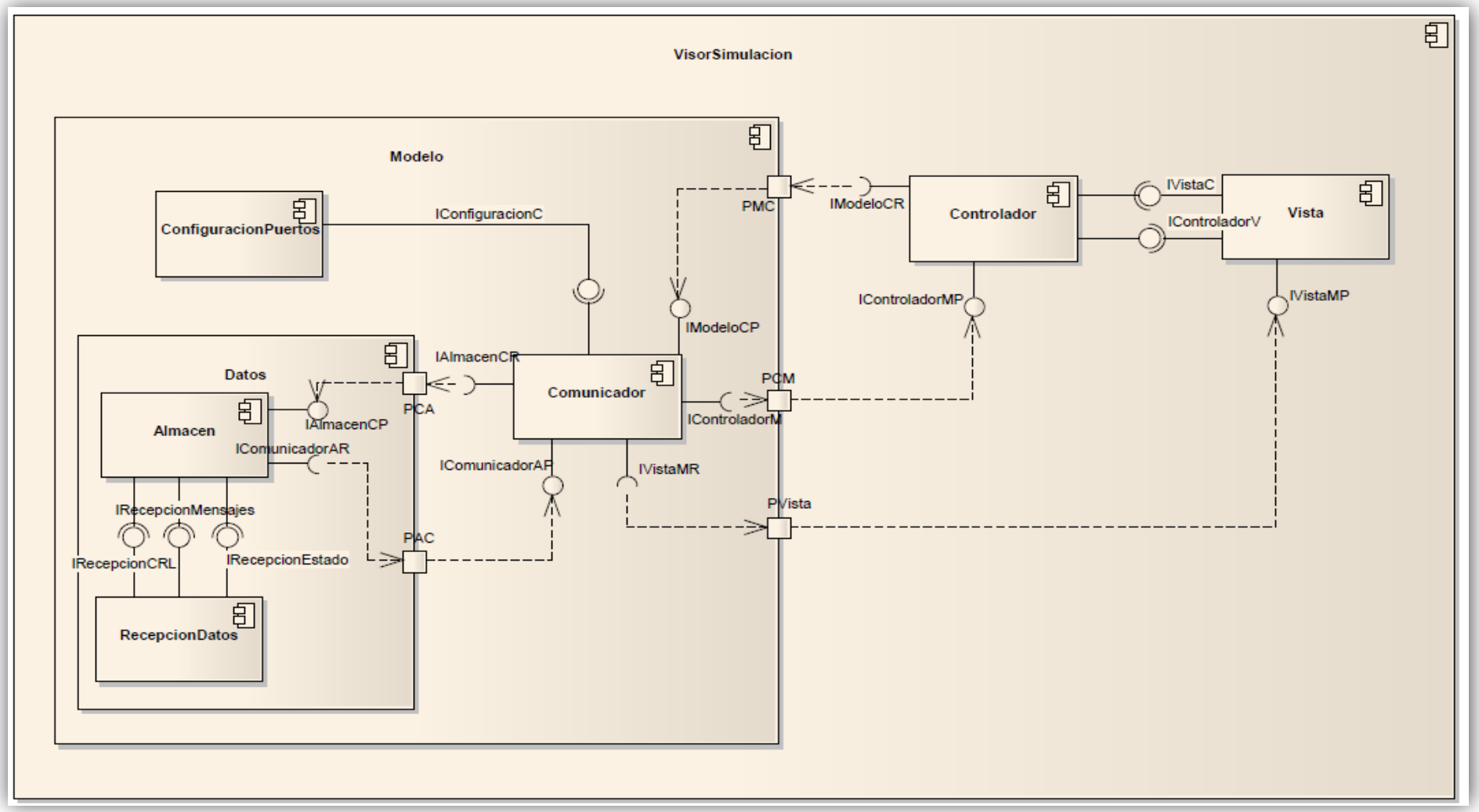


Ilustración 15: arquitectura definitiva sistema de visión de mensajes de la simulación.

Tras realizar el análisis preliminar hay que subrayar la modificación realizada en el comportamiento del componente *Modelo*, así como en la descomposición realizada en el mismo.

Por un lado, el cambio de funcionamiento se corresponde con el envío de determinada información recibida desde el simulador NCTUns 5.0 (1), a la Vista, de modo que ésta última se encargue de mostrarla por pantalla.

Además, se puede apreciar la aparición de tres nuevos componentes. Por un lado, el componente *ConfiguracionPuertos* se centra en la actualización y almacén de los puertos establecidos en la interfaz para la comunicación con el simulador NCTUns 5.0 (1).

Por otro lado, en el componente *Datos* se centran dos grandes funciones asociadas a los tres tipos de datos que pueden ser recibidos, la lista de identificadores de nodos con certificado revocado, los mensajes intercambiados en la simulación y la información de estado enviada en los mensajes de *beaconing* junto con la lista de nodos participantes en cada simulación. Una de las grandes funciones se desarrolla en el componente *RecepcionDatos*, cuyo objetivo es el de realizar la recepción de los datos procedentes del simulador NCTUns 5.0 (1), los cuales se corresponden con los tres tipos de datos mencionados. La otra función es abordada por el componente *Almacen*, donde se produce el almacenamiento de los datos mencionados, así como la preparación de los mismos para su posterior presentación. Por todo lo comentado, se puede apreciar la existencia de un patrón *Observer* (17) que tras la recepción de un dato lo notifica para su posterior visualización.

Por último, el componente *Comunicador* actúa como un patrón *Facade* (17) para proporcionar una interfaz común, tanto para el acceso a los elementos configurables, los puertos, como para el acceso a la información recibida del simulador NCTUns 5.0 (1).

En cuanto a la funcionalidad a desarrollar por el resto de componentes, *Vista* y *Controlador*, hay que señalar que no se ha modificado con respecto a la descrita en la arquitectura preliminar (sección 3.3.1).

3.6.2 Arquitectura definitiva de la extensión de NCTUns para la simulación del protocolo EVIGEN

En el diagrama correspondiente con la Ilustración 16 se presenta la parte del sistema en el que se desarrolla el protocolo EVIGEN.

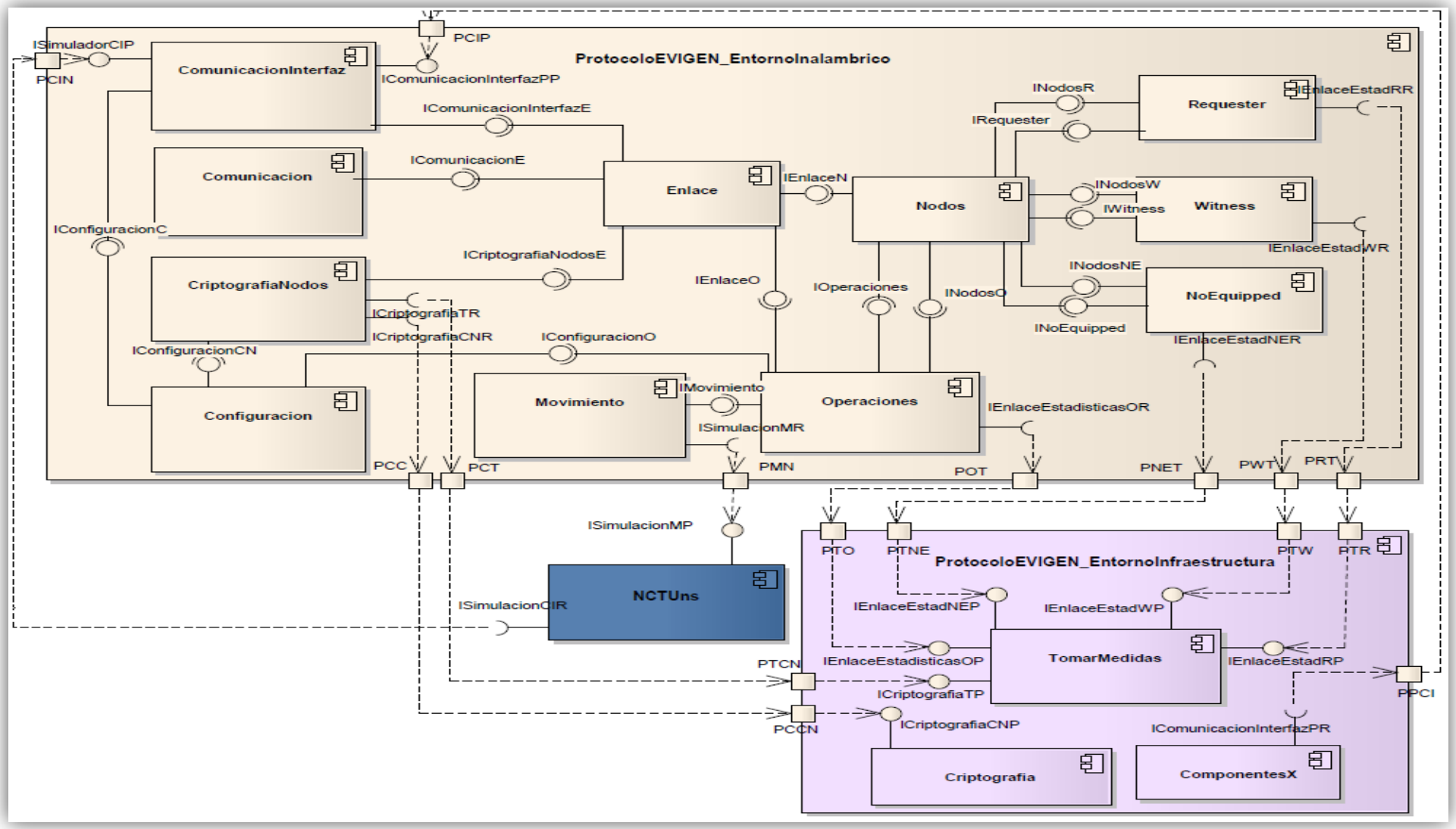


Ilustración 16: arquitectura definitiva de la extensión de NCTUns para la simulación del protocolo EVIGEN.

Tal y como se puede apreciar, se requieren tres componentes para el desarrollo del protocolo. Por un lado se sitúa *ProtocoloEVIGEN_EntornoInalambrico*, el cual es el asociado al desarrollo de este protocolo. Por otro lado, *Criptografia*, *TomarMedidas* y *ComponenteX* son los componentes vinculados al proyecto complementario a éste (2). Por último, *NCTUns* se corresponde con los componentes del simulador NCTUns 5.0 (1) presentados en la sección 3.2.

Primeramente se describen los componentes asociados al desarrollo de *ProtocoloEVIGEN_EntornoInalambrico*.

En la parte superior derecha se aprecian los componentes correspondientes con los distintos roles partícipes en el protocolo, *Requester*, *Witness* y *NoEquipped*. En dichos componentes se realizarán las operaciones propias de cada rol. Además, se puede observar el componente *Nodos*, utilizado como enlazador de todos los roles, de modo que se establezca una interfaz común para el acceso a múltiples operaciones. (Patrón *Facade* (17))

En la zona inferior se observan dos componentes. Por un lado, *Operaciones* es el lugar en el que se realizan las acciones comunes a todos los roles y, por otro lado, *Movimiento*, donde se incluye la acción que proporciona movilidad a los vehículos, la cual se crea por medio de operaciones proporcionadas por el componente *NCTUns*.

En la parte central de diagrama se aprecia el componente *Enlace*, utilizado como enlazador entre los componentes *Configuracion*, *CriptografiaNodos* y *ComunicacionInterfaz*, y, de este modo, poder presentar una interfaz común tanto para el componente *Nodos* como para el componente *Operaciones*.

En la parte inferior izquierda se presenta el componente *Configuracion*, donde se realiza la lectura de los distintos puertos con los que establecer la comunicación, las rutas donde se almacenan los certificados y las claves privadas y las variables que pueden ser modificadas en distintas simulaciones.

Otro de los componentes es el de *Comunicacion*, donde se incluyen las operaciones relacionadas con la creación de sockets, la creación de hilos asociados a las recepciones de datos y los envíos y las recepciones de mensajes.

En la zona central izquierda se localiza el componente *CriptografiaNodos* responsable de la realización de las operaciones criptográficas para lograr que los mensajes intercambiados no puedan ser alterados o examinados.

En la esquina superior izquierda se observa el componente *ComunicacionInterfaz*, encargado de la preparación de datos y del envío de los mismos al sistema de visión de mensajes de la simulación. Además, es necesaria la creación de una interfaz con la que otros componentes externos puedan enviar datos sin necesidad de ningún tratamiento previo.

Por otra parte, en la zona inferior de la ilustración se encuentra el componente *NCTUns*, sombreado de color azul, el cual es el responsable de la vinculación del protocolo EVIGEN con el simulador NCTUns 5.0 (1). En este componente se proporcionan operaciones necesarias para el funcionamiento de los nodos, correspondiendo con la obtención de las posiciones de cada uno de ellos y la creación del movimiento. Además, habiendo estudiado (sección 3.2) que es en este componente donde se crean los nodos participantes en la simulación, se deberán desarrollar las operaciones necesarias para proporcionar la identidad de cada uno de ellos al sistema de visión de mensajes de la simulación y, posteriormente, representar los mensajes asociados a dichos nodos.

En último lugar, tal y como se mencionó al comienzo del apartado, los componentes sombreados de color morado se corresponden con los desarrollados en el proyecto complementario a éste (2). Por un lado, el componente *ProcoloEVIGEN_EntornoInfraestructura* refleja la integración con *ProtocoloEVIGEN_EntornoInlambrico*, mostrándose así el desarrollo completo del protocolo. Concretamente, con el nombre de *TomarMedidas* se alude al componente al que se envían los datos pertinentes para la correcta creación de las estadísticas deseadas. Por otro lado, con el nombre de *Criptografía* se referencia al lugar donde se efectúa la implementación de operaciones criptográficas y que es utilizado para la creación y verificación de los mensajes del protocolo. Finalmente, con el nombre *ComponenteX* se señalan el o los componentes que harán uso de *ComunicacionInterfaz* para poder mostrar, también, mensajes intercambiados en la parte del protocolo desarrollada en el proyecto complementario (2).

3.7 Selección de las tecnologías no impuestas

Una vez presentada la arquitectura preliminar (sección 3.3), es preciso finalizar con la selección de aquellas tecnologías que no quedaban impuestas por el contexto técnico preexistente. Así, en esta sección se efectúa la selección definitiva de las tecnologías, teniendo en cuenta las alternativas tecnológicas presentadas en la sección 3.5.

Tal y como se ha realizado en secciones anteriores, se divide en las elecciones realizadas para el sistema de visión de mensajes de la simulación y las realizadas para la extensión de NCTUns para la simulación del protocolo EVIGEN.

En la Tabla 3 posterior, se presentan, en verde, las elecciones escogidas en relación con las posibilidades especificadas.

Sistema de visión de mensajes de la simulación	
Necesidad tecnológica	**Alternativas Posibles**
Lenguajes de programación aplicables al sistema de visión de mensajes de la simulación.	Java
	C#
	C++
	HTML
Librerías para la creación de la interfaz gráfica del sistema de visión de mensajes de la simulación.	Java Swing (8)
	Windows Forms (9)
	GTK+ (10)
Tecnologías aplicables para la creación de ficheros PDF.	iText (11)
	iTextSharp (12)
Extensión de NCTUns para la simulación del protocolo EVIGEN	
Necesidad tecnológica	**Alternativas Posibles**
Tecnologías aplicables para sistemas distribuidos.	Java RMI
	AXIS (13)
	CORBA (14)
	Sockets
Tecnologías aplicables a la ejecución en paralelo, comunicación y sincronización.	Tuberías
	Paso de mensajes
	Ficheros
	Variables de memoria compartida
	Mutex
	Variables condicionales
Tecnologías para el almacenamiento de los elementos configurables.	Ficheros
	Base de datos

Tabla 3: tecnologías escogidas.

3.7.1 Selección realizada para desarrollar el sistema de visión de mensajes de la simulación

Para comenzar, considerando la facilidad y la flexibilidad que proporciona Java así como la gran extensión de este lenguaje, aún más para el desarrollo de interfaces gráficas, se ha decidido que es la mejor opción para llevar a cabo este propósito.

Además, hay que indicar que dentro de este sistema se hará uso del lenguaje HTML para la creación del manual de usuario, de modo que ante cualquier problema o duda, se pueda consultar en internet o incluso enviar un correo al fabricante en el que se indique alguna debilidad del sistema, y poder mejorarla en futuras versiones.

Asimismo, hay que indicar que para la implementación de la interfaz gráfica se escoge la librería Java Swing (8) por disponer de un mayor conocimiento de la misma.

En último lugar, para la creación de ficheros PDF, asumiendo la utilización del lenguaje Java, se hará uso de la librería itext (11).

3.7.2 Selección realizada para la extensión de NCTUns para la simulación del protocolo EVIGEN

Por otra parte, para la comunicación entre el sistema de visión de mensajes de la simulación y el protocolo EVIGEN se ha descartado el uso de Java RMI debido a la incompatibilidad de los lenguajes, y se han escogido los sockets. Esta decisión se basa en la simplicidad de la implementación, puesto que no hay necesidad de crear interfaces o utilizar modos más complejos de programación, como ocurriría en el caso de hacer uso de CORBA (14).

En lo referente a los sockets hay que indicar que se seleccionan los bloqueantes, de forma que no haya que realizar operaciones de lectura cada cierto tiempo y comprobar si ha llegado un dato, asumiéndose un buffer de recepción ilimitado para evitar problemas de descarte de paquetes. Asimismo, es indispensable considerar el tipo de protocolo que puede ser utilizado, TCP, por su parte, es la mejor opción ya que siempre habría garantías de que los datos llegasen en la secuencia establecida y sin errores. Sin embargo, la rapidez es menor que en los sockets UDP debido a todos los controles que han de realizarse, pudiéndose producir una sobrecarga en el sistema. Por tanto, teniendo en cuenta que la aplicación necesita varios procesos funcionando simultáneamente (tantos como nodos), se ha decidido escoger los sockets UDP y prevenir, de este modo, un decremento en la velocidad y en el rendimiento de la máquina.

En lo referente a la creación de procesos ligeros, señalar que se va a hacer uso del lenguaje C y, para la comunicación de los mismos, se ha escogido el mecanismo de paso de mensajes puesto que es el que más se adapta a la comunicación de redes vehiculares, las cuales realizan intercambios de mensajes.

Además, indicar que el intercambio de mensajes entre los nodos participantes en la simulación también se va a implementar utilizado sockets. Sin embargo, la base de esta decisión es distinta con respecto a la indicada para la elección del mecanismo de comunicación entre protocolo y el sistema de visión de los mensajes de la simulación, en este caso hay que considerar la necesidad de la aplicación de funciones criptográficas en los mensajes intercambiados. Por tanto, si el mecanismo de comunicación es de un nivel de abstracción superior, la aplicación de criptografía no tendría sentido puesto que se utilizarían mecanismos que encapsularían los mensajes e implícitamente se proporcionarían funciones de distintos tipos como es la de la seguridad, un ejemplo de ello es lo que ocurre en AXIS (13), por viajar los mensaje por medio de HTTP. Asociado a los sockets está la elección del transporte, el cual también coincide con el indicado anteriormente, UDP, pero esta vez sin tener en cuenta el aumento o decremento de rendimiento, sino considerando que en un entorno real los mensajes entre vehículos se intercambian utilizando como medio el aire y, de este modo, se simula, en la medida de lo posible, la realidad.

Por otro lado, es importante indicar el uso de *mutex* para que los datos compartidos entre los distintos procesos ligeros sean correctos y no se produzcan modificaciones indebidas.

Finalmente, con el fin de realizar el almacenamiento y gestión de los elementos configurables y considerando que la cantidad no va a ser elevada, así como indicándose la simplicidad de uso, se decide que la utilización de ficheros es el modo más apropiado para dicho almacenamiento.

3.8 Diagramas de casos de uso

Tras estudiar las tecnologías y la arquitectura a realizar, en esta sección se presenta la forma y el desarrollo de los casos de uso del sistema, los cuales facilitan la obtención de parte de los requisitos necesarios para el cumplimiento de los objetivos del presente proyecto.

3.8.1 Formato para la especificación de los casos de uso

La tabla de especificación de casos de uso es la Tabla 4.

CASO DE USO		
Título:		
Identificador:	**Versión:**	**Fuente:**
Actores:		
Objetivo:		
Precondiciones:		
Postcondiciones:		
ESCENARIO BÁSICO		
1.- Acción 2.- Acción … N.- Acción		
ESCENARIO ALTERNATIVOS		
Escenario 1		
X.- Condición X+1. Acción alternativa … X+s. Acción alternativa		
Escenario 2		
(…)		
Escenario N		
(…)		

Tabla 4: formato casos de uso.

El propósito de cada uno de los campos es:

- Título: frase en la que se resume el caso de uso.
- Identificador: identificador del caso de uso. El formato será CU-XX, donde XX se corresponden con dígitos comenzando por el 01.
- Versión: versión actual del caso de uso. El formato es X.Y, donde X se corresponde con la versión e Y se corresponde con la revisión.
- Fuente: origen del que se ha obtenido el caso de uso.
- Actores: entidades que interactúan con el sistema.
- Objetivo: breve descripción del propósito a conseguir.
- Precondiciones: estado del sistema necesario para la ejecución del caso de uso.

- Postcondiciones: estado del sistema tras la ejecución del caso de uso.
- Escenario básico: situación en la que el caso de uso se desarrolla siguiendo el flujo de ejecución normal.
- Escenarios alternativos: flujos de ejecución distinto al principal que se llevan a cabo si se cumple una determinada condición en un determinado punto del flujo de ejecución básico.

3.8.2 Actores del sistema

Dentro de este proyecto los casos de uso se centran en la interacción que se produce entre el usuario del sistema de visión de mensajes de la simulación y el usuario que ejecuta las simulaciones en el simulador NCTUns 5.0 (1). Por tanto, los actores son los siguientes:

- Usuario sistema: persona que hace uso de la interfaz gráfica de visualización de los mensajes intercambiados durante la simulación.
- Usuario NCTUns: persona que hace uso del simulador NCTUns 5.0 (1) para visualizar la simulación del protocolo EVIGEN.

3.8.2.1 Casos de uso del actor Usuario sistema

Los casos de uso del sistema se presentan en las tablas posteriores.

CASO DE USO		
Título: escoger nodo.		
Identificador: CU-01	**Versión:** 1.0	**Fuente:** cliente
Actores: usuario sistema.		
Objetivo: escoger un nodo del que observar el intercambio de mensajes.		
Precondiciones: • Sistema visor de mensajes de la simulación arrancado. • Simulación ejecutada en NCTUns arrancada.		
Postcondiciones: • Visualización del nodo escogido.		
ESCENARIO BÁSICO		
1. Seleccionar “Elección de nodo”. 2. Seleccionar un nodo. 3. Terminar.		
ESCENARIO ALTERNATIVOS		
Escenario 1		
1-3a No existe información. 1. Volver a paso 1.		

Tabla 5: CU-01 escoger nodo.

CASO DE USO		
Título: visualizar mensajes.		
Identificador: CU-02	**Versión:** 1.0	**Fuente:** cliente
Actores: usuario sistema.		
Objetivo: observar los mensajes intercambiados entre un nodo y el resto de nodos participantes en la simulación.		
Precondiciones: • Sistema visor de mensajes de la simulación arrancado. • Simulación ejecutada en NCTUns arrancada.		
Postcondiciones: • Visualización de mensajes.		
ESCENARIO BÁSICO		
1. Seleccionar "Elección de nodo". 2. Seleccionar un nodo. 3. Terminar.		

Tabla 6: CU-02 visualizar mensajes.

CASO DE USO		
Título: visualizar estado.		
Identificador: CU-03	**Versión:** 1.0	**Fuente:** cliente
Actores: usuario sistema.		
Objetivo: observar la localización, la velocidad, el tipo, el identificador y el vecindario de un nodo entre los participantes en la simulación.		
Precondiciones: • Sistema visor de mensajes de la simulación arrancado. • Simulación ejecutada en NCTUns arrancada.		
Postcondiciones: • Visualización de la información de estado de un nodo.		
ESCENARIO BÁSICO		
1. Seleccionar "Elección de nodo". 2. Seleccionar un nodo. 3. Pulsar el "icono del nodo" 4. Terminar.		

Tabla 7: CU-03 visualizar estado.

CASO DE USO		
Título: visualizar CRL.		
Identificador: CU-04	**Versión:** 1.0	**Fuente:** cliente
Actores: usuario sistema.		
Objetivo: observar la lista de nodos, participantes en la simulación, cuyos certificados están revocados.		
Precondiciones: • Sistema visor de mensajes de la simulación arrancado. • Simulación ejecutada en NCTUns arrancada.		
Postcondiciones: • Visualización de la CRL.		
ESCENARIO BÁSICO		
1. Seleccionar "Visualización de CRL". 2. Terminar.		

Tabla 8: CU-04 visualizar CRL.

CASO DE USO		
Título: obtener fichero PDF.		
Identificador: CU-05	**Versión:** 1.0	**Fuente:** cliente
Actores: usuario sistema.		
Objetivo: obtener un fichero con extensión PDF en el que se presenten los mensajes intercambiados entre los nodos partícipes en la simulación.		
Precondiciones: • Sistema visor de mensajes de la simulación arrancado.		
ESCENARIO BÁSICO		
1. Seleccionar "Guardar". 2. Seleccionar el identificador del nodo deseado o "Todos", en caso de desear almacenar la información de todos los nodos. 3. Seleccionar una ruta en donde almacenarlo. 4. Escribir un nombre para el fichero. 5. Terminar.		
ESCENARIO ALTERNATIVOS		
Escenario 1		
1-6a No existe información. 1. Volver a paso 1.		
Escenario 2		
5-6b Nombre del fichero incorrecto. 1. Volver a paso 4.		

Tabla 9: CU-05 obtener fichero PDF.

CASO DE USO		
Título: limpiar pantalla.		
Identificador: CU-06	**Versión:** 1.0	**Fuente:** cliente
Actores: usuario sistema.		
Objetivo: limpiar la pantalla y dejarla vacía para poder seguir mostrando más mensajes.		
Precondiciones: • Sistema visor de mensajes de la simulación arrancado. • Simulación ejecutada en NCTUns arrancada.		
Postcondiciones: • Pantalla limpia de mensajes.		
ESCENARIO BÁSICO		
1. Seleccionar "Limpiar". 2. Terminar.		

Tabla 10: CU-06 limpiar pantalla.

CASO DE USO		
Título: acceder a manual.		
Identificador: CU-07	**Versión:** 1.0	**Fuente:** cliente
Actores: usuario sistema.		
Objetivo: acceder al manual de usuario.		
ESCENARIO BÁSICO		
1. Seleccionar "Manual". 2. Terminar.		

Tabla 11: CU-07 acceder a manual.

CASO DE USO		
Título: visualizar mensajes de simulación previa.		
Identificador: CU-08	**Versión:** 1.0	**Fuente:** cliente
Actores: usuario sistema.		
Objetivo: visualización de los mensajes previamente almacenados y, tras reanudar la ejecución del sistema, los mensajes visualizados aumentan.		
Precondiciones: • Sistema visor de mensajes de la simulación pausado. • Simulación ejecutada en NCTUns arrancada.		
Postcondiciones: • Visualización de mensajes, sistema visor de mensajes de la simulación continúa su ejecución.		
ESCENARIO BÁSICO		
1. Seleccionar "Continuar". 2. Terminar.		

Tabla 12: CU-08 visualizar mensajes de simulación previa reanudar sistema.

En último lugar, el diagrama correspondiente a la representación de los casos de uso es mostrado en la siguiente imagen, Ilustración 17:

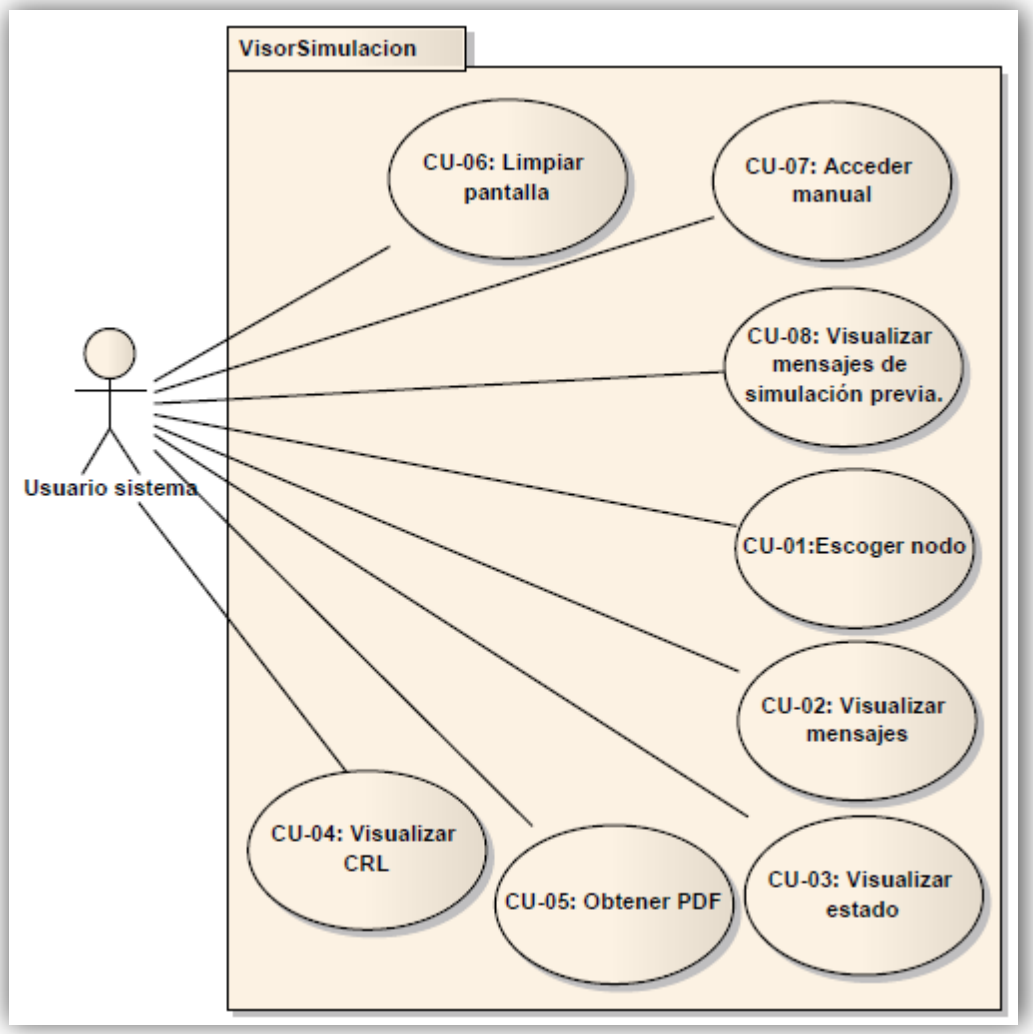


Ilustración 17: diagrama casos de uso, usuario sistema.

3.8.2.2 Casos de uso del actor Usuario NCTUns

El caso de uso asociado al protocolo EVIGEN se presenta posteriormente, Tabla 13.

CASO DE USO		
Título: simular.		
Identificador: CU-09	**Versión:** 1.0	**Fuente:** cliente
Actores: usuario NCTUns.		
Objetivo: visualizar la ejecución del protocolo EVIGEN en la interfaz gráfica proporcionada por el simulador NCTUns 5.0 (1).		
Precondiciones: • Simulación creada en el simulador NCTUns.		
Postcondiciones: • Simulación en ejecución en el simulador NCTUns.		
ESCENARIO BÁSICO		
1. Seleccionar “Simulation Run”. 2. Terminar.		

Tabla 13: CU-09 simular.

El diagrama vinculado al caso de uso presentado es el siguiente, Ilustración 18.

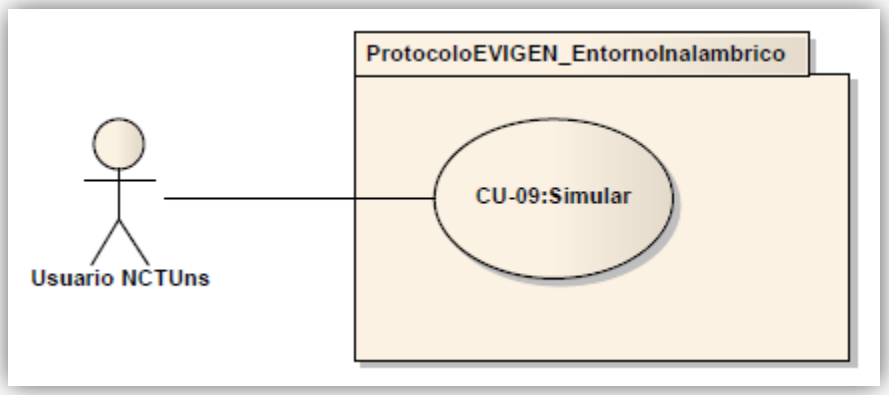


Ilustración 18: diagrama de casos de uso, usuario NCTUns.

3.9 Catálogo de requisitos software

En esta sección se exponen los requisitos de software necesarios para el cumplimiento y desarrollo de los objetivos especificados en este proyecto. El desarrollo de esta sección se realiza siguiendo la metodología ESA (18) y estableciendo que aquellas categorías de requisitos que no se presenten se consideran no aplicables.

3.9.1 Formato para la especificación de los requisitos

La tabla de especificación de requisitos es la siguiente:

REQUISITOS DE SOFTWARE					
Tipo:					
Id	**Ver.**	**Título**	**Descripción**	**Estabilidad**	**Prioridad**
Tipo:					
Id	**Ver.**	**Título**	**Descripción**	**Estabilidad**	**Prioridad**

Tabla 14: formato requisitos de software.

El propósito de cada uno de los campos es:

- Tipo: valor dependiente del identificador, a continuación se exponen los tipos posibles. Los valores que puede tomar son “Funcional”, “Rendimiento”, “Interfaz”, “Manejo de errores”, “Seguridad” o “Mantenibilidad”.
- Id: identificador del requisito. Los requisitos pueden ser funcionales, con identificador RF-XX, no funcionales, con identificador RNF-XX, e inversos con identificador RI-XX. Además, dentro de los no funcionales pueden ser de rendimiento, interfaz, manejo de errores, seguridad y mantenibilidad. Los XX se corresponden con dígitos, comenzando por el 01.
- Título: frase en la que se resume el requisito.
- Descripción: descripción del requisito.
- Ver.: versión actual del requisito. El formato es X.Y, donde X se corresponde con la versión e Y se corresponde con la revisión.
- Estabilidad: probabilidad de que un requisito cambie durante la realización del proyecto. Los valores que pueden ser utilizados son “Alta”, “Media” y “Baja”.
- Prioridad: grado de preferencia en el que un requisito ha de ser desarrollado, desde el punto de vista del cliente o del programador, dependiendo de la fuente del mismo. Los valores que pueden ser utilizados son “Alta”, “Media” y “Baja”.

Además, para cada tipo de requisitos se indicarán las siguientes propiedades:

- Necesidad: grado en el que un requisito se hace necesario en el desarrollo del proyecto. Los valores que pueden ser utilizados son "Imprescindible" y "Opcional".

- Fuente: origen del que se ha obtenido el requisito.

3.9.2 Requisitos funcionales

En esta sección se presentan los requisitos funcionales del sistema a desarrollar. Además, se tiene en cuenta la división existente entre la parte correspondiente con el sistema de visión de mensajes de la simulación y el desarrollo del protocolo EVIGEN.

3.9.2.1 Requisitos funcionales de la extensión de NCTUns para la simulación del protocolo EVIGEN

Los requisitos correspondientes con esta sección del proyecto se presentan en la Tabla 15.

Es conveniente indicar que, para todos ellos, la fuente es el cliente y que la necesidad es imprescindible.

REQUISITOS DE SOFTWARE					
Tipo: funcional					
Id	**Ver.**	**Título**	**Descripción**	**Estabilidad**	**Prioridad**
RF-01	1.0	Recepción de notificación.	Recibir una notificación desde de una RSU.	Alta	Alta
RF-02	1.0	Comprobación de la notificación.	Verificar que el mensaje de notificación recibido es correcto y que el tipo de notificación es el adecuado.	Alta	Media
RF-03	1.0	Recepción de CRL.	Recibir la CRL desde de una RSU, para descartar las solicitudes realizadas por nodos con certificado revocado.	Alta	Alta
RF-04	1.0	Comprobación de la CRL.	Comprobar que la CRL recibida es correcta.	Alta	Media
RF-05	1.0	Recepción de información estado RSU, mensaje de *beaconing*.	Recibir la posición (en metros, m), e identificador en el que se encuentra una determinada RSU.	Alta	Alta
RF-06	1.0	Envío de información de estado, mensaje de *beaconing*.	Enviar la información correspondiente con la posición (en metros, m), la velocidad (en metros/segundo, m/sg), el identificador (dígito mayor que 0) y el rol que desempeña en cada momento un determinado nodo.	Alta	Alta
RF-07	1.0	Envío solicitud de Requester a Witness.	Enviar una solicitud a un nodo o nodos con rol Witness para que con la evidencia recibida, la sanción pueda ser revocada.	Alta	Alta
RF-08	1.0	Comprobación solicitud.	Verificar que la solicitud recibida en un nodo con rol Witness es correcta y proveniente de un nodo adecuado.	Alta	Media
RF-09	1.0	Comprobación testimonio.	Comprobar que el testimonio recibido en un nodo con rol Requester es correcto.	Alta	Media
RF-10	1.0	Envío de testimonio desde Witness a NoEquipped/RSU.	Enviar los testimonios desde un nodo que desempeña el rol de Witness a otro u otros con rol NoEquipped considerados cercano o, de no ser posible, a nodos con rol RSU.	Alta	Alta
RF-11	1.0	Envío de testimonio desde NoEquipped a NoEquipped/RSU.	Enviar testimonio desde nodo que desempeña el rol de NoEquipped a otro u otros con rol NoEquipped considerados cercano o, de no ser posible, a nodos con rol RSU.	Alta	Alta
RF-12	1.0	Envío de testimonio a Requester.	Enviar desde un nodo que desempeñe el rol de Witness, NoEquipped o RSU el testimonio solicitado por un nodo con rol Requester.	Alta	Alta

REQUISITOS DE SOFTWARE					
Tipo: funcional					
Id	**Ver.**	**Título**	**Descripción**	**Estabilidad**	**Prioridad**
RF-13	1.0	Creación de *beaconing*.	Implementar *beaconing*, de modo que sólo los coches situados en un determinado rango puedan recibir ciertos mensajes, lo cual se corresponde con el establecimiento del vecindario de un nodo dado. El rango de cercanía será indicado en el fichero de configuración de variables (RF-20).	Alta	Alta
RF-14	1.0	Creación de evidencia.	Filtrar las velocidades recibidas realizando la intersección de todas ellas y la posterior media del intervalo resultante. Tras esta operación se obtiene el dato con el que se creará la evidencia.	Media	Media
RF-15	1.0	Envío de evidencia.	Envió, a la RSU más cercana, de la evidencia creada por un nodo con rol Requester tras recibir los testimonios creados por distintos Witness.	Alta	Media
RF-16	1.0	Establecimiento de movilidad en vehículos.	Establecer el movimiento en cada uno de los nodos con roles Requester, Witness y NoEquipped.	Alta	Alta
RF-17	1.0	Reenvío de solicitud	Reenviar una solicitud, desde un nodo con rol Requester, si en un cierto tiempo (en milisegundos, ms), indicado en RNF-54, no se ha recibido alguna respuesta satisfactoria, es decir, cuya evidencia contenga una velocidad inferior a la sancionada.	Alta	Media
RF-18	1.0	Configuración rutas de certificados y claves privadas.	Configurar la ruta en la que se sitúan: ▪ Ruta del certificado de un nodo con rol Requester, Witness o NoEquipped, denominad. ▪ Ruta de la clave privada de un nodo con rol Requester, Witness o NoEquipped. ▪ Ruta del certificado de la AC. ▪ Ruta del certificado de la DGT.	Alta	Baja

REQUISITOS DE SOFTWARE					
Tipo: funcional					
Id	**Ver.**	**Título**	**Descripción**	**Estabilidad**	**Prioridad**
RF-19	1.0	Creación de los intervalos de velocidad.	Establecer la velocidad a la que circulaba el nodo Requester que envió la solicitud. Esto se realiza en nodos que desempeñan el rol de Witness, mediante la creación de un intervalo. Dicho intervalo debe establecerse según la ecuación $(velocidadEmisor \mp (velocidadEmidor \times numIntervalo))$, siendo numIntervalo un número comprendido en el intervalo [0,1; 0,4].	Alta	Media
RF-20	1.0	Configuración de variables	Las variables configurables son[1]: ▪ Para la implementación del intercambio de mensajes se establecen tres variables asociadas a las distancias máximas a la que un nodo con rol Requester, Witness o NoEquipped acepta/rechaza la recepción o el envío de mensajes. Se denominan, *DISTANCIA_ACEPTADA_REQUESTER*, *DISTANCIA_ACEPTADA_WITNESS* y *DISTANCIA_ACEPTADA_NOEQUIPPED*. ▪ Tiempo máximo aceptado para el almacenamiento de los mensajes de *beaconing*, denominado *TIEMPO_MAX_DESCARTE_GEOCASTING*. ▪ Variable correspondiente con el número máximo de veces que un envío puede ser realizado, denominada *NUMERO_MAX_REENVIOS*.	Media	Baja

[1] Los nombres proporcionados a cada una de las variables únicamente tienen el propósito de facilitar la referencia a cada una de ellas en distintas partes del documento, no teniendo relación con los nombres que puedan ser especificados finalmente.

REQUISITOS DE SOFTWARE					
Tipo: funcional					
Id	**Ver.**	**Título**	**Descripción**	**Estabilidad**	**Prioridad**
RF-20	1.0	Configuración de variables	▪ Tiempos máximos transcurridos desde que un mensaje fue enviado hasta que es procesado en un nodo con rol Requester, Witness o NoEquipped, denominados *TIEMPO_MAX_RESPUESTA_REQUESTER*, *TIEMPO_MAX_RESPUESTA_WITNESS* y *TIEMPO_MAX_RESPUESTA_NOEQUIPPED*, respectivamente. ▪ Variable correspondiente con el tiempo máximo transcurrido desde que un mensaje ha sido creado hasta que es procesado tras haberse almacenado en la lista de mensajes almacenados por estar otro mensaje procesándose previamente, denominada *TIEMPO_DESCARTE_ALMACENADOS*. ▪ Variable correspondiente con la ruta del fichero en el que se encuentra la velocidad a la que circula cada nodo, denominada *RUTA_VELOCIDAD*. Este fichero se corresponde con el fichero de configuración de la simulación a ejecutarse, mencionado en el estudio del simulador (sección 3.2), el cual dispone de datos como la velocidad o la posición inicial de cada nodo. Además, la extensión es *.xtpl*. ▪ Variable correspondiente con el número máximo de saltos que un mensaje puede realizar hasta llegar el Requester, nodo solicitante, denominada *MAX_TTL*.	Media	Baja

REQUISITOS DE SOFTWARE					
Tipo: funcional					
Id	**Ver.**	**Título**	**Descripción**	**Estabilidad**	**Prioridad**
RF-20	1.0	Configuración de variables	▪ Variable correspondiente con el tiempo tras el cual se realizan los reenvíos de las solicitudes si no se ha recibido ningún testimonio, denominada *TIEMPO_MAX_REENVIOS.*	Media	Baja
RF-21	1.0	Configuración puertos comunicación, protocolo EVIGEN.	Los puertos configurables son: ▪ Puerto de recepción de mensajes de *beaconing* de nodos con rol Requester, Witness o NoEquipped. ▪ Puerto de envío de mensajes de Requester a Witness. ▪ Puerto de envío de mensajes de Witness o NoEquipped a Requester. ▪ Puerto de envío de mensajes entre NoEquipped. ▪ Puerto de envío de mensajes de Witness a NoEquipped. ▪ Puerto de recepción de mensajes de *beaconing* de RSU. ▪ Puerto de recepción de mensajes desde RSU. ▪ Puerto de envío de testimonio a RSU.	Media	Baja

Tabla 15: requisitos funcionales de la extensión de NCTUns para la simulación del protocolo EVIGEN.

3.9.2.2 Requisitos funcionales del sistema de visión de mensajes de la simulación

Los requisitos funcionales correspondientes con el sistema de visión de mensajes de la simulación se detallan en la Tabla 16.

Es necesario indicar que, para todo ellos, la fuente es el cliente y la necesidad es imprescindible.

REQUISITOS DE SOFTWARE					
Tipo: funcional					
Id	**Ver.**	**Título**	**Descripción**	**Estabilidad**	**Prioridad**
RF-22	1.0	Observación intercambio de mensajes Requester-Witness.	Mostrar los siguientes mensajes, intercambiados por nodos con rol Requester y Witness. • Envío/Reenvío de solicitud de testimonio de Requester a Witness. • Envío de testimonio de Witness a Requester.	Alta	Alta
RF-23	1.0	Observación intercambio de mensajes Witness-NoEquipped/RSU.	Mostrar el siguiente mensaje, intercambiado por los nodos con rol Witness y NoEquipped o RSU. • Envío de testimonio de Witness a NoEquipped o RSU.	Alta	Alta
RF-24	1.0	Observación intercambio de mensajes NoEquipped-NoEquipped.	Mostrar el siguiente mensaje, intercambiado por los nodos con rol NoEquipped y NoEquipped. • Envío de testimonio de NoEquipped a NoEquipped.	Alta	Alta
RF-25	1.0	Observación intercambio de mensajes NoEquipped-RSU.	Mostrar los siguientes mensajes, intercambiados por los nodos con rol NoEquipped y RSU. • Envío de testimonio de NoEquipped a RSU. • Recepción de testimonio en NoEquipped desde RSU.	Alta	Alta
RF-26	1.0	Observación de llegada notificación.	Mostrar los siguientes mensajes, intercambiados por nodos con rol RSU, Witness o NoEquipped. • Envío de notificación desde RSU. • Recepción de notificación en nodo Witness o NoEquipped desde RSU.	Alta	Alta
RF-27	1.0	Observación de evidencia enviada.	Mostrar el siguiente mensaje, intercambiado por nodos con rol RSU y Requester. • Envío de evidencia de Requester a RSU. • Recepción de evidencia en RSU.	Alta	Alta

REQUISITOS DE SOFTWARE					
Tipo: funcional					
Id	**Ver.**	**Título**	**Descripción**	**Estabilidad**	**Prioridad**
RF-28	1.0	Observación del contenido de mensajes.	Mostrar el contenido de los mensajes indicados en RF-22, RF-23, RF-24, RF-25, RF-26 y RF-27.	Alta	Alta
RF-29	1.0	Observación del estado de nodo, recibido en los mensajes de *beaconing*.	Mostrar la velocidad, la posición, el identificador y el tipo de nodo, en cada momento, por cada uno de los nodos que desempeñen el rol de Requester, Witness, NoEquipped o RSU.	Alta	Alta
RF-30	1.0	Observación vecindad.	Indicar la información de estado de cada vecino, por cada nodo que desempeñe el rol de Requester, Witness, NoEquipped o RSU.	Alta	Alta
RF-31	1.0	Parar sistema.	Parar en el momento que se desee.	Alta	Alta
RF-32	1.0	Pausar el sistema.	Pausar el sistema en el momento que se desee	Alta	Media
RF-33	1.0	Continuar el sistema.	Continuar la ejecución del sistema de visión de mensajes de la simulación en el momento que se desee, tras haber realizado una pausa del mismo.	Alta	Media
RF-34	1.0	Almacén mensajes.	Ofrecer la posibilidad de almacenar los mensajes intercambiados por cada uno de los nodos, así como su contenido. El almacenamiento se realizará en un fichero con formato PDF.	Media	Media
RF-35	1.0	Limpieza de pantalla.	Limpiarlos mensajes existentes en pantalla, en el momento que se desee.	Alta	Baja
RF-36	1.0	Observación de nodos con certificado revocado.	Mostrar la lista de los nodos cuyo certificado está revocado.	Alta	Alta

REQUISITOS DE SOFTWARE					
Tipo: funcional					
Id	**Ver.**	**Título**	**Descripción**	**Estabilidad**	**Prioridad**
RF-37	1.0	Configuración de puertos, sistema de visualización de mensajes.	Los puertos configurables son: ▪ Puerto de envío de información de estado desde la simulación ejecutada en el simulador NCTUns 5.0 (1) al sistema de visión de mensajes la simulación. ▪ Puerto de envío de mensajes desde el simulador NCTUns 5.0 (1) al sistema de visión de mensajes la simulación. ▪ Puerto de envío de la CRL desde el simulador NCTUns 5.0 (1) al sistema de visión de mensajes la simulación.	Alta	Media

Tabla 16: requisitos funcionales del sistema de visión de mensajes de la simulación.

3.9.3 Requisitos inversos

Los requisitos inversos presentes en este sistema se muestran en la Tabla 17.

Conviene indicar que este tipo de requisitos no requieren la especificación de la necesidad ni de la prioridad. Sin embargo, hay que señalar que la fuente de todos los requisitos inversos presentados es el equipo de desarrollo.

REQUISITOS DE SOFTWARE					
Tipo: inverso					
Id	**Ver.**	**Título**	**Descripción**	**Estabilidad**	**Prioridad**
RI-38	1.0	DGT y AC no presentadas en interfaz.	Los mensajes intercambiados entre los nodos con rol DGT y AC no podrán ser visualizados puesto que el protocolo está centrado en los nodos con roles Requester, Witness y NoEquipped y RSU en el momento que tome el rol de NoEquipped o reciba o envíe una notificación de sanción.	Alta	-
RI-39	1.0	Recepción de notificaciones limitadas.	Una notificación no puede ser recibida en un nodo antes de que dicho nodo finalice el tratamiento de una notificación de sanción recibida con anterioridad.	Alta	-

Tabla 17: requisitos inversos.

3.9.4 Requisitos no funcionales

Estudiados los requisitos funcionales e inversos, en esta sección se presentan los no funcionales, que complementan el sistema, y que se corresponden con rendimiento, interfaz, manejo de errores, mantenibilidad y seguridad.

Al igual que en otras ocasiones se divide en los requisitos correspondientes con el protocolo EVIGEN y los asociados al sistema de visión de mensajes de la simulación.

3.9.4.1 Requisitos no funcionales de la extensión de NCTUns para la simulación del protocolo EVIGEN

Los requisitos correspondientes con esta sección del proyecto se presentan en la Tabla 18.

Es conveniente indicar que, para todos ellos, la fuente es el equipo de desarrolladores y que la necesidad es imprescindible.

REQUISITOS DE SOFTWARE					
Tipo: rendimiento					
Id	**Ver.**	**Título**	**Descripción**	**Estabilidad**	**Prioridad**
RNF-40	1.0	Sobrecarga de nodos controlada.	Establecer las medidas adecuadas para que un nodo no pueda procesar más de un mensaje simultáneamente. De este modo, los mensajes recibidos mientras uno de ellos esté procesándose se han de almacenar para su posterior procesamiento.	Alta	Alta
Tipo: interfaz					
Id	**Ver.**	**Título**	**Descripción**	**Estabilidad**	**Prioridad**
RNF-41	1.0	Comunicación.	Establecer un modo de comunicación entre los nodos creados en el protocolo y el mismo u otro modo de comunicación entre el protocolo y el sistema de visión de mensajes.	Alta	Alta
Tipo: manejo de errores					
Id	**Ver.**	**Título**	**Descripción**	**Estabilidad**	**Prioridad**
RNF-42	1.0	Descarte si exceso de tiempo.	Descartar los mensajes que sobrepasen un cierto tiempo tras haber sido creados, comprendido entre 1 y 10 segundos.	Alta	Alta
RNF-43	1.0	Descarte de mensajes contestados.	Descartar los mensajes de solicitud que hayan sido contestados con anterioridad, un número máximo de veces, entre 0 y 20, en nodos que desempeñen el rol de Witness, Requester, o NoEquipped.	Alta	Alta
RNF-44	1.0	Descarte de solicitudes.	Descartar, en los nodos que desempeñen el rol de Witness, las solicitudes provenientes de un nodo que disponga de un certificado revocado.	Alta	Alta
RNF-45	1.0	Descarte de mensajes si obtención de certificado incorrecto.	Descartar, en cualquiera de los nodos, los mensajes en los que la obtención de un certificado no haya sido correcta.	Alta	Alta
RNF-46	1.0	Descarte de mensajes si comprobación de certificado incorrecto.	Descartar, en cualquiera de los nodos, los mensajes en los que la comprobación de un certificado no sea correcta.	Alta	Alta

REQUISITOS DE SOFTWARE					
Tipo: manejo de errores					
Id	**Ver.**	**Título**	**Descripción**	**Estabilidad**	**Prioridad**
RNF-47	1.0	Descarte de mensajes si verificación firma incorrecta.	Descartar, en cualquiera de los nodos, si la operación de verificación de firma no resulta exitosa.	Alta	Alta
RNF-48	1.0	Descarte de mensajes si obtención de clave incorrecta.	Descartar, en cualquiera de los nodos, los mensajes en los que la obtención de una clave no sea correcta.	Alta	Alta
RNF-49	1.0	Descarte de mensajes si comprobación de resumen incorrecta.	Descartar, en cualquiera de los nodos, los mensajes en los que las operaciones de comprobación de resúmenes no sean exitosas.	Alta	Alta
RNF-50	1.0	Descarte si exceso TTL.	Descartar, en los nodos que desempeñan el rol de NoEquipped o Requester, los mensajes cuyo TTL sea inferior a o igual a 0. El TTL se inicializa con un valor entre 3 y 20.	Alta	Alta
RNF-51	1.0	Descarte de mensaje de solicitud si nodo en CRL.	Descartar los mensajes de solicitud recibidos en un Witness siempre que el Requester que envió dicha solicitud se encuentre en la CRL.	Alta	Alta
RNF-52	1.0	Comprobación duplicidad de puertos.	Comprobar que ninguno de los puertos establecidos en el fichero de configuración esté duplicado.	Alta	Alta
RNF-53	1.0	Comprobación de puertos.	Comprobar que ninguno de los puertos establecidos en el fichero de configuración contenga números menores a 1024 o mayores a 65536.	Alta	Alta
RNF-54	1.0	Comprobación reenvío solicitud.	Comprobar cada cierto tiempo, comprendido entre 1 y 15 segundos, si se ha realizado alguna recepción de testimonio en el nodo con rol Requester.	Alta	Alta

REQUISITOS DE SOFTWARE					
Tipo: mantenibilidad					
Id	**Ver.**	**Título**	**Descripción**	**Estabilidad**	**Prioridad**
RNF-55	1.0	Modularidad del sistema.	Implementar las funcionalidades del sistema en módulos que faciliten el mantenimiento, la extensibilidad, y la reusabilidad. De este modo, se asegura que el sistema sea mantenible, que pueda ser ampliado con nuevas funcionalidades y que, partes de él, puedan ser utilizadas de forma independiente para integrarse en otros sistemas.	Alta	Alta
Tipo: seguridad					
Id	**Ver.**	**Título**	**Descripción**	**Estabilidad**	**Prioridad**
RNF-56	1.0	Integridad y autenticidad.	Utilizar operaciones criptográficas en mensajes que contengan información que pueda ser modificada, de forma que los mensajes no puedan ser modificados por terceras personas.	Alta	Alta

Tabla 18: requisitos no funcionales de la extensión de NCTUns para la simulación del protocolo EVIGEN.

3.9.4.2 Requisitos no funcionales del sistema de visión de mensajes de la simulación

Los requisitos correspondientes con esta sección del proyecto se presentan en la Tabla 19.

Es conveniente indicar que, para todos ellos, la fuente es el equipo de desarrolladores y que la necesidad es imprescindible.

REQUISITOS DE SOFTWARE					
Tipo: interfaz					
Id	**Ver.**	**Título**	**Descripción**	**Estabilidad**	**Prioridad**
RNF-57	1.0	Formato de mensaje.	Representar en formato NVT ASCII tanto los mensajes utilizados en la comunicación de los nodos participantes en el protocolo como los presentados en el sistema de visión de mensajes de la simulación.	Alta	Alta
RNF-58	1.0	Formato PDF.	Utilizar el formato PDF para la presentación de los mensajes intercambiados en la simulación, así como el contenido de los mismos.	Media	Alta
RNF-59	1.0	Opciones del sistema de visión de mensajes de la simulación.	Ofrecer opciones que permitan almacenar información en PDF, arrancar, parar, pausar/continuar la visualización de una simulación, configurar puertos y obtener ayuda.	Media	Alta
RNF-60	1.0	Habilitación/deshabilitación de funciones.	Indicar las funciones que pueden ser realizadas y aquellas que no.	Alta	Media
RNF-61	1.0	Mensajes de error.	Presentar un cuadro de diálogo que muestre los errores cometidos y advertencias que puedan ocurrir tanto previamente al arranque del sistema como durante o posteriormente.	Alta	Alta
RNF-62	1.0	Elección de almacenamiento en PDF.	Permitir elegir el nodo o nodos de los que se desea almacenar, en PDF, los mensajes intercambiados.	Alta	Alta

REQUISITOS DE SOFTWARE					
Tipo: interfaz					
Id	**Ver.**	**Título**	**Descripción**	**Estabilidad**	**Prioridad**
RNF-63	1.0	Formato del contenido de mensajes.	Presentar los mensajes intercambiados durante una simulación, indicando la siguiente información: • Receptor/Emisor del mensaje. • Identificador del mensaje • TTL del mensaje. • Marca de tiempo del mensaje. • Velocidad a la que un nodo es sancionado/ Velocidad establecida en un evidencia para evitar la sanción.	Alta	Alta
RNF-64	1.0	Formato del contenido de mensajes en PDF.	Presentar en PDF los mensajes intercambiados durante una simulación indicando la información indicada en RNF-63, pero incluyendo las operaciones criptográficas que se realicen.	Alta	Alta
RNF-65	1.0	Establecimiento de manual.	Establecer un manual para poder resolver las dudas que puedan surgir.	Media	Media
REQUISITOS DE SOFTWARE					
Tipo: manejo de errores					
Id	**Ver.**	**Título**	**Descripción**	**Estabilidad**	**Prioridad**
RNF-66	1.0	Comprobación campos guardar.	Comprobar que todos los campos para poder crear un fichero con extensión PDF (la ruta y nombre del fichero) son establecidos correctamente.	Alta	Alta

Tabla 19: requisitos no funcionales del sistema de visión de mensajes de la simulación.

3.10 Diseño del plan de pruebas de aceptación

Realizado un estudio de las características del sistema a desarrollar, en esta sección se va a realizar una descripción tanto de las pruebas que se han de efectuar como de los resultados que se han de obtener tras la realización de cada una de ellas.

3.10.1 Formato para la realización de pruebas de aceptación

En esta sección se han de establecer los formatos adecuados tanto para indicar las pruebas a realizar como para indicar el resultado de la realización de las mismas.

En la tabla posterior, Tabla 20, se presenta el formato en el que se mostrarán las pruebas que se han de realizar:

PRUEBAS DE ACEPTACIÓN				
Id	**Ver.**	**E. prueba**	**Entrada**	**Salida**

Tabla 20: formato pruebas de aceptación.

El propósito de cada uno de los campos es:

- Id: identificador de la prueba. El formato será P-XX, correspondiendo los XX con dígitos comenzando por el 01.
- Ver.: versión actual de la prueba. El formato es X.Y, donde X se corresponde con la versión e Y se corresponde con la revisión.
- E. prueba: elemento que ha de ser probado.
- Entrada: elementos de entrada necesarios para la ejecución de la prueba.
- Salida: elementos de salida que se esperan obtener tras la ejecución de la prueba.

Posteriormente, en la siguiente tabla, Tabla 21, se indica el formato en el que se ha de presentar el resultado obtenido tras la realización de las pruebas:

RESULTADOS PRUEBAS DE ACEPTACIÓN		
Id	**Ver.**	**Resultado**

Tabla 21: formato resultado pruebas aceptación.

El objetivo de cada uno de los campos es:

- Id: identificador de la prueba. El formato será P-XX, correspondiendo los XX con dígitos comenzando por el 01.
- Ver.: versión actual de la prueba. El formato es X.Y, donde X se corresponde con la versión e Y se corresponde con la revisión.

- Resultado: resultado obtenido tras la realización de la prueba con identificador "Id". Los valores que pueden ser utilizados son "Superada", si la prueba se ha realizado exitosamente e "Fallida", si la prueba no ha sido superada.

3.10.2 Pruebas de aceptación

Las pruebas especificadas se presentan en la primera sección del anexo asociado a este documento (3).

4 Diseño detallado

Finalizado el análisis, en esta sección se presenta la fase de diseño detallado. En este punto se describen los principales diagramas que muestran la estructura y el funcionamiento de los artefactos software que se desarrollan en el presente proyecto para dar respuesta a las necesidades expuestas en la sección 1.1.

En particular, en la sección 4.1 se muestran los diagramas de clases y la descripción de cada uno de ellos, en relación con el componente al que están asociadas, los cuales fueron presentados en la arquitectura definitiva (sección 3.6). Por otra parte, en la sección 4.2 se muestran los diagramas de secuencia, proporcionando una visión global del camino a recorrer para llevar a cabo la correcta ejecución de cada una de las funcionalidades establecidas.

4.1 Diseño del software

Tal y como se ha realizado en otras ocasiones, esta sección, en la que se presenta la descripción de los componentes establecidos, se va a dividir teniendo en cuenta que el proyecto engloba la extensión de NCTUns para la simulación del protocolo EVIGEN y la creación del sistema de visión de mensajes de la simulación. Debe recordarse que el visor de mensajes se implementa de forma independiente al conjunto del simulador ya que no se dispone del código de la interfaz de usuario.

Antes de comenzar con la descripción del diseño, conviene subrayar que, por claridad, no se mencionarán los métodos relacionados con la asignación y obtención del valor de las variables (i.e. métodos "get" y "set").

4.1.1 Diseño del sistema de visión de mensajes de la simulación

En las siguientes secciones se estudiarán cada uno de los componentes establecidos en la sección 3.6.1, correspondientes con *Vista*, *Modelo*, subdividido en tres, y *Controlador*.

Además, en la siguiente imagen, Ilustración 19, equivalente a la presentada en la sección indicada en el párrafo anterior, se señalan las partes en las que se va a centrar el desarrollo de este diseño.

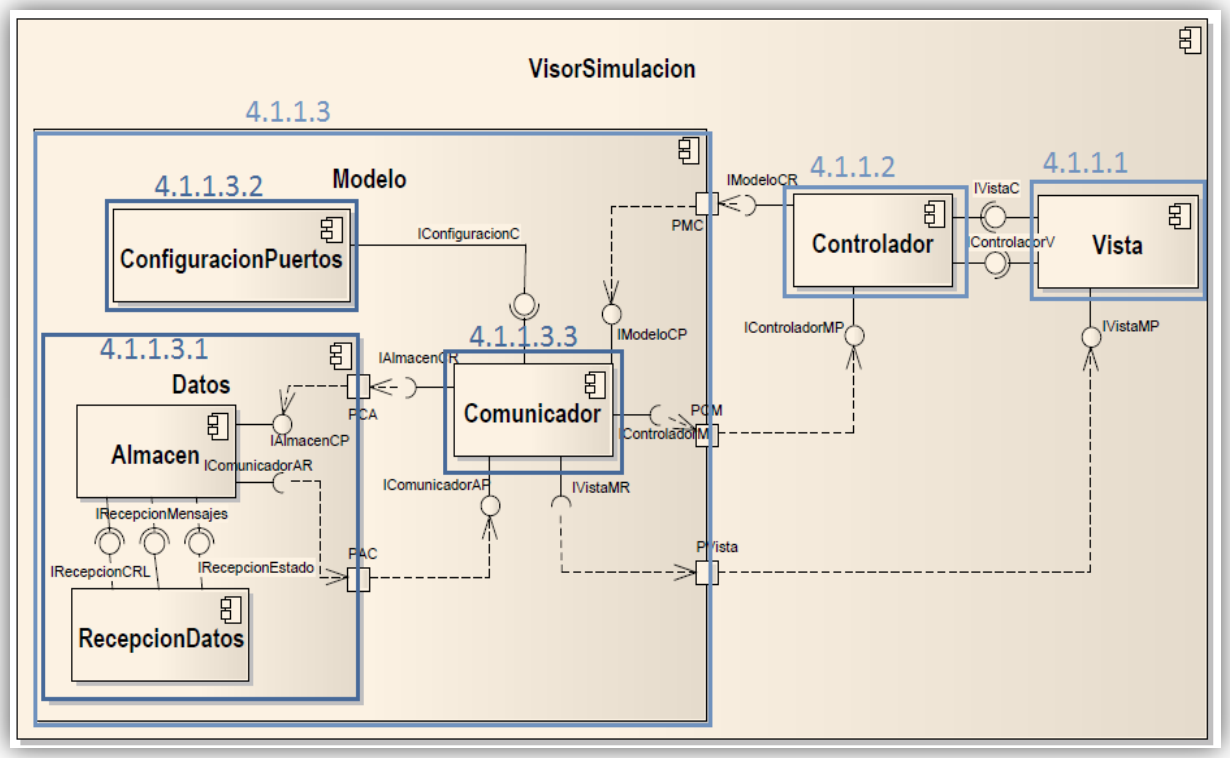


Ilustración 19: división de componentes del sistema de visión de mensajes de la simulación.

4.1.1.1 Diseño del componente Vista

Para comenzar, el componente *Vista* se corresponde con la implementación de la interfaz de visualización de los mensajes.

Considerando que las clases de *Vista son* "Simulacion" e "ISimulacion", se presentan las operaciones que han de implementarse:

- Método "escogerNodo": elección de un nodo entre todos los participantes en una simulación.
- Método "limpiar": limpieza de la pantalla en la que se presentan los mensajes intercambiados en la simulación del protocolo EVIGEN por un nodo en concreto.
- Método "observarCRL": observación de la lista de identificadores de nodo cuyo certificado está revocado.
- Método "almacenarPDF": almacenamiento en un fichero PDF de los mensajes intercambiados en la simulación del protocolo EVIGEN por un nodo en concreto.
- Método "pintarInfoEstado": presentación en pantalla de todos los datos relacionados tanto con la información de estado de cada nodo, como con la información de estado de cada uno de los nodos considerados vecinos de uno indicado. Esta información se corresponde con:
    - Id del nodo.
    - Velocidad del nodo, en caso de no ser RSU no es necesario mostrar un valor.
    - Posición en el eje X en la que se encuentra el nodo en la pantalla de simulación del simulador NCTUns 5.0 (1).
    - Posición en el eje Y en la que se encuentra el nodo en la pantalla de simulación del simulador NCTUns 5.0 (1).
    - Tipo de vehículo/rol del nodo: "R" Requester, "W" Witness, "O" NoEquipped y "U" RSU.
    - Información de vecindario, para todos los nodos que se consideren vecinos a uno dado y que no desempeñen el rol de RSU. La información se corresponde con:
        - Id del nodo.
        - Velocidad del nodo.
        - Posición en el eje X.
        - Posición en el eje Y.
        - Tipo de vehículo/rol.

En cuanto a la información de estado de las RSU, identificador y posición, se dispone de los campos mostrados anteriormente para su presentación. Por tanto, aquellos de los que no se reciba información se han de completar con algún carácter que lo indique, como pueden ser "?" o "-".

- Método "pintarNotificaciones": presentación en pantalla de todos los mensajes asociados al nodo mostrado en la interfaz gráfica. Es importante tener en cuenta el requisito no funcional RNF-63, de modo que se presenten por pantalla:
  - Receptor/Emisor del mensaje.
  - Identificador del mensajes.
  - TTL del mensaje.
  - Marca de tiempo del mensaje.

  Además, en los mensajes de recepción de la notificación de sanción o envío de evidencia, también debe mostrarse la velocidad. Del mismo modo, en los mensajes correspondientes con los testimonios, siempre que se disponga de la información adecuada, se debe mostrar el intervalo de velocidad establecido por cada nodo Witness con respecto a una solicitud realizada.

  Finalmente, señalar que, tal y como se ha comentado con respecto a la información de estado de las RSU, los campos que no puedan ser completados por falta de datos se han de rellenar con un carácter que lo indique, como puede ser "?" o "-".

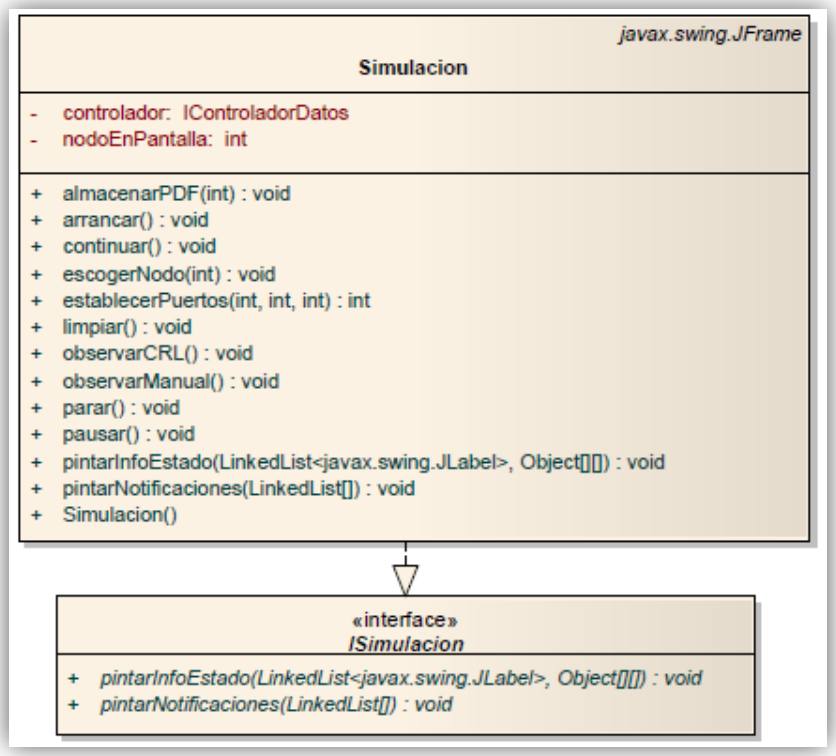


Ilustración 20: diseño componente Vista.

4.1.1.2 Diseño del componente Controlador

Posteriormente al estudio de la *Vista*, en esta sección se presenta el diseño del componente *Controlador*, el cual fue presentado en la sección 3.6.1.

En este componente, Ilustración 21, compuesto por la clase "ControladorDatos", cuya interfaz es "IControladorDatos", se caracteriza por efectuar la función de intermediario entre las solicitudes realizadas por la *Vista* al *Modelo*, quedando las operaciones más estructuradas y facilitando modificaciones posteriores. De acuerdo con su función, las operaciones que en él se incluyen se asocian con las situadas dentro de "IComunicacionInterfaz" (componente *Comunicador*), que son arrancar, parar, continuar, reanudar la ejecución, crear el fichero PDF, obtener datos, obtener la CRL y obtener los nodos participantes en una simulación ejecutada en NCTUns 5.0.

Además, indicar que es en cada uno de los métodos de la clase "ControladorDatos" donde se verifica la corrección de los datos y de las acciones realizadas por el *Modelo*. Tomando en consideración lo comentado, se elimina de la *Vista* la función de controlar las salidas obtenidas por cada método invocado, quedando centralizado el control de errores en este componente.

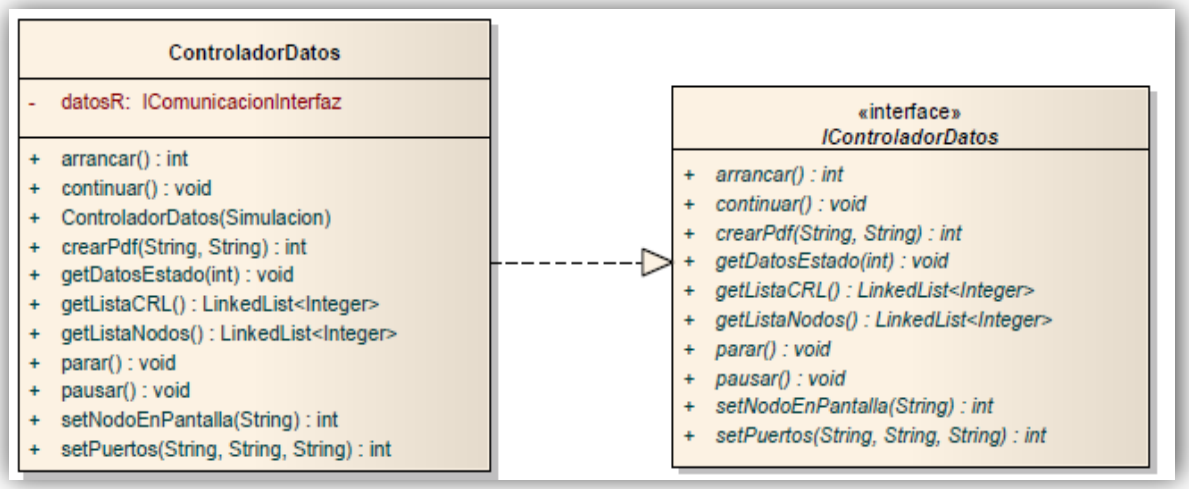


Ilustración 21: diseño componente Controlador.

4.1.1.3 Diseño del componente Modelo

Finalmente, se encuentra el componente *Modelo*, en el que se realiza el almacenamiento y gestión de los datos recibidos desde el simulador NCTUns 5.0 (1), correspondientes con la ejecución de cada una de las simulaciones del protocolo EVIGEN. A la vista del diagrama de componentes de la Ilustración 19, se puede apreciar la existencia de tres subcomponentes, uno de ellos subdividido en dos. Por tanto, en esta sección se expone la descripción del diseño de cada uno de dichos componentes, a saber, *Datos*, *ConfiguracionPuertos* y *Comunicador*.

4.1.1.3.1 Diseño del componente Datos

En este componente, *Datos*, subdividido en dos, se resumen las funciones más importantes del sistema, que son la recepción de datos desde el simulador NCTUns 5.0 (1), componente *RecepcionDatos*, y el almacenamiento y procesamiento de la misma, componente *Almacen*.

Por una parte, coincidiendo con las cinco clases situadas en la parte inferior de la Ilustración 22, se presenta el componente *RecepcionDatos* en el que se produce la recepción de tres tipos distintos de datos, la lista de identificadores de nodos con certificado revocado, los mensajes intercambiados en la simulación y la información de estado enviada en los mensajes de *beaconing* junto con la lista de nodos participantes en cada simulación, mencionados en la sección 3.6.1. Además, es conveniente indicar que para la recepción de cada uno de los tipos se dispone de un hilo de ejecución que permite abordar, de forma simultánea, la recepción de dichos datos junto con la realización de tareas de procesamiento.

Seguidamente se presentan las distintas clases comprendidas en este componente.

Por una parte se sitúa la clase "RecibirDatos", responsable de la recepción tanto de la información de estado de cada nodo como de la lista de nodos considerados vecinos y la lista de nodos participantes en la simulación.

Por otra parte, otro de los tipos de datos son los mensajes intercambiados en la ejecución del protocolo EVIGEN, los cuales son recibidos en la clase "RecibirNotificaciones".

Por otro lado, el último tipo de dato que puede ser recibido se corresponde con la lista de identificadores de nodos cuyo certificado está revocado, clase "RecibirCRL".

Finalmente, la clase "RecibirInfoSimulador", cuya interfaz es "IRecibirInfoSimulador", es la encargada de proporcionar el acceso a cada una de las clases mencionadas en párrafos anteriores.

Las operaciones contenidas en cada una de las clases se pueden describir conjuntamente:

- Método "arrancar": arranque de cada uno de los hilos asociados a los distintos tipos de datos que pueden ser recibidos.
- Método "parar": parada de cada uno de los hilos asociados a los distintos tipos de datos que pueden ser recibidos.
- Método "run": asociado a cada hilo de ejecución y encargado de la entrega de la información recibida a la clase apropiada, mediante los métodos "reciboDatos", "reciboCRL" o "reciboNotificaciones".

En cuanto al componente *Almacen*, cuyas clases se corresponden con las cinco presentadas en la zona superior de la Ilustración 22, indicar que es responsable del procesamiento y almacenamiento de la información proporcionada por el componente *RecepcionDatos*, la cual se guarda para una posterior presentación en la interfaz gráfica, por medio del componente *Vista*. En los siguientes párrafos se presentan cada una de las clases que comprenden este componente.

Por un lado, se encuentra la clase "NotificacionesRecibidas", encargada del procesamiento de los mensajes intercambiados entre los nodos del protocolo (excepto los de *beaconing* y CRL), recibidos mediante "RecibirNotificaciones". Además, es importante indicar la creación de un hilo de ejecución para el envío a la interfaz gráfica de los mensajes recibidos en cada momento, siempre que dichos mensajes estén asociados al nodo situado en pantalla, del cual se deseen observar los intercambios de información realizados. Las operaciones comprendidas en esta clase son:

- Método "reciboNotificaciones": recepción de los mensajes enviados desde una simulación en la que se ejecuta el protocolo EVIGEN, por medio de "run", situado en "RecibirNotificaciones". En este lugar se procesa el mensaje y se invoca "anyadirEtiqueta" para realizar el almacenamiento del mismo.

- Método "anyadirEtiqueta": creación de los elementos correspondientes con el mensaje recibido, así como su posterior almacenamiento. Todo esto se realiza teniendo en cuenta cada uno de los campos mencionados en "pintarNotificaciones", componente *Vista*.

- Método "crearPdf": creación del fichero PDF con nombre dado y correspondiente con el o los elementos almacenados y asociados a cada uno de los mensajes recibidos. Esto se realiza haciendo uso de la librería itext (11) pero sin ninguna restricción en cuanto a la utilización de la misma. Además, se debe considerar lo comentado en el componente *Vista*.

- Método "limpiarDatos": eliminación de toda la información de los mensajes recibidos y almacenados previamente.

Por otro lado, en este componente se realiza el procesamiento y recepción tanto de la lista de nodos participantes en la simulación como de la información de estado de cada uno de ellos, obtenido a través de "RecibirDatos". Asimismo, es importante indicar la creación de un hilo de ejecución para el envío a la interfaz gráfica de la información de estado y de la información de vecindad recibida en cada momento, siempre que dicha información esté asociada al nodo situado en pantalla. La clase en la que se efectúa esta funcionalidad es "DatosRecibidos". Las operaciones realizadas en esta clase son las siguientes:

- Método "reciboDatos": recepción de la lista de nodos o de la información de estado y vecindario, procedente de la ejecución del protocolo EVIGEN.

Dependiendo del tipo de mensaje obtenido, bien la lista de nodos o bien la información de estado, se realiza la invocación de "crearPanelEleccion" o "crearListaCoches", respectivamente.

- Método "crearPanelEleccion": creación de la lista con los nodos participantes en la simulación, para, posteriormente, ser enviados al componente *Controlador* debido a una solicitud realizada desde el componente *Vista*.

- Método "crearListaCoches": creación de los elementos correspondientes con la información a presentar por pantalla, la cual se ha indicado en "pintarInfoEstado", componente *Vista*, junto con su posterior almacenamiento.

- Método "limpiarDatos": eliminación de toda la información recibida y almacenada previamente.

Otra clase vinculada a este componente es "CRLRecibidas", la cual es responsable del procesamiento de la CRL entregada desde "RecibirCRL". Los métodos comprendidos en esta clase son:

- Método "reciboDatos": recepción de la lista de identificadores de nodos con certificado revocado, procedente de la ejecución del protocolo EVIGEN, junto con su posterior procesamiento y almacenamiento.

- Método "limpiarDatos": eliminación de toda la lista de identificadores, recibida y almacenada previamente.

Por otro lado, se puede apreciar la clase "InfoSimulacionRecibida", cuya interfaz es "IInfoSimulacionRecibida", responsable de proporcionar el acceso a cada una de las clases del componente *Almacen* previamente comentadas.

Por otra parte, conviene tener en cuenta que el motivo por el que únicamente se envía al componente *Vista* la información correspondiente con el nodo visualizado en un determinado momento, es el de conseguir que dicho componente muestre la información proporcionada sin necesidad de realizar ninguna comprobación o procesamiento previo.

Por último, conviene apreciar la utilización del patrón *Observer* (17) para realizar la notificación al componente *Vista* de cada una de las recepciones de datos.

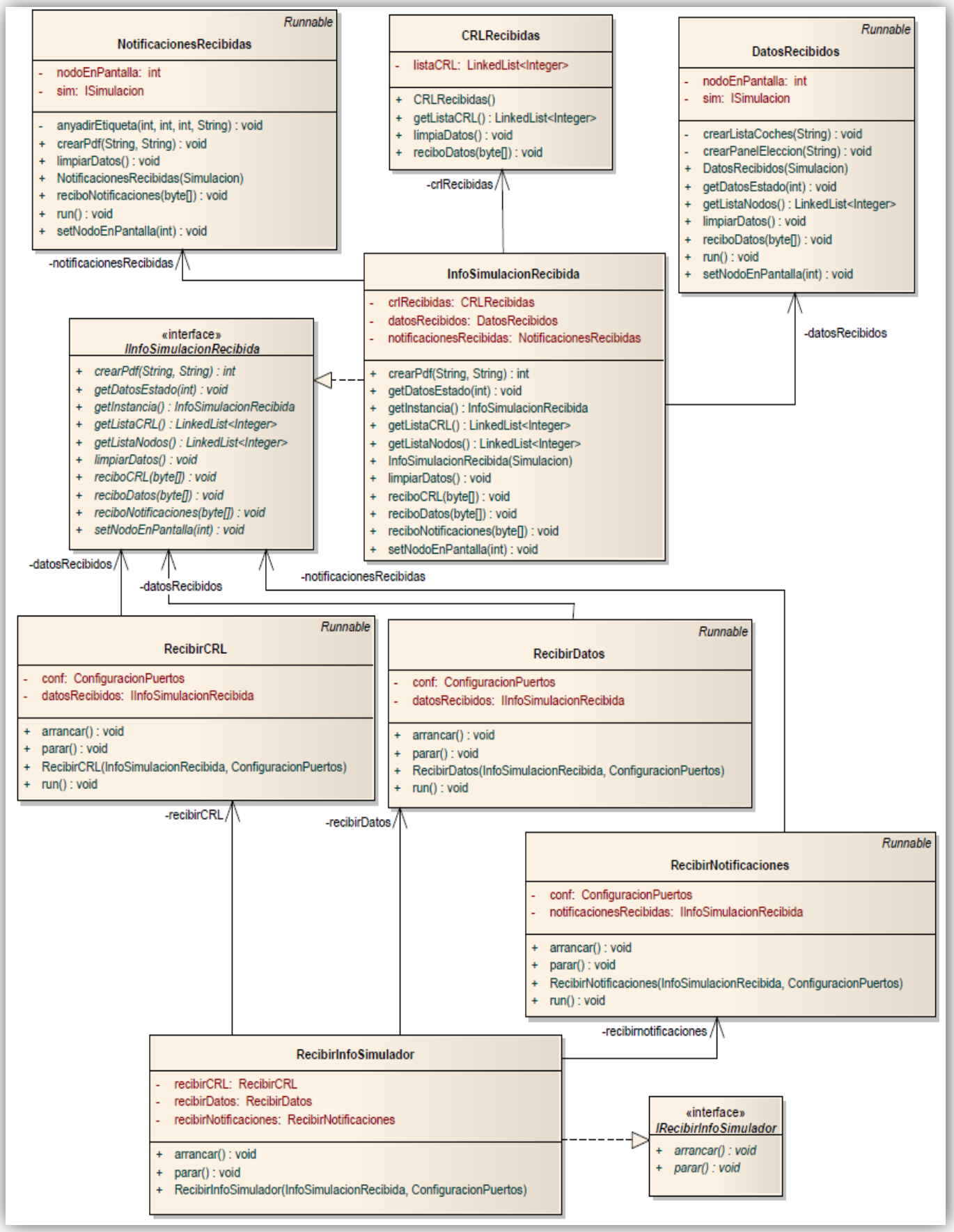


Ilustración 22: diseño componente Datos.

4.1.1.3.2 Diseño del componente ConfiguracionPuertos

Otro de los componentes comprendidos dentro de *Modelo*, es el de *ConfiguracionPuertos*, Ilustración 23, responsable de la obtención y actualización de los puertos de configuración y cuya clase es "ConfiguracionPuertos", con interfaz "IConfiguracionPuertos". Además, se establece la existencia de un fichero de configuración de puertos en el que se almacenen los puertos especificados en RF-37. Las operaciones comprendidas en este componente son:

- Método "leerConfiguracion": carga en memoria del valor de los puertos asociados a la recepción de la información recibida, bien procedentes de la interfaz gráfica o del fichero de configuración de puertos. En caso de ser modificados en la interfaz se realiza la actualización de los mismos en el fichero de configuración de puertos, invocando "actualizarFichero".

- Método "actualizarFichero": actualización del fichero de configuración de puertos con los valores establecidos en la interfaz gráfica.

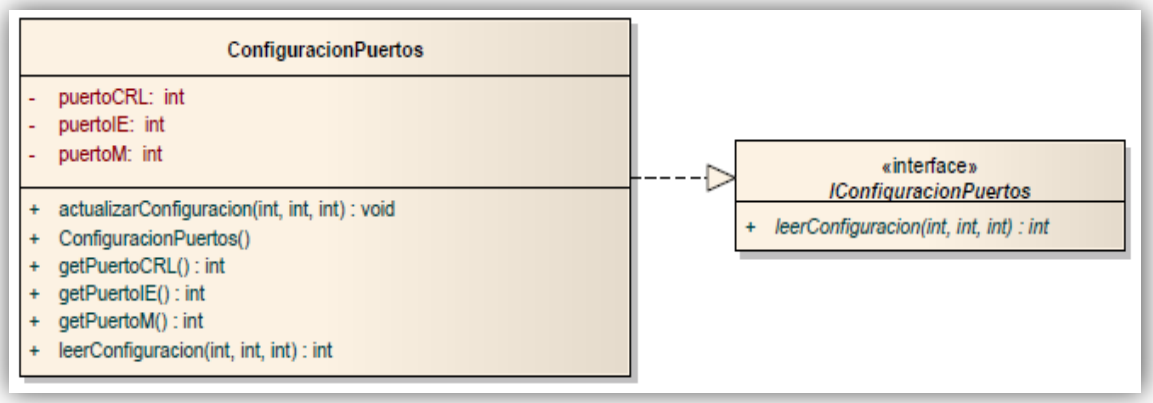


Ilustración 23: diseño componente ConfiguracionPuertos.

4.1.1.3.3 Diseño del componente Comunicador

En último lugar se encuentra el componente *Comunicador*, cuyo propósito es el de realizar la función de fachada, mostrando una única interfaz al cliente para llevar a cabo las distintas operaciones ofrecidas. Además, se consigue que el cliente se mantenga al margen del funcionamiento interno de las operaciones y se faciliten las modificaciones de las clases ocultas tras la fachada por ser las únicas afectadas por un cambio.

Las clases incluidas en este componente son "ComunicacionInterfaz" e "IComunicacionInterfaz", la interfaz asociada, la cual proporciona la interfaz hacia el cliente correspondiente con las funciones proporcionadas por el componente *Datos* y el componente *ConfiguracionPuertos*.

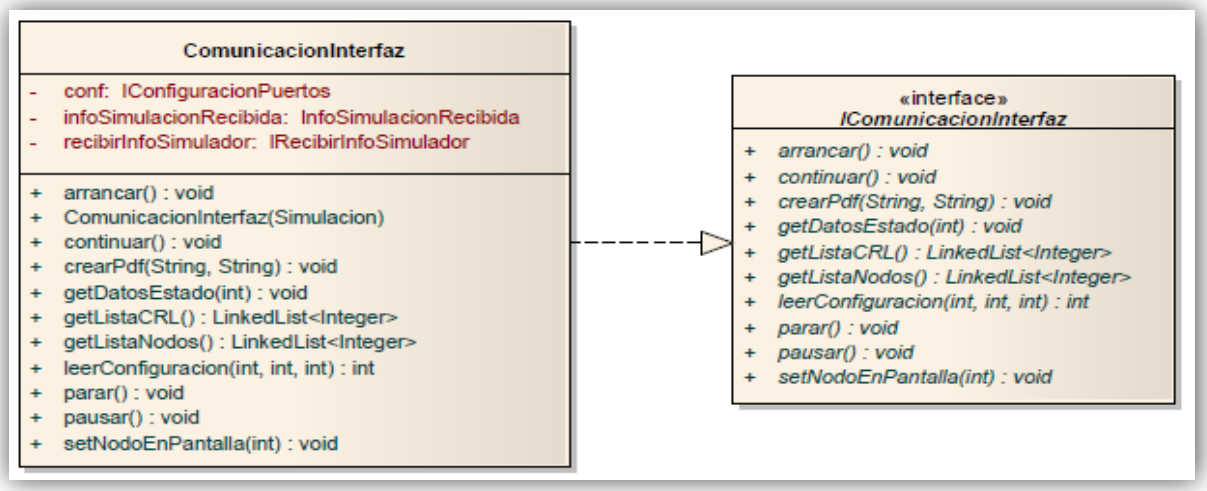


Ilustración 24: diseño componente Comunicador.

4.1.2 Diseño de la extensión de NCTUns para la simulación del protocolo EVIGEN

En las siguientes secciones se estudiarán cada uno de los componentes establecidos en la arquitectura definitiva asociada al desarrollo del protocolo EVIGEN, sección 3.6.2.

Conviene indicar que se van a presentar diez grandes secciones, correspondientes con los componentes que han sido señalados en la siguiente imagen, Ilustración 25, equivalente a la presentada en la arquitectura definitiva (sección 3.6.2).

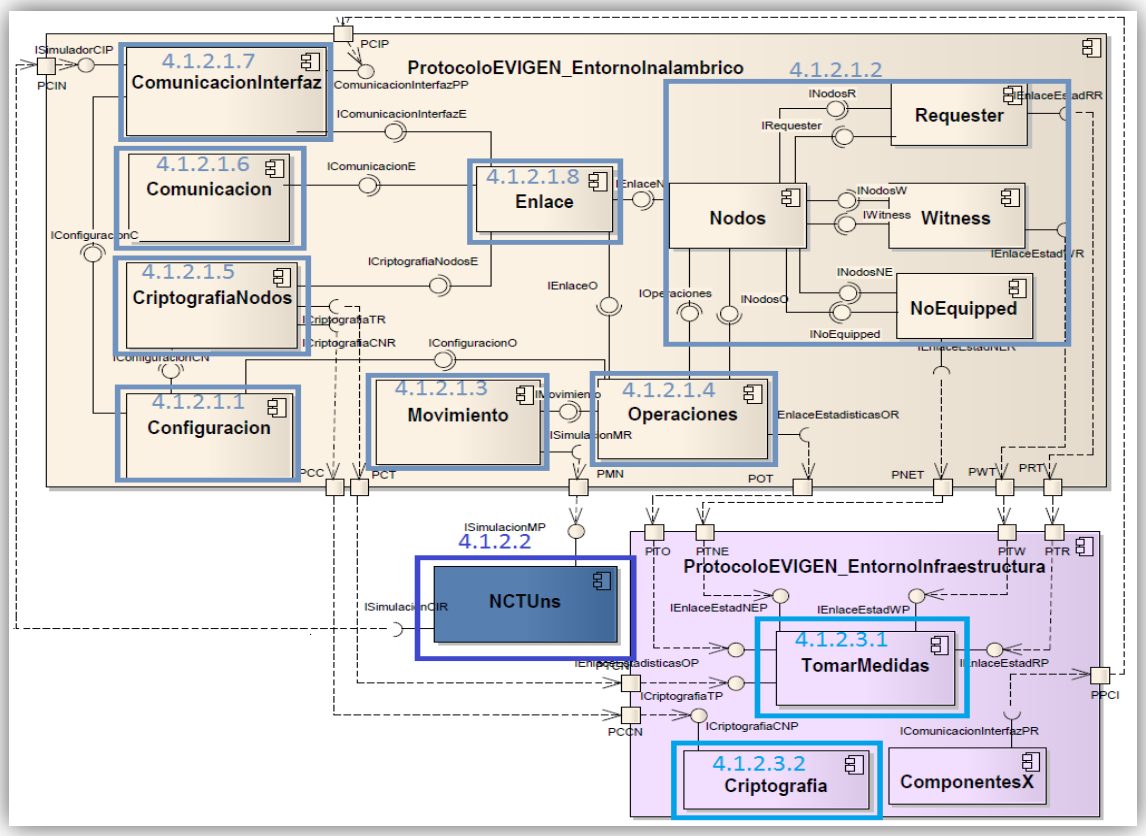


Ilustración 25: división de componentes de la extensión de NCTUns para la simulación del protocolo EVIGEN.

4.1.2.1 Diseño del componente ProtocoloEVIGEN_EntornoInalambrico

Previamente a la descripción de las clases incluidas en cada uno de los componentes que comprenden *ProtocoloEVIGEN_EntornoInalambrico*, resulta conveniente indicar la utilización de las siguientes estructuras, establecidas para facilitar la implementación de algunas de las funciones y pudiendo ser utilizadas según el programador lo considere oportuno:

- Estructura "Geocasting": responsable del almacenamiento de los datos de información de estado de cada uno de los nodos.
- Estructura "Testimonio": responsable del almacenamiento de los datos enviados en cada uno de los mensajes intercambiados en el protocolo EVIGEN.
- Estructura "Velocidad": responsable del almacenamiento del intervalo de velocidad resultante tras el procesamiento de cada uno de los testimonios recibidos en un nodo con rol Requester.
- Estructura "almacenMensajes": responsable del almacenamiento de los mensajes que no pueden ser procesados en un determinado momento, por estar otro mensaje procesándose.
- Estructura "Datos": responsable del almacenamiento de los datos que contienen la información de estado, junto con otros datos de soporte que se necesiten mantener vinculados a dicha información.
- Estructura "almacenDatos": responsable de la creación de una lista de estructuras "Datos".
- Estructura "PeticionesRecibidas": responsable del almacenamiento de los mensajes de notificación de sanción recibidos, o de un identificador de los mismos, para tener constancia de cada uno de ellos.
- Estructura "Notificacion": responsable del almacenamiento de los datos recibidos en cada notificación.
- Estructura "MensajeNotificacion": responsable del almacenamiento de una estructura "Notificacion" así como de los datos que se consideren adecuados para adjuntarlos en un mensaje de notificación.
- Estructura "CRL": responsable del almacenamiento de los datos relacionados con la CRL.
- Estructura "MensajeCRL": responsable del almacenamiento de una estructura "CRL" así como de los datos que se consideren adecuados para adjuntarlos en un mensaje de CRL.

4.1.2.1.1 Diseño del componente Configuracion

Para comenzar, en el componente *Configuracion* se localizan las lecturas tanto de puertos y variables de configuración como de certificados y clave privada utilizados en las operaciones criptográficas, Ilustración 26.

Por un lado, indicar la creación de la clase "Configuracion" asociada a la interfaz "IConfiguracion", la cual proporciona una interfaz común para el acceso a los elementos configurables, puertos, variables y rutas, de modo que se proporcione tanto funciones para la lectura de los elementos como para la obtención de cada uno de ellos.

Por otro lado, en la clase "ConfiguracionPuertos" se presentan las operaciones necesarias para conseguir los puertos de comunicación especificados en RF-21. Por tanto, se establece que en un fichero de configuración de puertos se han de almacenar cada uno de los puertos existentes, realizándose la obtención de cada uno de ellos por medio de la función "leerConfiguracion".

Por otro lado, en cuanto a las variables de configuración, clase "ConfiguracionVariables", se establece que en un fichero de configuración de variables se almacenen todas las necesarias, indicadas en RF-20, para posteriormente ser cargadas al sistema por medio de la función "leerVariables".

Finalmente, también es posible configurar la localización de los certificados y clave privada, clase "ConfiguracionRutas". Por ello, se establece que en un fichero de configuración de rutas se han de almacenar las rutas indicadas en RF-18 para, posteriormente, ser cargadas al sistema mediante la función "leerRutaFicheros".

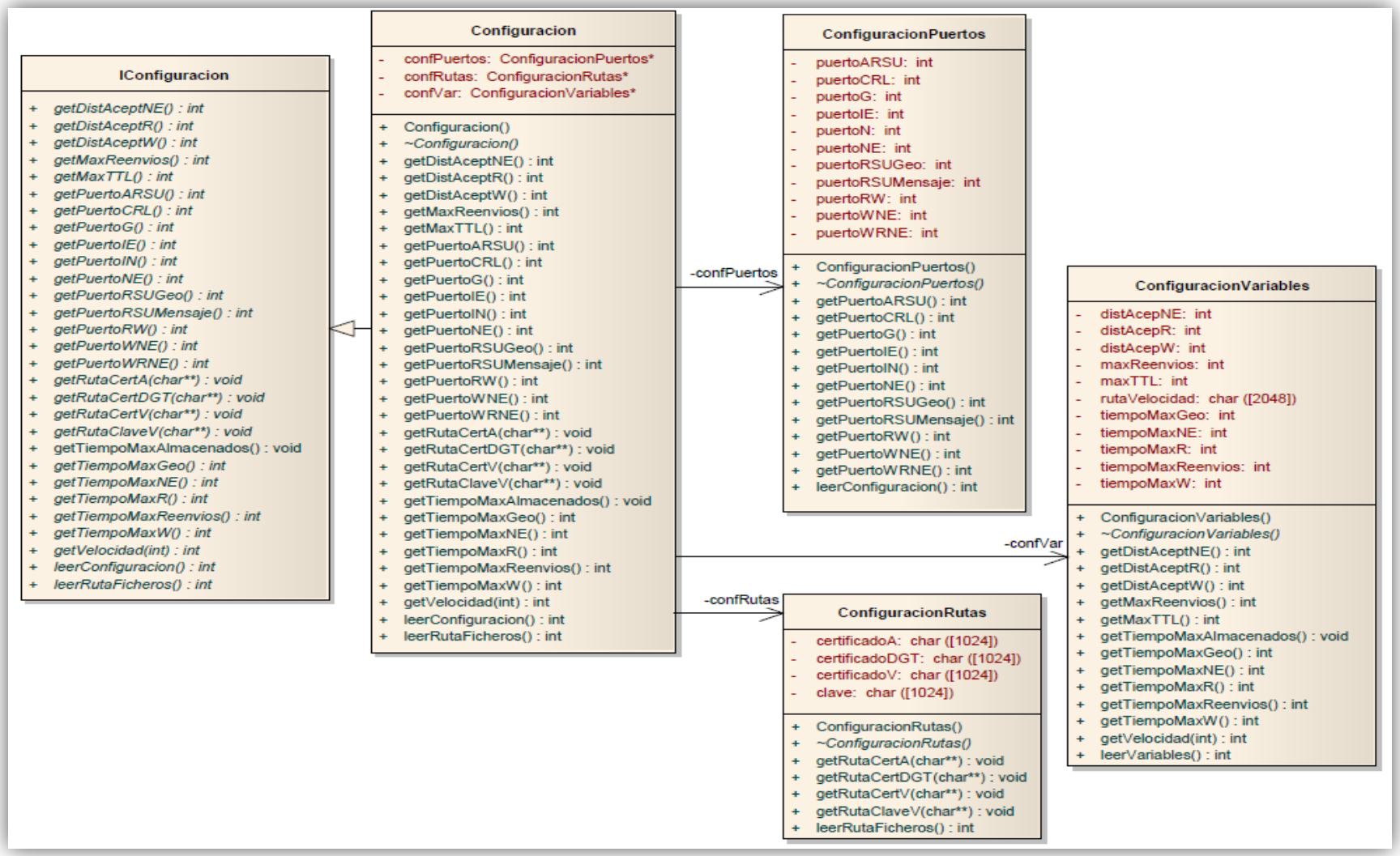


Ilustración 26: diseño componente Configuracion.

4.1.2.1.2 Diseño de los componentes Nodos, Requester, Witness y NoEquipped

Otros de los componentes participantes en este protocolo son *Nodos*, *Requester*, *Witness y NoEquipped*, los cuales se presentan conjuntamente para que el diseño sea comprendido con mayor claridad. La funcionalidad que caracteriza a los tres últimos componentes es la de centralizar las funcionalidades de los roles Witness, NoEquipped y Requester participantes en la simulación.

Previamente, considerando que a lo largo de esta sección se indicará la utilización de distintos hilos, la gestión de la concurrencia en cada uno ellos, acorde a lo especificado en la sección las tecnologías escogidas (sección 3.7.2), se debe realizar utilizando los *mutex* que se consideren apropiados.

Por un lado, en el componente *Nodos*, Ilustración 27, se encuentran las clases "Car" e "ICar", la interfaz asociada, en las que se concentran llamadas a las clases "IEnlace" (componente *Enlace*), "ICarsOperations" (componente *Operaciones*), "IRequester" (componente *Requester*), "IWitness" (componente *Witness*) e "INoEquipped" (componente *NoEquipped*), de modo que se proporcione una única interfaz de acceso para los componentes asociados a dichas clases. Por tanto, se puede indicar la utilización de un patrón *Facade* (17).

Por otro lado, las clases "IWitness" y "Witness" se corresponden con el componente *Witness*, las clases "IRequester", "Requester" y "ComprobarRecepcionTestigo" se corresponden con el componente Requester y las clases "INoEquipped" y "NoEquipped" se corresponden con el componente *NoEquipped*, Ilustración 28. En cada uno de ellos se realizan las operaciones propias de un nodo con cada tipo de rol. En cuanto a las asociaciones, es importante subrayar la creada con "ITomarMedidas" puesto que se corresponde con el componente *TomarMedidas*, del cual se hace uso pero que, como se expuso en la sección 3.6.2, queda fuera del ámbito de este proyecto.

Las operaciones incluidas son:

- Rol Requester

  En el componente *Requester*, a diferencia del resto, hay que señalar la existencia de la clase "ComprobarRecepcionTestigo", la cual es responsable de la creación de un hilo de ejecución para la verificación de la existencia de testimonios recibidos, así como la posterior creación y envío de la evidencia asociada. Esta decisión se basa en el beneficio de recibir más de un testimonio antes de la creación de una evidencia, puesto que a mayor número de testimonios mayor es la probabilidad de construir una evidencia fiable. Por tanto, el único modo de proporcionar tiempo para la formación de una evidencia, aún dependiendo esto de muchos factores, es creando un hilo de ejecución que cada cierto tiempo, *TIEMPO_MAX_REENVIOS*, verifique si se han recibido testimonios para, de lo contrario, reenviar la solicitud hasta un máximo de veces, *NUMERO_MAX_REENVIOS*. Con esta clase se asocian las siguientes funciones:

- Función “crearComprobacionRecepcionTestigo”: creación del hilo que verifica los testimonios recibidos.
- Función “comprobar”: asociada al hilo y encargada de la espera de un tiempo oportuno, *TIEMPO_MAX_REENVIOS*, para posteriormente invocar a “comprobarRespuestas” de la clase “Requester”.
- Función “finHiloComprobacionRespuestas”: finalización de la ejecución del hilo, invocada una vez que se haya finalizado el procesamiento de una evidencia, de modo que éste no permanezca activo indefinidamente.

Por otra parte, las funciones asociadas al rol Requester, clase “Requester” son:

- Función “arrancarRequester”: inicialización y carga de las variables y elementos necesarios para el comienzo de un nodo que desempeñe el rol Requester. Además, se realiza la creación del cliente emisor de los mensajes por medio de la función “cliente” de “ICar”.
- Función “inicializarComprobacionRespuestas”: arranque del hilo de comprobación de la recepción de testimonios indicado en párrafos anteriores.
- Función “filtrarVelocidad”: obtención de la velocidad a incluir en la evidencia por medio de los testimonios recibidos. Esto se efectúa mediante la intersección de los intervalos enviados en los distintos testimonios y el promedio del intervalo resultante.
- Función “comprobarRespuestas”: creación de la evidencia, mediante “crearEvidencia” de “ICar”, y envío de la misma a la RSU más cercana. Esta función se realizará cada cierto tiempo, *TIEMPO_MAX_REENVIOS*, por medio del hilo mencionado anteriormente.
- Función “enviarPeticion”: recepción de notificaciones de sanción para realizar el posterior envío de la solicitud correspondiente.

  Previamente a la realización del envío se comprueba si dicha solicitud no se ha enviado, con anterioridad, un número superior al establecido en *NUMERO_MAX_REENVIOS*, lo cual se realiza por medio de la función “comprobarPeticion” de “ICar”.

  Del mismo modo, el TTL incluido en el mensaje de solicitud ha de ser inicializado con el *MAX_TTL*.

  Para concluir, una vez creada la solicitud se envía a los nodos Witness considerados cercanos, utilizando las funciones “crearPeticion” y “comprobarCercania” de “ICar”.
- Función “recibirPeticion”: arranque del hilo correspondiente con la recepción de testimonios procedentes de nodos con rol Witness, NoEquipped o RSU. Este hilo será detallado en el componente *Comunicacion*.

- Función “recibidasPeticion”: recepción de los testimonios correspondientes con las solicitudes enviadas.

  Primeramente se comprueba la corrección de cada uno de los mensajes recibidos, lo cual se relaciona con la utilización de operaciones criptográficas realizadas a través de la función “verificarRespuesta” de “ICar”.

  Además, se verifica tanto que la marca de tiempo no sobrepasa *TIEMPO_MAX_RESPUESTA_REQUESTER* como que el TTL es superior a cero.

- Rol Witness, clase “Witness”.
  - Función “arrancarWitness”: inicialización y carga de las variables y elementos necesarios para el comienzo de un nodo que desempeñe el rol Witness. Además, se realiza la creación del cliente emisor de los mensajes por medio de la función “cliente” de “ICar”.
  - Función “recibirPeticionesdeR”: arranque del hilo correspondiente con la recepción de solicitudes provenientes de un nodo con rol Requester.
  - Función “recibidasPeticion”: recepción y procesamiento de las solicitudes realizadas por un nodo con rol Requester y envío del testimonio correspondiente.

    Concretamente, tras la recepción de un mensaje, se garantiza que la solicitud es correcta realizando las operaciones criptográficas adecuadas mediante la función “comprobarPeticion” de “ICar”, así como se verifica que la marca de tiempo no sobrepasa la cantidad establecida en *TIEMPO_MAX_RESPUESTA_WITNESS*.

    Del mismo modo, se comprueba la validez del certificado del nodo emisor de la solicitud, puesto que de estar revocado, el mensaje sería descartado. Esto se realiza a través de la función “comprobarPertenenciaCRL” de “ICar”.

    Por último, se envía el testimonio al Requester o, en caso de no encontrarse cercano, a otro u otros nodos con rol NoEquipped considerados cercanos o, de no ser posible, a nodos con rol RSU. Este proceso se efectúa utilizando las funciones “crearRespuesta” y “comprobarCercania” de “ICar”.

    No hay que olvidar que, tras el envío de un mensaje, es necesaria la comprobación de la existencia de mensajes en la lista de recibidos mientras uno de ellos estaba siendo procesado, lo cual se efectúa por medio de la función “comprobarListaMensajesAlmacenados” de “ICar”. Por el mismo motivo, si se recibe un mensaje que no puede procesarse debido a otro cuya ejecución está en proceso, debe almacenarse. Por ejemplo, esto puede suceder en el momento en el que un nodo con rol Witness esté creando un testimonio y, en ese instante, recibe una nueva solicitud.

- Función “establecerRangoVelocidad”: creación del intervalo de velocidad asociado al emisor de la solicitud recibida. Dicho intervalo debe establecerse según la ecuación

  $(velocidadEmisor \mp (velocidadEmidor \times numIntervalo))$, siendo numIntervalo un número comprendido en el intervalo [0,1;0,4].

- Rol NoEquipped, clase “NoEquipped”:
  - Función “arrancarNoEquipped”: inicialización y carga de las variables y elementos necesarios para el comienzo de un nodo que desempeñe el rol NoEquipped. Además, se realiza la creación del cliente emisor de los mensajes por medio de la función “cliente” de “ICar”.
  - Función “peticionesDeW”: arranque del hilo correspondiente con la recepción de testimonios procedentes de un nodo con rol Witness.
  - Función “peticionesEntreNoEquipped”: arranque del hilo correspondiente con la recepción de testimonios procedentes de un nodo con rol NoEquipped o RSU.
  - Función “recibidasPeticion”: recepción, creación y envío del testimonio pertinente.

    Tras recibir un testimonio, un primer paso es considerar si el mensaje no ha sido recibido un número superior a *NUMERO_MAX_REENVIOS*, descartándolo en caso contrario. Esta funcionalidad se realiza a través de la función “comprobarPeticion” de “ICar”.

    Asimismo, se verifica si la marca de tiempo sobrepasa *TIEMPO_MAX_RESPUESTA_NOEQUIPPED* y se comprueba que el TTL es superior a cero.

    Por otra parte, se crea el testimonio a enviar.

    Finalmente, se realiza el envío del testimonio al nodo correspondiente y se comprueba la existencia de nuevos mensajes para su posterior procesamiento, tal y como queda descrito anteriormente en la función “recibidasPeticion” de la clase “Witness”.

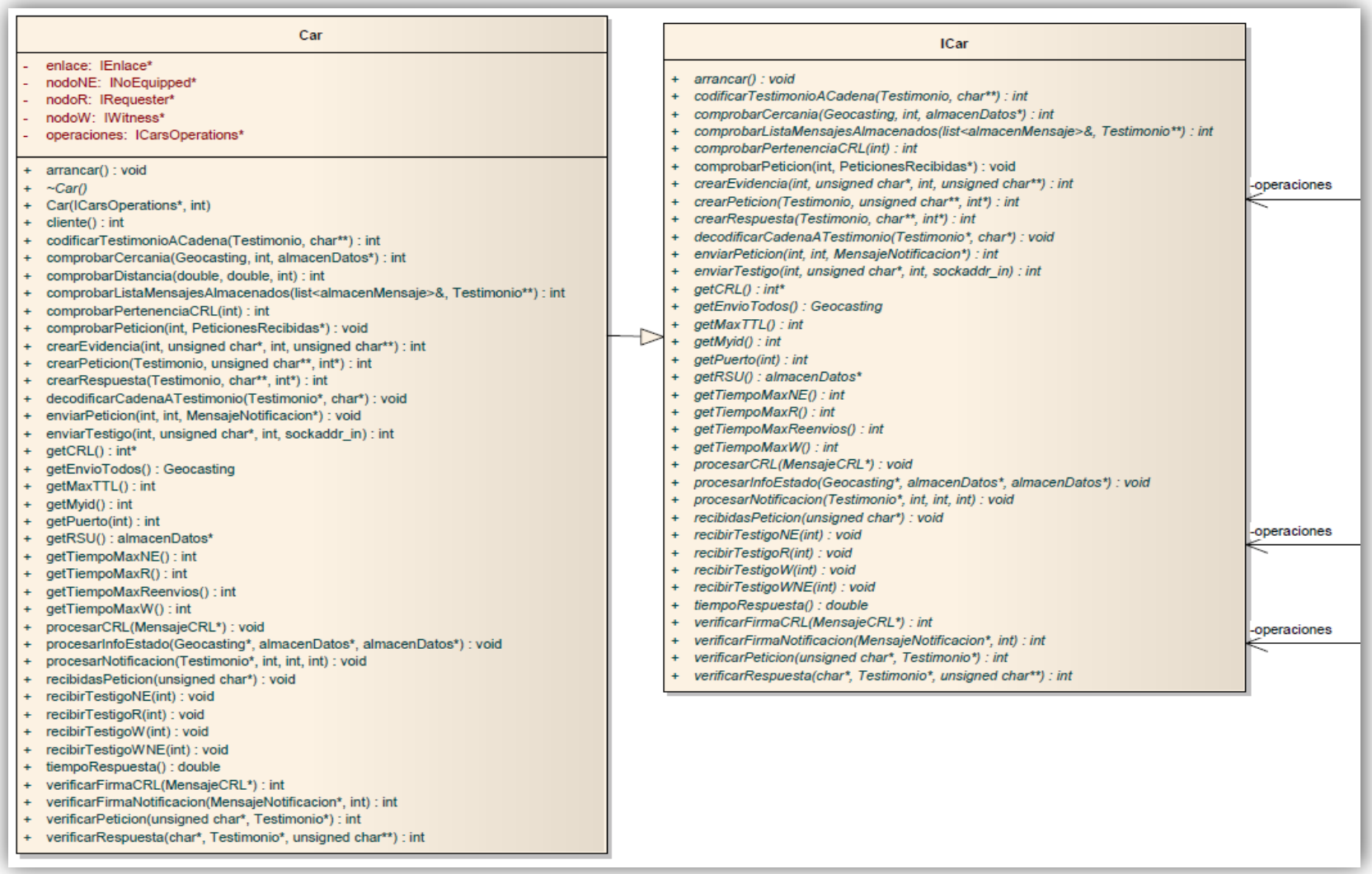


Ilustración 27: diseño componente Nodos.

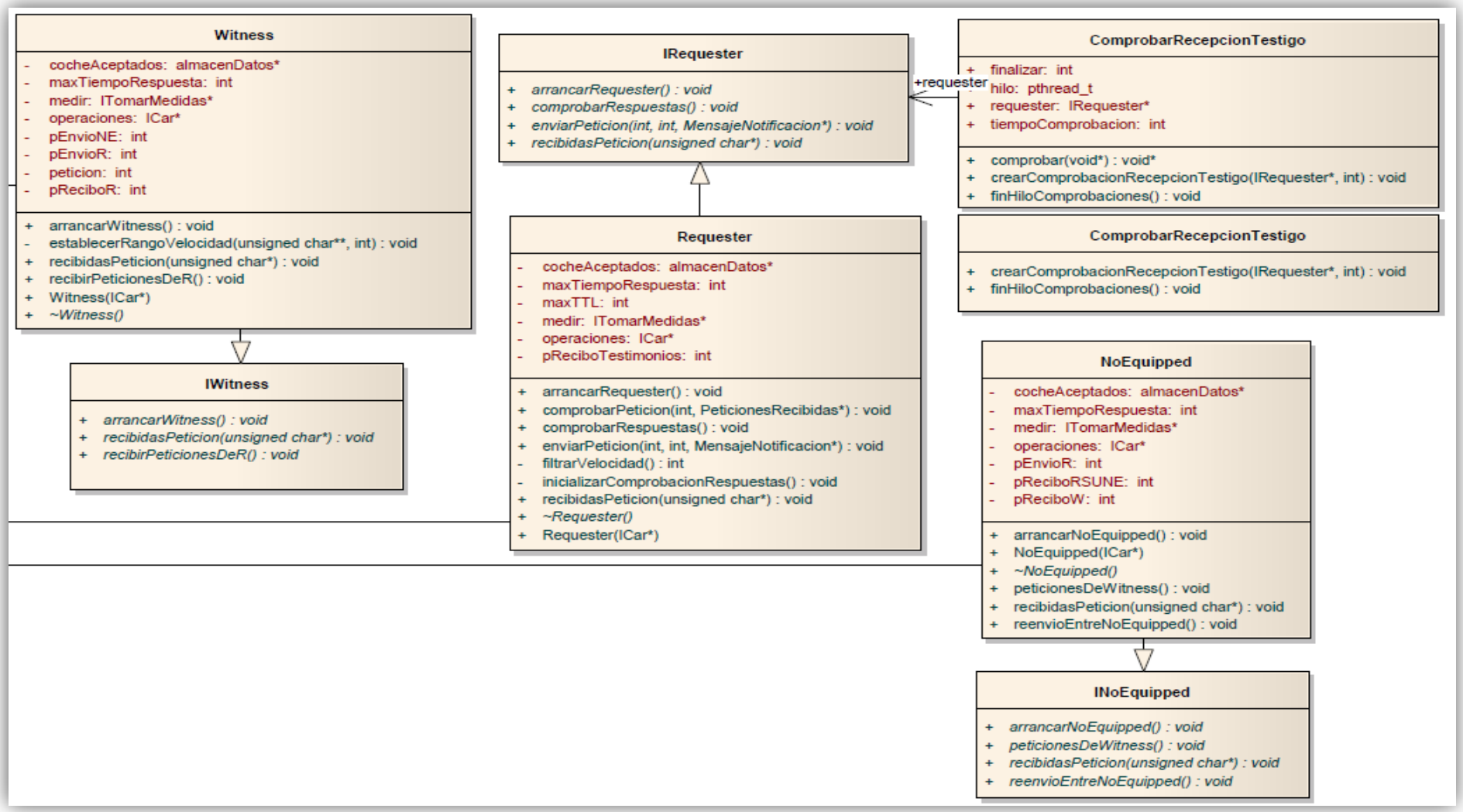


Ilustración 28: diseño componentes Requester, Witness y NoEquipped.

4.1.2.1.3 Diseño del componente Movimiento

El componente *Movimiento,* Ilustración 29, se caracteriza por ser el utilizado para la creación del movimiento de los nodos, siendo éste el lugar en el que se produce una interacción entre funciones propias del simulador NCTUns 5.0 (1) y funciones creadas para el desarrollo del protocolo.

Por una parte, en la clase "Movimiento" se efectúan las operaciones necesarias para proporcionar movilidad a cada nodo, lo cual se consigue tras realizar el estudio de la situación actual, presentado en la sección 3.2, y la integración de funciones previamente desarrolladas en el simulador NCTUns 5.0 (1), correspondientes con "DetermineAcceleration", "DetermineVc", "FulTheQueue", "RandomDirection", "RandomTheNextTriggerPointOrNot", "SelectMaximun" y "SelectMinimun".

Teniendo en cuenta lo comentado, en la clase "Movimiento" sólo se han de crear tres funciones, dos de ellas encargadas de proporcionar las coordenadas de la pantalla en las que se sitúan cada uno de los nodos y la tercera, responsable del movimiento:

- Función "getCoorX": obtención de la coordenada X en la que se encuentra un determinado nodo en la pantalla del simulador NCTUns 5.0 (1).
- Función "getCoorY": obtención de la coordenada Y en la que se encuentra un determinado nodo en la pantalla del simulador NCTUns 5.0 (1).
- Función "mover": creación de movimiento mediante la integración de las funciones desarrolladas previamente en el simulador NCTUns 5.0 (1).

Por otra parte, en la clase "EfectuarMovimiento" es donde se realiza la petición de movimiento así como donde se crea y ejecuta el hilo que realiza esta funcionalidad. De lo contrario, un mismo hilo debería realizar simultáneamente el movimiento y las funciones correspondientes con su rol. Asimismo, esta clase es utilizada para conocer la posición de cada nodo en la pantalla de simulación, por lo que cada vez que un nodo se desplaza, se comunica dicho desplazamiento a la clase "ICarsOperations", componente *Operaciones*. Considerando lo comentado, se puede tener constancia de la localización de cada nodo para enviarla en los mensajes de *beaconing*. Concretamente las operaciones asociadas a esta clase son:

- Función "crearMovimiento": creación del hilo que inicia el movimiento de cada nodo.
- Función "moverH": asociada al hilo y encargada de notificar las coordenadas a las que se ha desplazado un nodo, a través de la función "obtenerCoordenas" de la clase "ICarsOperacions", tras la recepción periódica de cada una de ellas.

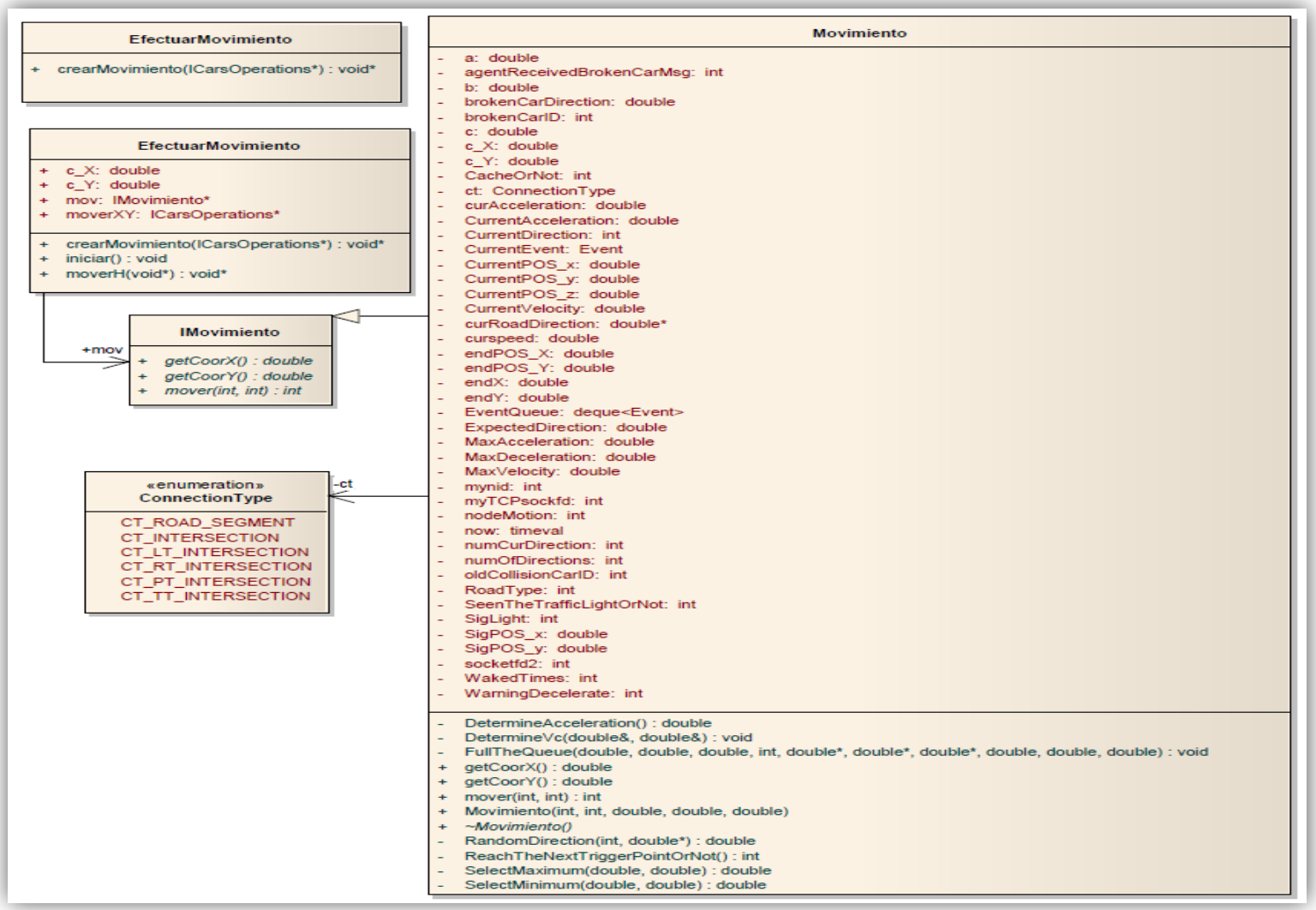


Ilustración 29: diseño componente Movimiento.

4.1.2.1.4 Diseño del componente Operaciones

Otro de los componentes es *Operaciones*, Ilustración 30, el cual está asociado con *Nodos*, anteriormente presentado, *IConfiguracion* e *IEnlace*. Este componente comprende las operaciones comunes a todos los roles, situadas en "ICarsOperations".

Además, es importarte subrayar la asociación creada con "ITomarMedidas" puesto que se corresponde con el componente *TomarMedidas*, del cual se hace uso pero que, como se expuso en la sección 3.6.2, queda fuera del ámbito de este proyecto.

Las funcionalidades asociadas a este componente son:

- Lectura de la configuración y establecimiento del movimiento:
    - Función "comenzar": responsable del arranque de la creación del movimiento de cada uno de los nodos, continuado en el componente *Movimiento* presentado anteriormente. Asimismo, en esta misma función se efectúa la lectura de los puertos de comunicación, de modo que muchos de los puertos sean proporcionados a la clase "ICar" para que los envíos de mensajes entre nodos con distintos roles se lleven a cabo satisfactoriamente. Además, se realiza la creación del cliente emisor de los mensajes por medio de "cliente" de "IEnlace".
- Arranque de los hilos correspondientes tanto con la recepción de notificaciones, de CRL y de información de estado de una RSU como con la recepción de la información de estado de nodos con rol Requester, Witness y NoEquipped :
    - Función "recibirGeoRSU": arranque del hilo correspondiente con la recepción de mensajes de información de estado de un RSU.
    - Función "recibirMensajeRSU": arranque del hilo correspondiente con la recepción de mensajes de notificación y CRL procedentes de una RSU.
    - Función "recepcionGeocastingTodos": arranque del hilo correspondiente con la recepción de mensajes de *beaconing*, responsables del almacén de la información de estado de nodos con rol Requester, Witness o NoEquipped.
- Recepción y envío de la información de estado:
    - Función "envioGeocastingTodos": envío de la información de estado, identificador, velocidad, tipo de vehículo/rol y posición, en cada uno de los mensajes de *beaconing*, a todos los nodos participantes en la simulación.

      Teniendo en cuenta que los nodos son móviles, los mensajes de *beaconing* se han de enviar cada segundo, de modo que la información

de estado de cada nodo permanezca actualizada. Además, es fundamental considerar, tal y como se señaló en el estudio del simulador (sección 3.2), que la dirección de *broadcast* se corresponde con la 1.0.1.255.

- Función "reciboGeocastingTodos": recepción de la información de estado presente en cada uno de los mensajes de *beaconing*, procedente de los nodos participantes en la simulación cuyo rol sea Requester, Witness o NoEquipped.

- Función "recibidasGeoRSU": del mismo modo que en la operación anterior, en esta función se recibe la información de estado, posición e identificador, asociada a los nodos con rol RSU.

  Posteriormente, tras recibir cualquiera de los datos mencionados, si el nodo receptor se encuentra del emisor lo suficientemente próximo como para considerarlo vecino, se han de almacenar dichos datos. De este modo, posteriormente se puede tener constancia de los considerados cercanos, enviando únicamente datos a aquellos que se mantengan dentro de una distancia máxima. Considerando lo mencionado, es gracias a este procedimiento por lo que se tiene constancia de la información de vecindad de un determinado nodo, la cual habrá que enviar al sistema de visión de mensajes de la simulación. Esto se realiza haciendo uso de "anyadirCocheAceptado" y "comprobarCercania".

- Recepción de mensajes de notificación y de mensajes con la lista de identificadores de nodos con certificado revocado:

  - Función "recibidosMensajeRSU": recepción tanto de los mensajes de notificación de sanción como de la CRL. Dependiendo del tipo de mensaje recibido se invocará "rellenarNotificacion" o "rellenarCRL".

  - Función "rellenarNotificacion": procesamiento de la notificación recibida, así como la llamada adecuada para que se produzca la activación del comienzo del protocolo y el nodo oportuno inicie las funciones asociadas a la ejecución del rol Requester.

    Asimismo, es imprescindible la verificación de la corrección de la firma adjunta al mensaje, a través de la función "verificarFirmaNotificacion" de "IEnlace", lo cual queda indicado en la descripción de los mensajes, sección 3.1.1.

  - Función "rellenarCRL": procesamiento de la CRL con el fin de poder consultar la existencia de un determinado identificador de nodo y conocer la validez de su certificado.

Además, es imprescindible la verificación de la corrección de la firma adjunta al mensaje, mediante la función "verificarFirmaCRL" de "IEnlace", lo cual queda indicado en la descripción de los mensajes, sección 3.1.1.

- Comprobaciones necesarias para el correcto funcionamiento del protocolo, las cuales satisfacen muchos de los requisitos no funcionales, presentados en la sección 3.9.4, y relacionados con el control de errores y el rendimiento del sistema:

  - Función "comprobarPertenenciaCRL": comprobar la pertenencia de un determinado identificador en la lista de identificadores de nodos cuyo certificado está revocado.

  - Función "comprobarListaMensajesAlmacenados": verificación de la existencia de un mensaje en la cola de mensajes que han de ser procesados. Considerando que en nodos con rol Witness o NoEquipped se pueden recibir mensajes mientras uno de ellos se está procesando, se han de almacenar los recibidos en ese momento, pudiéndose obtener posteriormente para ser procesados siempre que la marca de tiempo no sobrepase *TIEMPO_DESCARTE_ALMACENADOS*.

  - Función "comprobarCercania": efectuar las operaciones necesarias para indicar si un nodo se encuentra cercano a otro dado. Concretamente, con esta función se realiza la implementación del intercambio de mensajes, de modo que se toma como base la posición de cada uno de los nodos, emisor y receptor, y se determinada, de acuerdo a los parámetros *DISTANCIA_ACEPTADA_REQUEESTER*, *DISTANCIA_ACEPTADA_WITNESS* y *DISTANCIA_ACEPTADA_NOEQUIPPED*, la aceptación o el rechazo del procesamiento del mensaje recibido o el mensaje a enviar.

    Es una de las funciones principales para el funcionamiento del protocolo y se analizará en la sección de implementación (sección 5.1), específicamente en la parte correspondiente con "Implementación del intercambio de mensajes".

  - Función "anyadirCocheAceptado": tras la recepción de un mensaje de *beaconing*, si el nodo se encuentra cercano, la información es almacenada. De este modo, se almacena información de un nodo considerado vecino, al que posteriormente se pueden realizar envíos de los mensajes asociados al protocolo EVIGEN.

  - Función "obtenerCoordenadas": almacenamiento de las coordenadas a las que un nodo se ha desplazado. Esta función en invocada desde el hilo

creador de movimiento, de modo que se actualicen las posiciones del nodo y se envíe la posición correcta en cada uno de los mensajes de *beaconing*.

- Función "comprobarPeticion": verificación de la recepción previa de una solicitud, es decir, se ha de poder comprobar si una solicitud ha sido reenviada, o no, un número superior a *NUMERO_MAX_REENVIOS*, de modo que se descarte el reenvío en caso contrario.

Ilustración 30: diseño componente Operaciones

4.1.2.1.5 Diseño del componente CriptografiaNodos

Teniendo en cuenta la necesidad de hacer uso de la seguridad para el envío de mensajes, se ha desarrollado el componente *CriptografiaNodos*, clases "OpeCriptograficas" e "IOpeCriptograficas", Ilustración 31, en el que se engloban las funciones correspondientes con la creación de mensajes en los que se requieren operaciones criptográficas, tal y como queda descrito al comienzo de la sección de análisis (sección 3.1.1).

Además, es importarte subrayar la asociación creada con "ICriptografia" puesto que se corresponde con el componente *Criptografia*, del cual se hace uso pero que, como se expuso en la sección 3.6.2, queda fuera del ámbito de este proyecto.

Las funciones que comprendidas en este componente son:

- Función "crearPeticion": creación del mensaje de solicitud que es enviado desde un nodo con rol Requester a un Witness. Este mensaje es firmado así como adjuntado el certificado correspondiente.
- Función "verificarPeticion": comprobación de la validez de la firma incluida en el mensaje y de la corrección del certificado adjunto.
- Función "crearRespuesta": creación del mensaje que es enviado a un nodo con rol Requester, correspondiente con la respuesta a una solicitud. En él se incluye un ticket firmado que contiene el intervalo de velocidad cifrado por una clave.
- Función "verificarRespuesta": comprobación de la validez de la firma incluida en el mensaje y de las operaciones de cifrado realizadas.
- Función "crearEvidencia": creación del mensaje a enviar para poder revocar una sanción recibida. Este mensaje se firma y se adjuntan todos los tickets recibidos en los distintos testimonios.
- Función "verificarFirmaNotificacion": comprobación de la corrección de la firma adjunta en un mensaje de notificación de sanción.
- Función "verificarFirmaCRL": comprobación de la corrección de la firma adjunta en un mensaje de CRL.

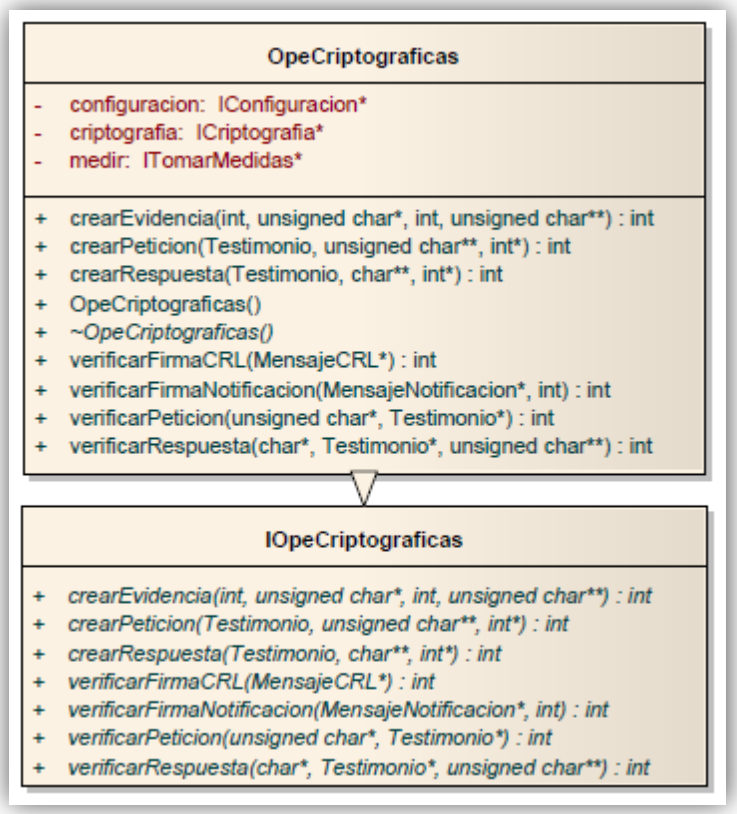


Ilustración 31: diseño componente CriptografiaNodos.

4.1.2.1.6 Diseño del componente Comunicación

Uno de los componentes más importantes es el de *Comunicacion*, Ilustración 32 e Ilustración 33, puesto que en él se realizan cada una de las comunicaciones necesarias para la correcta ejecución del protocolo, las cuales se corresponden con la recepción de notificaciones de sanción, el envío de evidencias, el envío de solicitudes y el envío y recepción de testimonios.

Por un lado, las clases "IComunicacionInterfaz" y "ComunicacionInterfaz" se corresponden con la creación del cliente y las llamadas a las funciones adecuadas para el envío de información hacia el sistema de visión de mensajes de la simulación.

Por otro lado, se encuentra la clase "MsgOperations", en la que se implementan las operaciones de recepción y envío de datos, de modo que se puedan modificar los mecanismos de envío y recepción en el momento que se desee. Las operaciones incluidas son:

- Función "enviarMensaje": envío de cualquier tipo de mensaje cuyo formato sea una cadena.

- Función "recibirTestigo": recepción de cualquier tipo de mensaje cuyo formato sea una cadena, correspondiéndose, en este caso, con la recepción de testimonios.

- Función “recibirMensajeGeoRSU”: recepción de mensajes de *beaconing* con la información de estado de una RSU. Ésta es la función invocada por nodos con rol RSU, anexos a este proyecto, para enviar su información de estado, identificador y posición.

- Función “recibirMensajeRSU”: recepción de mensajes de notificación y CRL procedentes de una RSU. Del mismo modo que la función anterior, es invocada por nodos con rol RSU, anexos a este proyecto, para enviar las notificaciones y CRL adecuadas.

- Función “enviarPropiedades”: envío de mensajes correspondientes con la información de estado de un nodo con rol Requester, Witness o NoEquipped.

- Función “recibirPropiedades”: recepción de la información de estado procedente de un nodo con rol Requester, Witness o NoEquipped.

Por otra parte, en la clase “Sockets” se centralizan las operaciones de creación y destrucción de clientes y servidores creados mediante sockets, de esta forma, la funcionalidad queda desacoplada y puede ser modificada en el momento en el que se requiera, estableciendo otro mecanismo de comunicación. Las funciones de esta clase son las siguientes:

- Función “cliente”: creación de un cliente.

- Función “servidor”: creación de un servidor.

- Función “cerrar”: cierre de la conexión de acuerdo con un determinado descriptor.

Otro gran bloque lo comprenden las clases “Comunicacion” y su interfaz “IComunicacion”, siendo en ellas donde se efectúan las llamadas de recepción y envío de datos, actúan a modo de fachada para que otros componentes hagan uso de ellas.

Por otra parte, es imprescindible indicar que la realización de la implementación de las recepciones de datos, en cada uno de los roles, se ha efectuado a través de la creación de hilos independientes para que la ejecución no quede suspendida hasta la recepción de un dato, sino que se puedan realizar tareas paralelamente. La base de la decisión tomada asociada a la creación de dichos hilos, es la de mejorar la sobrecarga del sistema, satisfaciendo, de este modo, requisitos no funcionales descriptos en la sección 3.9.4, los cuales están relacionados con el rendimiento del sistema. Los distintos hilos y la información a recibir en cada uno de ellos se corresponden con:

- Hilo de recepción de la información de estado, enviada en los mensajes de *beaconing*, procedente de una RSU, clase “RecepcionRSU”.

- Hilo de recepción de la información de estado, enviada en los mensajes de *beaconing*, procedente de nodos con rol Requester, Witness y NoEquipped, clase "RecepcionPropiedades".

- Hilo de recepción de mensajes enviados desde un nodo Requester a un nodo Witness, clase "RecepcionTestimonioW".

- Hilo de recepción de mensajes enviados desde un nodo Witness a un nodo NoEquipped, clase "RecepcionTestimonioWNE".

- Hilo de recepción de mensajes enviados desde un nodo NoEquipped o RSU a un nodo con rol NoEquipped, clase "RecepcionTestimonioNE".

- Hilo de recepción de mensajes enviados desde un nodo Witness, NoEquipped o RSU a un nodo con rol Requester, clase "RecepcionTestimonioR".

  En todas las clases mencionadas se presentan dos funciones las cuales, descriptas genéricamente, se corresponden con:

  - Función "crearX": creación del hilo receptor de cada uno de los mensajes.

  - Función "recepcionX": asociada al hilo y responsable tanto de la creación del servidor para la recepción de los mensajes, como de la llamada a la función oportuna de clase "Comunicaciones", tras la recepción de cada uno de los mensajes. Todo esto se realiza haciendo uso de "servidor" de "IComunicacion" y de las funciones "testigoRecibidoWNE", "testigoRecibidoW", "testigoRecibidoNE", "testigoRecibidoR", "propiedadesRecibidas", "geoRecibidoRSU", y "mensajeRecibidoRSU" de "Comunicaciones".

Finalmente se presenta la clase "Comunicaciones", a la que se accede para realizar el arranque de los hilos mencionados anteriormente, junto con la recepción de datos procedentes de los mismos y la entrega de dichos datos a las clases adecuadas. Las operaciones situadas en esta clase son las siguientes:

- Función "recibirTestigoR": realización de la llamada a "crearRecepcionTestimonio" para arrancar el hilo correspondiente a la recepción de mensajes en un nodo con rol Requester.

- Función "recibirTestigoWNE": realización de la llamada a "crearRecepcionTestimonioWNE" para arrancar el hilo correspondiente a la recepción de mensajes en un nodo con rol NoEquipped, procedentes de un nodo con rol Witness.

- Función "recibirTestigoNE": realización de la llamada a "crearRecepcionTestimonioNE" para arrancar el hilo correspondiente a la recepción de mensajes en un nodo con rol NoEquipped, procedentes de un nodo con rol NoEquipped o RSU.

- Función "recibirTestigoW": realización de la llamada a "crearRecepcionTestimonioW" para arrancar el hilo correspondiente a la recepción de mensajes en un nodo con rol Witness.

- Función "recibirRSU": realización de la llamada a "crearRecepcionRSU" para arrancar el hilo correspondiente a la recepción de los mensajes de *beaconing* procedentes de un nodo con rol RSU.

- Función "recibirMensajeRSU": realización de la llamada a "crearRecepcionMensajeRSU" para arrancar el hilo correspondiente a la recepción de los mensajes de notificación y CRL procedentes de un nodo con rol RSU.

- Función "recibirPropiedades": realización de la llamada a "crearRecepcionPropiedades" para arrancar el hilo correspondiente a la recepción de mensajes de *beaconing* procedentes de nodos con rol Requester, Witness o NoEquipped.

- Función "enviarTestigo": realización de la llamada a "enviarMensaje" de la clase "MsgOperations".

- Función "enviarPropiedades": realización de la llamada a "enviarPropiedades" de la clase "MsgOperations".

- Función "propiedadesRecibidas": tras recibir un mensaje de *beaconing* se prepara la posterior recepción, por medio de la llamada a "recibirPropiedades" de "MsgOperations". Además, se notifica la recepción mediante "recibidasPeticion" de "ICar".

- Función "testigoRecibidoW": tras recibir un mensaje de solicitud en un nodo con rol Witness, el servidor se prepara para una recepción posterior por medio de la llamada a "recibirTestigo" de "MsgOperations". Además, se notifica la recepción mediante la función "recibidasPeticion" de "ICar".

- Función "testigoRecibidoNE": tras recibir un testimonio, desde un nodo con rol NoEquipped o RSU en un nodo con rol NoEquipped, el servidor se prepara para una recepción posterior por medio de la llamada a "recibirTestigo" de "MsgOperations". Además, se notifica la recepción mediante la función "recibidasPeticion" de "ICar".

- Función “testigoRecibidoWNE”: tras recibir un testimonio, desde un nodo con rol Witness en uno con rol NoEquipped, el servidor se prepara para una recepción posterior por medio de la llamada a “recibirTestigo” de “MsgOperations”. Además, se notifica la recepción mediante la función “recibidasPeticion” de “ICar”.

- Función “testigoRecibidoR”: tras recibir un testimonio, en un nodo con rol Requester, el servidor se prepara para una recepción posterior por medio de la llamada a “recibirTestigo” de “MsgOperations”. Además, se notifica la recepción mediante la función “recibidasPeticion” de “ICar”.

- Función “geoRecibidoRSU”: tras recibir un mensaje de *beaconing* de una RSU el servidor se prepara para una recepción posterior por medio de la llamada a “recibirMensajeGeoRSU” de “MsgOperations”. Además, se notifica la recepción mediante la función “recibidasPeticionGeoRSU” de “ICarsOperations”.

- Función “mensajeRecibidoRSU”: tras recibir un mensaje notificación o CRL el servidor se prepara para una recepción posterior, por medio de la llamada a “recibirMensajeRSU” de “MsgOperations”. Además, se notifica la recepción mediante la función “recibidosMensajeRSU” de “ICarsOperations”.

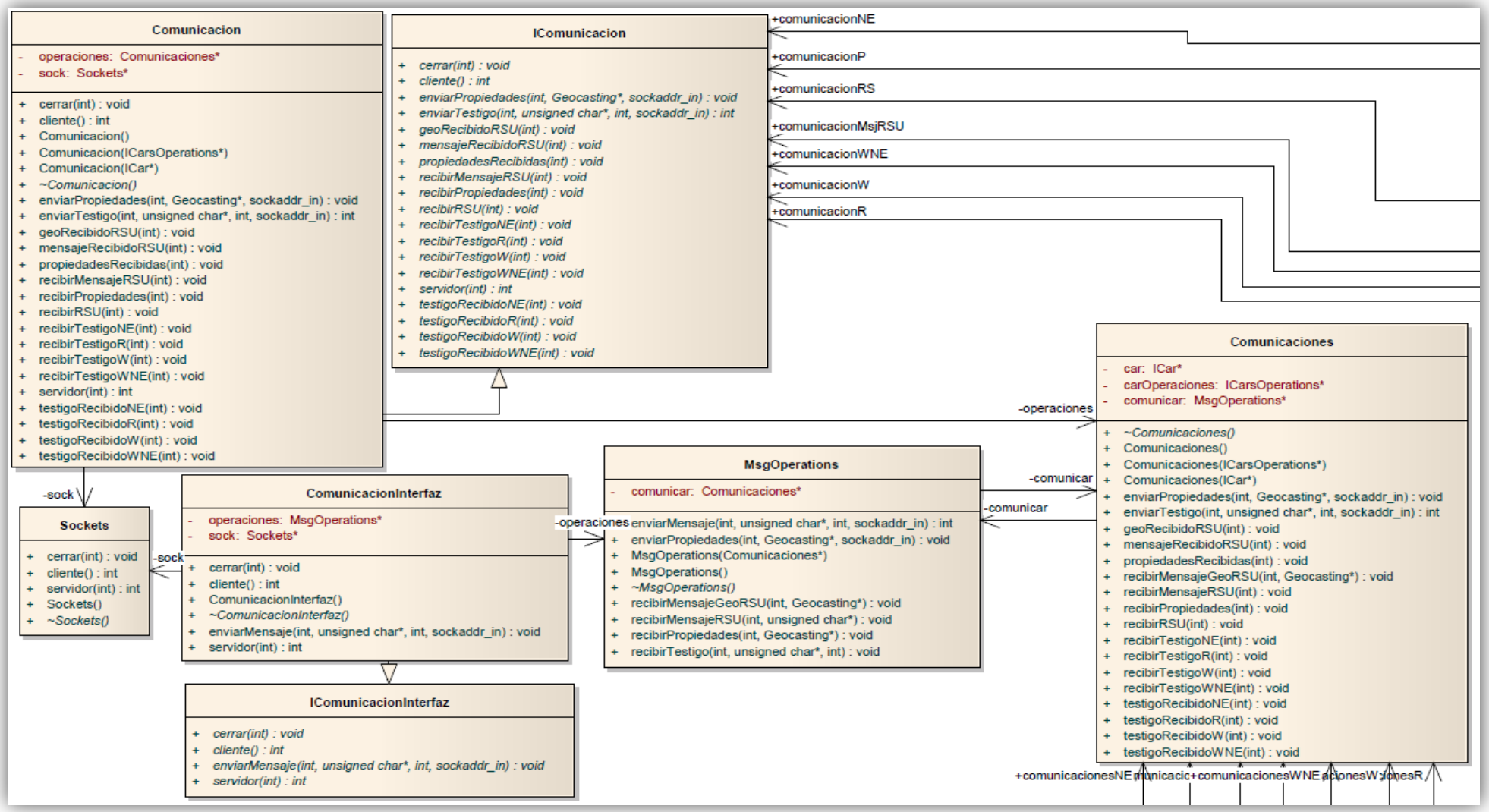


Ilustración 32: diseño componente Comunicacion 1.

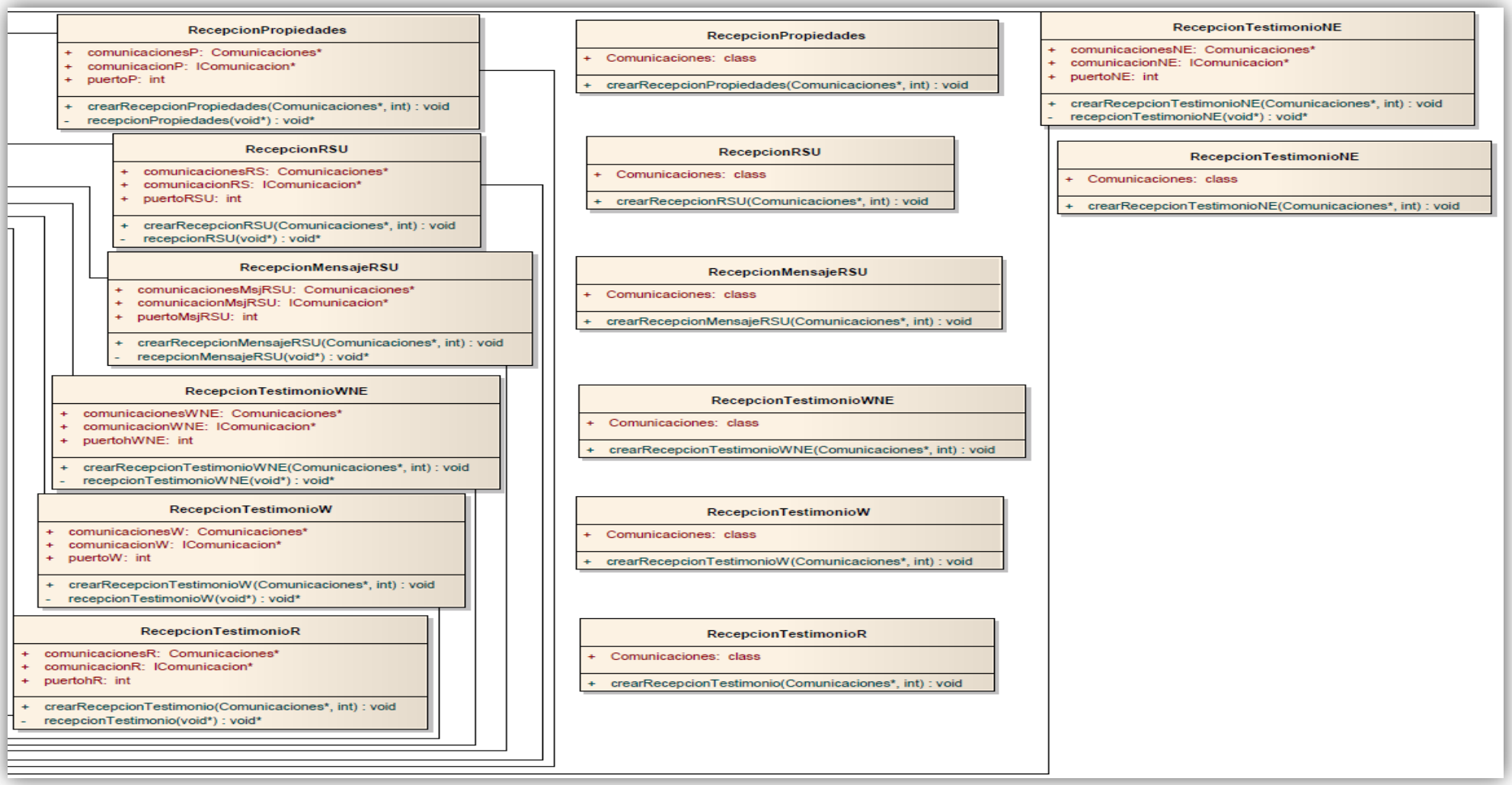


Ilustración 33: diseño componente Comunicacion 2.

4.1.2.1.7 Diseño del componente ComunicacionInterfaz

Una de las funciones necesarias, para la creación del sistema de visión de mensajes de la simulación, es el envío de los mensajes intercambiados en cada simulación, siendo esto solventado en este componente. Se pueden describir tres grandes funcionalidades.

Para comenzar, es imprescindible indicar que la creación de cada uno de los mensajes que han de ser enviados al sistema de visión de mensajes de la simulación, puede realizarse conforme a los criterios establecidos por el programador, pero teniendo en cuenta los requisitos funcionales presentados en la sección 3.9.2.1 y todo lo comentado en el componente *Vista* descrito anteriormente, sección 4.1.1.1.

La principal función es la creación de los mensajes que han de ser enviados, lo cual se realiza en la clase "ProcesarMsgInterfaz", cuya interfaz es "IProcesarMsgInterfaz". Las operaciones que realizan esta funcionalidad son:

- Función "procesarMensajeCRL": creación del mensaje de CRL correspondiente con una lista de identificadores de nodos.
- Función "procesarInfoEstado": creación del mensaje de *beaconing* correspondiente con la información de estado de cada uno de los nodos con rol Requester, Witness y NoEquipped.
- Función "procesarInfoEstadoRSU": creación del mensaje de *beaconing* correspondiente con la información de estado de cada uno de los nodos con rol RSU, la cual procederá de la parte "ProtocoloEVIGEN EntornoInfraestructura" comentado en la arquitectura definitiva, sección 3.6.2.
- Función "procesarNotificación": creación de todos los mensajes intercambiados entre los nodos del protocolo, a excepción de los de CRL y *beaconing*.

Otra de las funciones a realizar es el envío de los mensajes, ejecutado en "EnviarInterfaz" y cuyas funciones son:

- Función "enviarInfoEstado": envío de los mensajes de *beaconing* por medio del componente *Comunicacion,* al sistema de visión de mensajes de la simulación.
- Función "enviarNotificación": envío de los mensajes intercambiados en el protocolo, excepto los de *beaconing*, por medio del componente *Comunicacion*, al sistema de visión de mensajes de la simulación.

Y finalmente, la última función es la de proporcionar una interfaz para que el componente "*ProtocoloEVIGEN_p2*", anexo a este proyecto y mencionado en la arquitectura definitiva (sección 3.6.2), pueda realizar las invocaciones convenientes para que en el sistema de visión de mensajes de la simulación también se puedan presentar los mensajes intercambiados entre los nodos con rol RSU y los nodos con rol Requester,

Witness o NoEquipped. Todo lo mencionado se realiza en “EnlaceComponente”, cuya interfaz es “IEnlaceComponente”, y contiene las siguientes funciones:

- Función “recibidaEvidencia”: formación del mensaje asociado a la recepción de una evidencia en un nodo con rol RSU, invocando posteriormente “procesarNotificación” de “IProcesarMsgInterfaz”.

- Función “enviadaNotificacion”: formación del mensaje asociado al envío de una notificación de sanción desde un nodo con rol RSU, invocando posteriormente “procesarNotificación” de “IProcesarMsgInterfaz”.

- Función “recibidoMensajeNE_RSU”: formación del mensaje asociado a la recepción de un testimonio en un nodo con rol RSU, invocando posteriormente a “procesarNotificación” de “IProcesarMsgInterfaz”.

- Función “enviadoMensajeNE_RSU”: formación del mensaje asociado al envío de un testimonio desde un nodo con rol RSU, invocando posteriormente “procesarNotificación” de “IProcesarMsgInterfaz”.

- Función “procesarInfoEstadoRSU”: formación del mensaje de *beaconing*, con la información de estado, enviado desde una RSU e invocando posteriormente “procesarInfoEstadoRSU” de “IProcesarMsgInterfaz”.

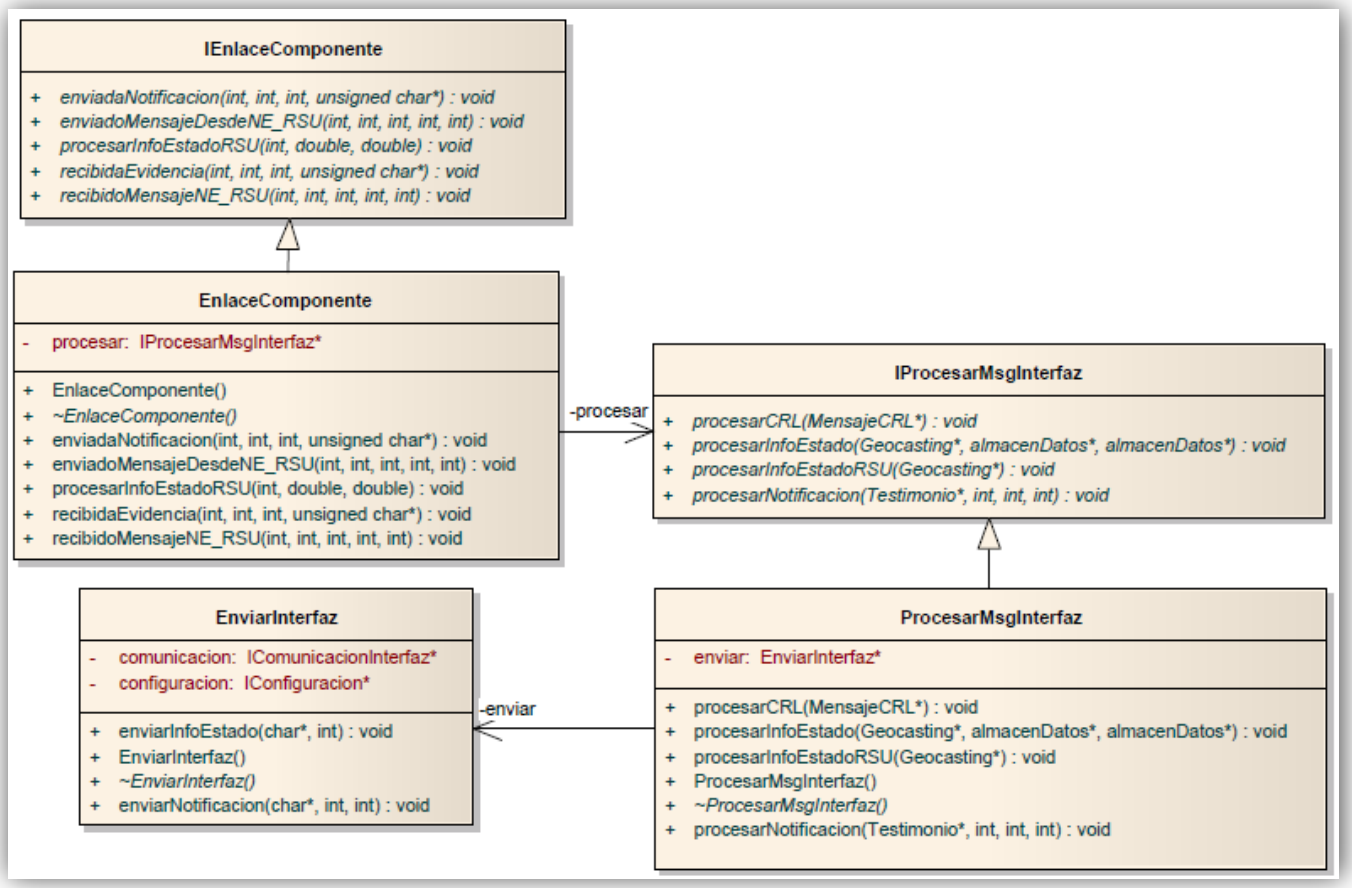


Ilustración 34: diseño componente ComunicacionInterfaz.

4.1.2.1.8 Diseño del componente Enlace

Para concluir, se presenta el componente *Enlace*, realizado con el fin de simplificar el acceso por parte de "ICar" (componente *Nodos*) e "ICarOperations" (componente *Operaciones*) a "IComunicacion" (componente *Comunicacion*), "IOpeCriptográficas" (componente *CriptografiaNodos*) e "IProcesarMsgInterfaz" (componente *ComunicacionInterfaz*).

De acuerdo con todo lo comentado, este componente proporciona una fachada para acceder a las operaciones de comunicación y, de este modo, realizar el envío y la recepción de datos. Asimismo, proporciona un acceso común para la creación de mensajes que requieran operaciones criptográficas y para la creación de los mensajes a enviar al sistema de visión de mensajes de la simulación.

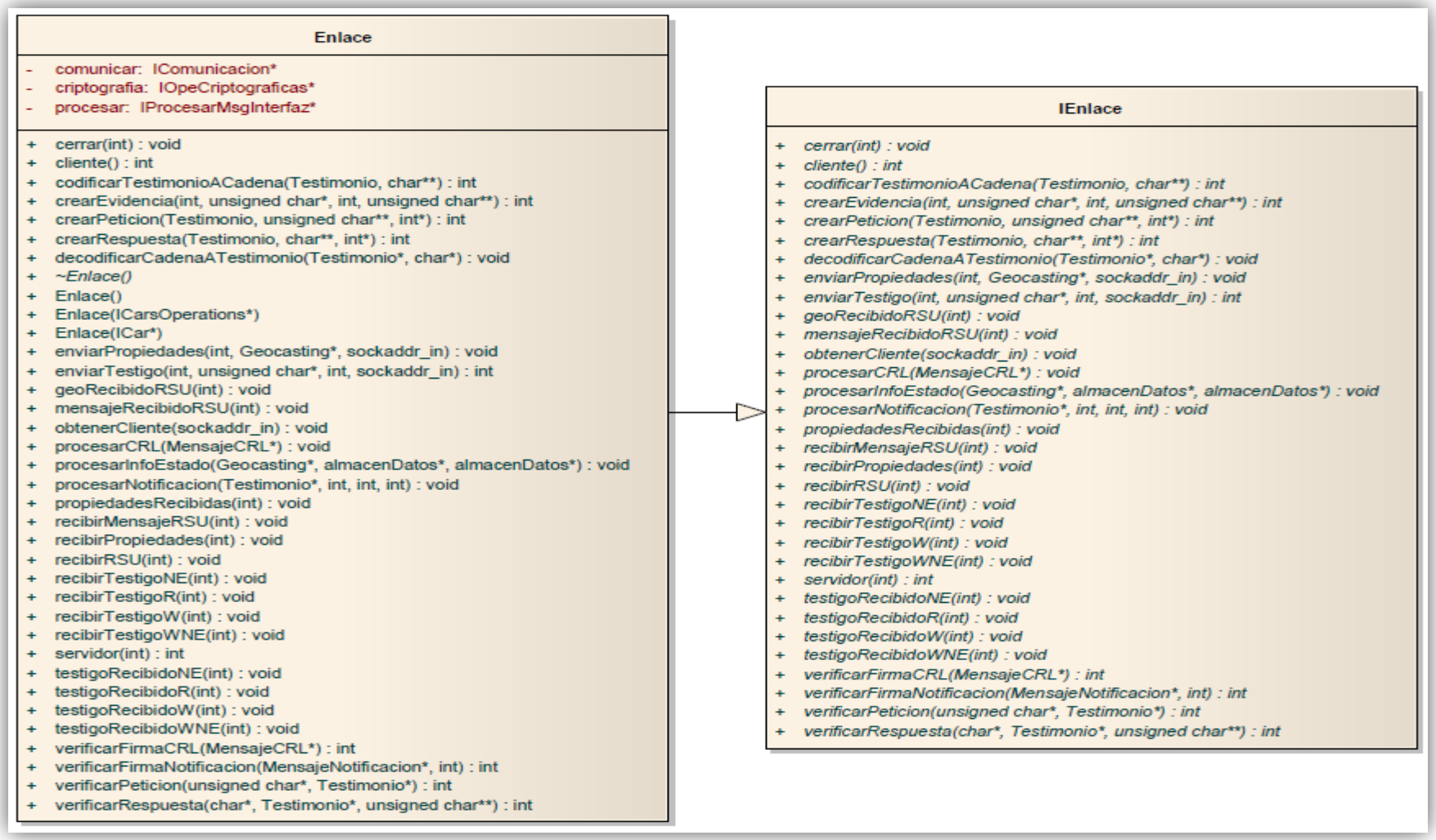


Ilustración 35: diseño componente Enlace.

4.1.2.2 Diseño del componente NCTUns

Por otra parte, este componente es el responsable del funcionamiento del simulador NCTUns 5.0 (1).

Teniendo en cuenta el estudio del simulador realizado en la sección 3.2 y las conclusiones obtenidas, se determina que es necesaria la modificación de una de las clases incluidas dentro del componente *SimulationEngine*, en concreto la clase "event" y más detalladamente la función "read_ trafficGen", en la que se produce la creación de los nodos participantes en cada una de las simulaciones.

Considerando lo comentado, se realiza la identificación y almacenamiento de una lista con los nodos con rol Witness, NoEquipped o RSU participantes en la simulación, el procesamiento de dicha lista y el envío de la misma. Las operaciones incluidas son:

- Función "read_trafficGen": desarrollada previamente, pero la cual hay que modificar para conseguir conocer el tipo y los identificadores de los nodos participantes en la simulación.
- Función "leerPuertoInterfaz": lectura del puerto correspondiente para el envío de los nodos participantes en la simulación al sistema de visión de mensajes.
- Función "enviarInfoEstado": envío de la lista de nodos participantes en la simulación al sistema de visión de mensajes.

Es importante subrayar que el componente *SimulationEngine* consta de un gran número de clases y estructuras, pero dado que únicamente se tiene que modificar la clase "event" no se han incluido en la Ilustración 36 para simplificar la comprensión de los cambios.

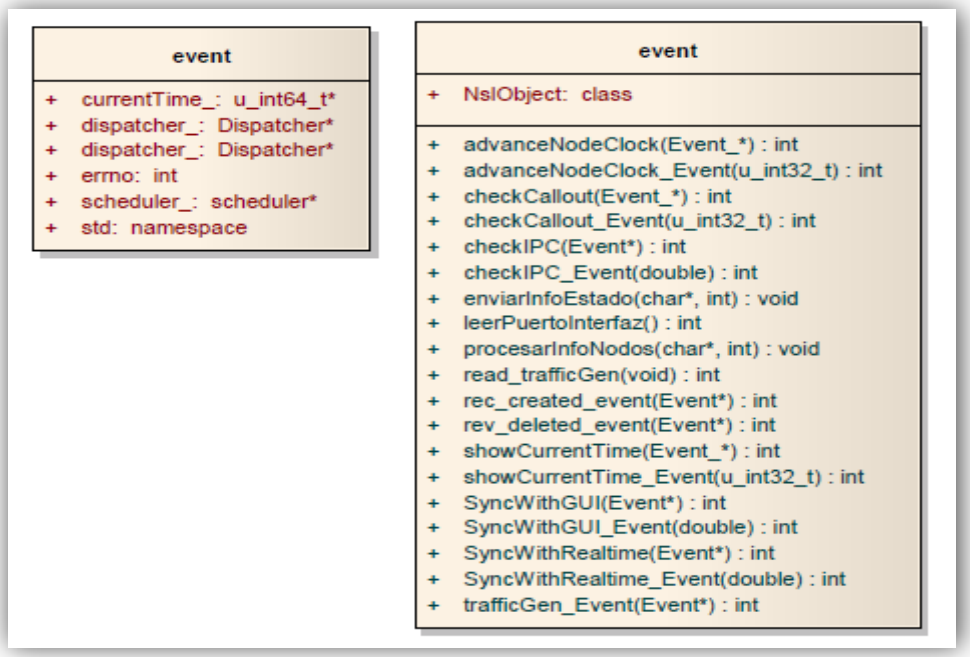


Ilustración 36: diseño componente NCTUns.

4.1.2.3 Diseño del componente ProtocoloEVIGEN_EntornoInfraestructura

Primeramente, es imprescindible señalar que la descripción completa y detallada de este componente se desarrolla en el proyecto complementario (2). Por ello, en esta sección únicamente se describirán las funciones que han de ser utilizadas en el presente proyecto.

4.1.2.3.1 Diseño del componente TomarMedidas

Considerando lo comentado, se hace uso del componente *TomarMedidas* para la colaboración con la obtención de estadísticas, realizadas en el proyecto complementario (2). La clase utilizada se denomina "TomarMedidas", en concreto la interfaz "ITomarMedias". Seguidamente se describen y muestran las funciones que deben ser utilizadas, correspondientes con la Ilustración 37:

- Función "medirBytesTransmitidos": medición de los bytes transmitidos en un mensaje determinado.
- Funciones "comenzarMedirTiempoEnvio"/"terminarMedirTiempoEnvio": comienzo/finalización de la medición del tiempo de envío de un mensaje.
- Función "comenzarMedirTiempoRespuestaGeocast": inicio de la medición del tiempo de envío de un mensaje de solicitud.
- Función "terminarMedirTiempoRespuesta": finalización de la medición tras la recepción de testimonios correspondientes a la solicitud que se comenzó a medir en "comenzarMedirTiempoRespuestaGeocast".
- Función "comenzarMedirTiempoComputacion"/ "terminarMedirTiempoComputacion": comienzo/finalización de la medición del tiempo requerido para la realización de una operación criptográfica.
- Funciones "aceptarParticipacionProtocolo"/"rechazarParticipacionProtocolo": determinación de la aceptación o el rechazo en la participación en el protocolo por parte de un nodo con rol Witness o NoEquipped.

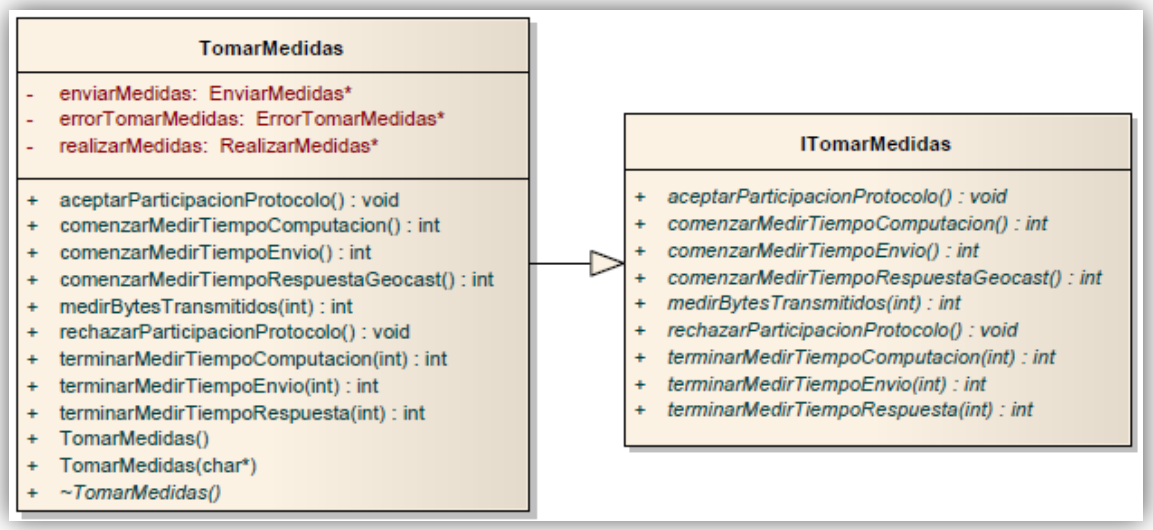


Ilustración 37: diseño componente TomarMedidas.

4.1.2.3.2 Diseño componente Criptografia

Este componente es el que comprende la implementación de las operaciones criptográficas necesarias para la creación de los mensajes intercambiados en el protocolo, tal y como se detalló al comienzo de la sección de análisis (sección 3.1.1). Del mismo modo que se comentó anteriormente, únicamente se describirán y mostrarán las funciones que han de ser utilizadas, cuya implementación se sitúa en la clase "Criptografía", con interfaz asociada "ICriptografía", y cuyo diagrama se corresponde con la Ilustración 38. Las funciones requeridas son:

- Función "obtenerCertificadoNombreFichero": obtención del certificado de un determinado nodo.
- Función "cifrar"/"cifrarNombreFichero": cifrado de una determinada cadena, bien proporcionando el certificado o bien indicando el fichero en el que se sitúa el certificado con el que realizar la operación.
- Función "descifrar"/"descifrarClaveFichero": equivalente a "cifrar"/"cifrarNombreFichero" pero para la operación de descifrado.
- Función "verificarFirma"/"verificarFirmaNombreFichero": verificar la corrección de una firma, bien proporcionando el certificado o bien indicando el fichero en el que se sitúa el certificado con el que realizar la operación.
- Función "comprobarCertificado": verificar la corrección de un certificado.
- Función "firmar"/"firmarClaveFichero": firmar una determinada cadena, bien proporcionando la clave privada o bien indicando el fichero en el que se sitúa la clave privada con la que realizar la operación.

- Función "realizarSHA1": creación de un resumen con el algoritmo SHA1 a partir de una determinada cadena.
- Función "generarClaveAES": obtención de una clase AES.
- Función "cifrarAES": cifrado de una cadena mediante la utilización de una clave AES.

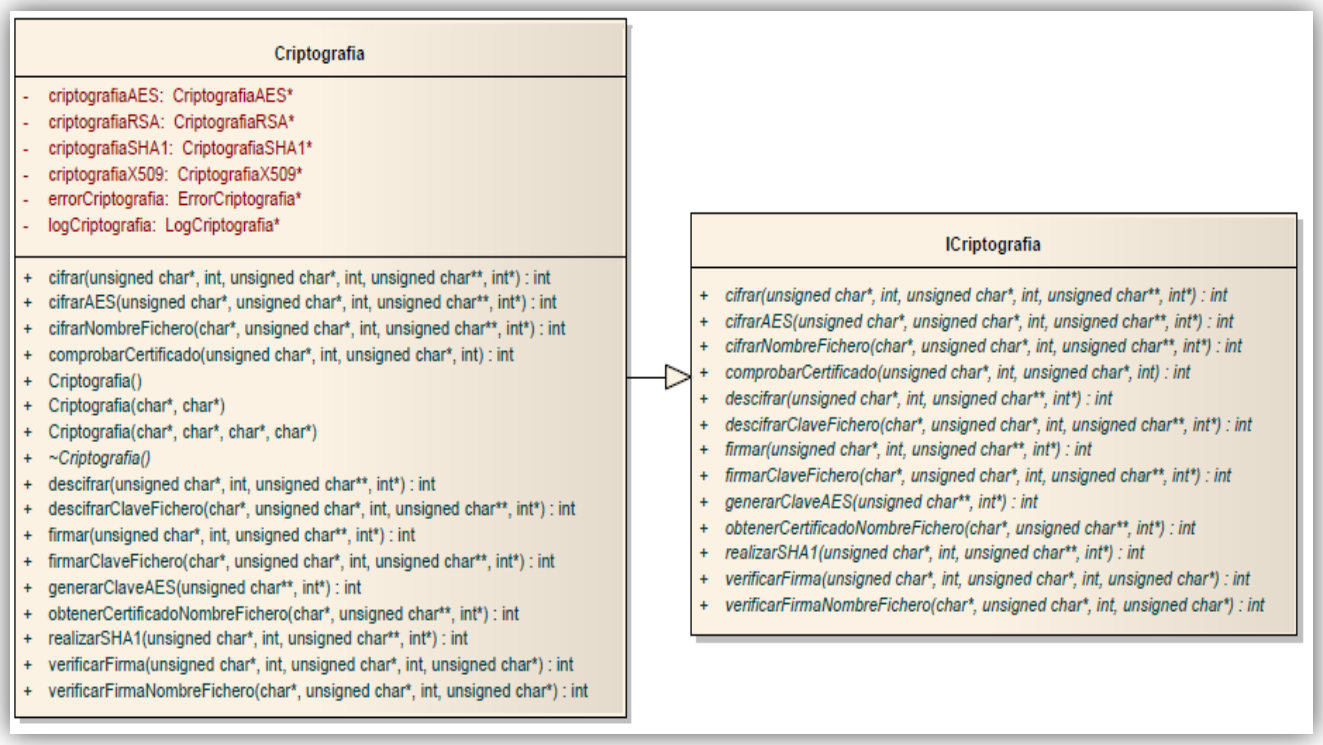


Ilustración 38: diseño componente Criptografia.

4.2 Diagramas de secuencia

Una vez concluida la descripción de las clases de cada uno de los componentes del sistema, se procede a la presentación de los diagramas de secuencia que permitan mostrar la interacción entre las distintas clases.

Al igual que en otras ocasiones, esta sección se divide en los diagramas de secuencia del sistema de visualización de mensajes y en los diagramas de secuencia de la extensión de NCTUns para la simulación del protocolo EVIGEN.

Previamente a la presentación de cada uno de ellos es conveniente subrayar la eliminación de las interfaces en todos los diagramas que van a ser presentados, facilitando la comprensión de los mismos. Por tanto, hay que considerar que todas las llamadas a funciones situadas en distintos componentes han de ser realizadas a través de interfaces.

4.2.1 Diagramas de secuencia del sistema de visión de mensajes de la simulación.

En esta sección se presentan los diagramas asociados a cada uno de los casos de uso del usuario sistema, establecidos en la sección 3.8.2.1.

CU-01 Escoger nodo

La interacción entre las clases necesarias para la elección de uno de los nodos participantes en la simulación se presenta en la Ilustración 39.

Para comenzar, el usuario accede a la lista de nodos participantes por medio de un elemento proporcionado, como puede ser un botón, dependiente de la elección tomada por el programador y asociado al método “escogerNodo”. Para realizar esta función se accede al método “getListaNodos” situado en el componente *Controlador*.

Por otra parte, una vez obtenida la lista, únicamente es necesaria la elección de uno de los nodos presentados, lo cual también se realiza considerando el elemento proporcionado según la decisión del programador. Es importante resaltar la necesidad de notificar al *Modelo*, método “setNodoEnPantalla” de las clases “NotificacionesRecibidas” y “DatosRecibidos”, el nodo que ha sido escogido para que en las peticiones posteriores sólo se reciba información relativa a dicho nodo.

Estas dos pequeñas acciones, tal y como se pueden apreciar en el diagrama, requieren la solicitud de información al componente *Controlador* y la posterior solicitud al componente *Modelo*. Además es conveniente observar, considerando la descripción de clases del *Modelo*, que los nodos participantes en la simulación son procesados y almacenados en la clase “DatosRecibidos”. Por ello, una vez recibida la lista, mediante “RecibirDatos”, procesada y almacenada, puede ser presentada en la *Vista* tantas veces como se desee.

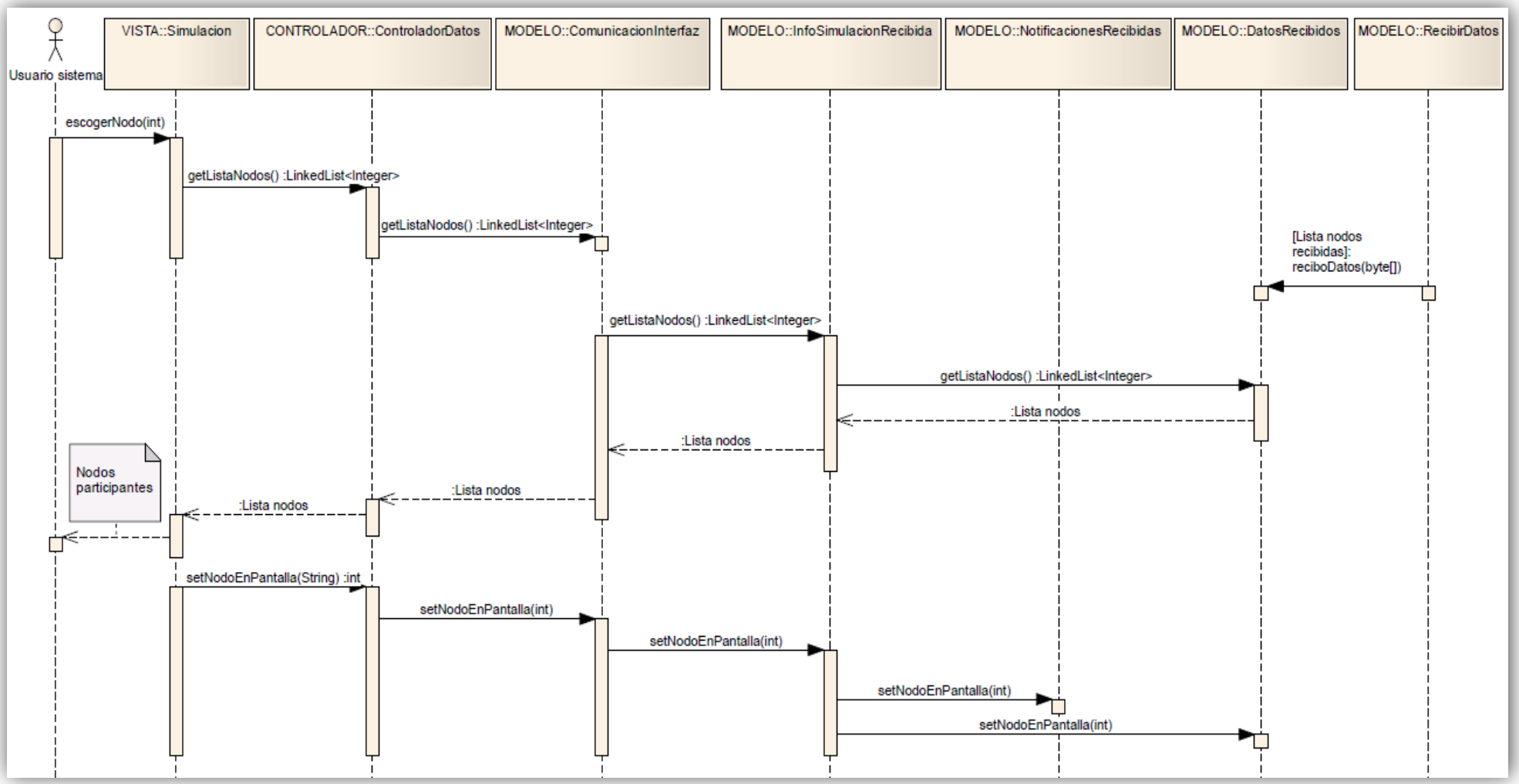


Ilustración 39: diagrama secuencia CU-01 escoger nodo.

CU-02 Visualizar mensajes

La interacción entre las clases necesarias para la visualización de los mensajes intercambiados, por un determinado nodo, es descrita en el siguiente diagrama de secuencia, Ilustración 40.

Para comenzar se realizan las dos funciones correspondientes con CU-01, obtención de la lista de nodos participantes en la simulación y elección de uno de ellos.

Posteriormente, se espera por los mensajes recibidos desde el protocolo EVIGEN, ejecutado en el simulador NCTUns 5.0 (1), para mostrarlos por pantalla. Para conseguirlo, una vez que el hilo receptor de notificaciones dispone de un mensaje, se lo notifica, mediante el método "reciboNotificaciones", a la clase "NotificacionesRecibidas" y es ésta la responsable del envío del mensaje al componente *Vista*, haciendo uso de "pintarNotificaciones", siempre que el emisor o receptor del mensaje coincida con el nodo en pantalla. Por tanto, en este caso únicamente se produce la interacción entre los componentes *Vista* y *Modelo*.

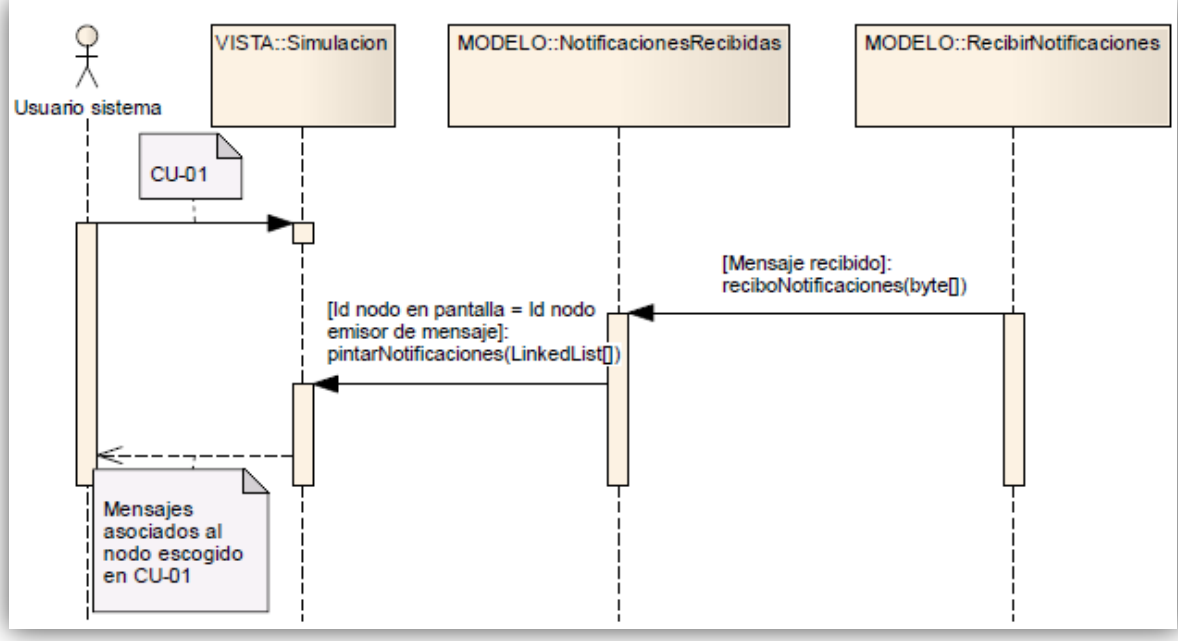


Ilustración 40: diagrama secuencia CU-02 visualizar mensajes.

CU-03 Visualizar estado

La interacción entre las clases necesarias para la visualización de la información de estado de cada uno de los nodos, así como de la información de estado de los nodos considerados vecinos a uno escogido, se muestra en el siguiente diagrama de secuencia, Ilustración 41.

Del mismo modo que en el caso anterior, tras realizar las operaciones asociadas a CU-01, obtención de la lista de nodos participantes en la simulación y elección de uno de ellos, se espera por la información recibida por parte del simulador NCTUns 5.0 (1).

El procedimiento de recibo de información es equivalente al mencionado anteriormente, por medio del método "reciboDatos" de la clase "DatosRecibidos" se proporcionan los datos recibidos para ser procesados, almacenados y, en este caso, presentados en la *Vista* a través de "pintarInfoEstado", siempre que dichos datos estén asociados al nodo en pantalla.

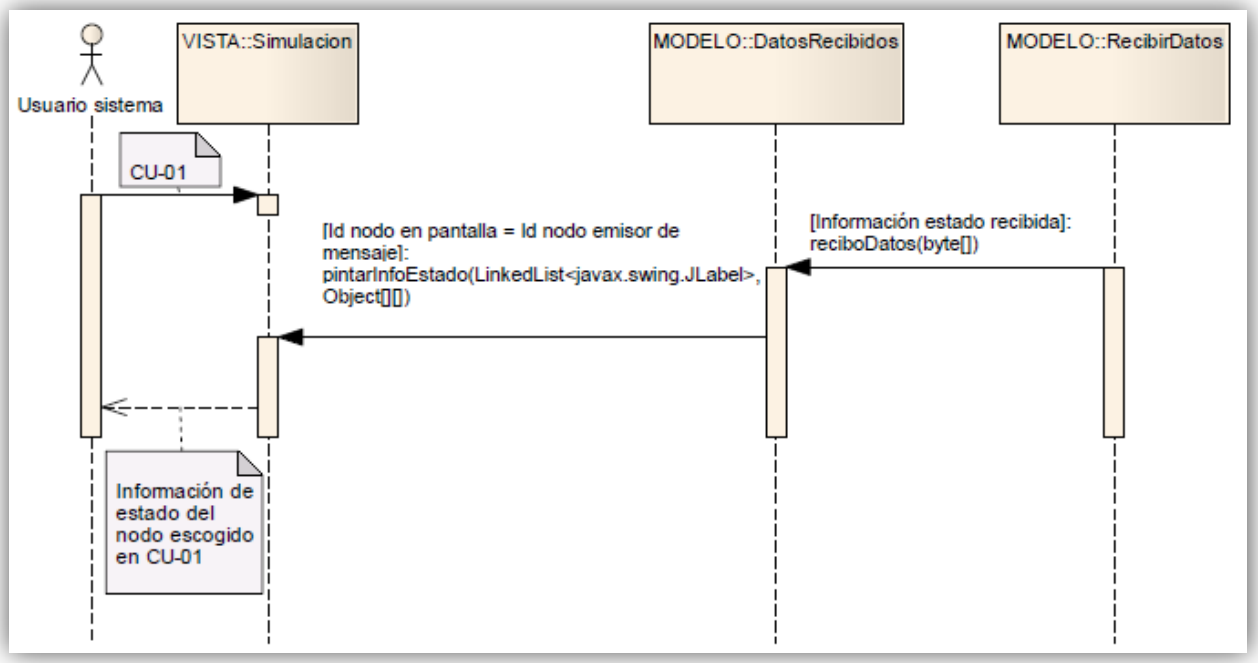


Ilustración 41: diagrama secuencia CU-03 visualizar estado.

CU-04 Visualizar CRL

La interacción entre las clases necesarias para la visualización de la lista de identificadores de nodos con certificado revocado es presentada en el diagrama posterior, Ilustración 42.

Esta función únicamente requiere la pulsación del elemento indicado por el programador, asociado a la función de "observarCRL". Además, hay que considerar que desde la clase "RecibirCRL" se proporcionará a la clase "CRLRecibidas" la CRL procedente de la ejecución del protocolo EVIGEN en el simulador NCTUns 5.0 (1).

En este caso se realiza una solicitud al *Controlador* y de éste al *Modelo*, de modo que sea el *Controlador* el que verifique la información obtenida y responda en consecuencia.

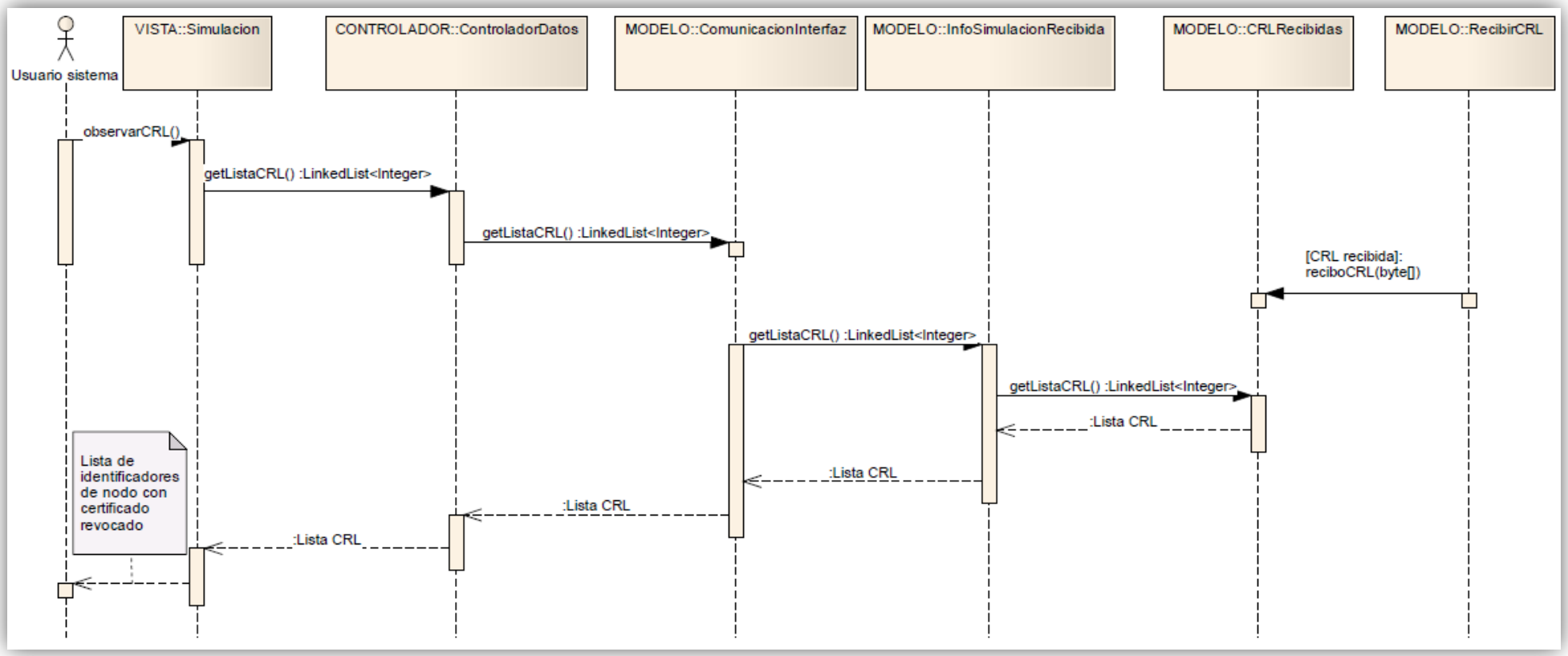


Ilustración 42: diagrama secuencia CU-04 visualizar CRL.

CU-05 Obtener PDF

La interacción entre las clases necesarias para la creación de un fichero PDF se presenta en el siguiente diagrama de secuencia, Ilustración 43.

La funcionalidad únicamente requiere la pulsación del elemento que el programador determine para este fin, vinculado al método "almacenarPDF", así como la completitud de los campos necesarios para concluir el proceso.

Del mismo modo que ocurre en situaciones anteriores, se produce la interacción entre los componentes *Vista*, *Modelo* y *Controlador*, realizando la solicitud requerida y actuando en consecuencia.

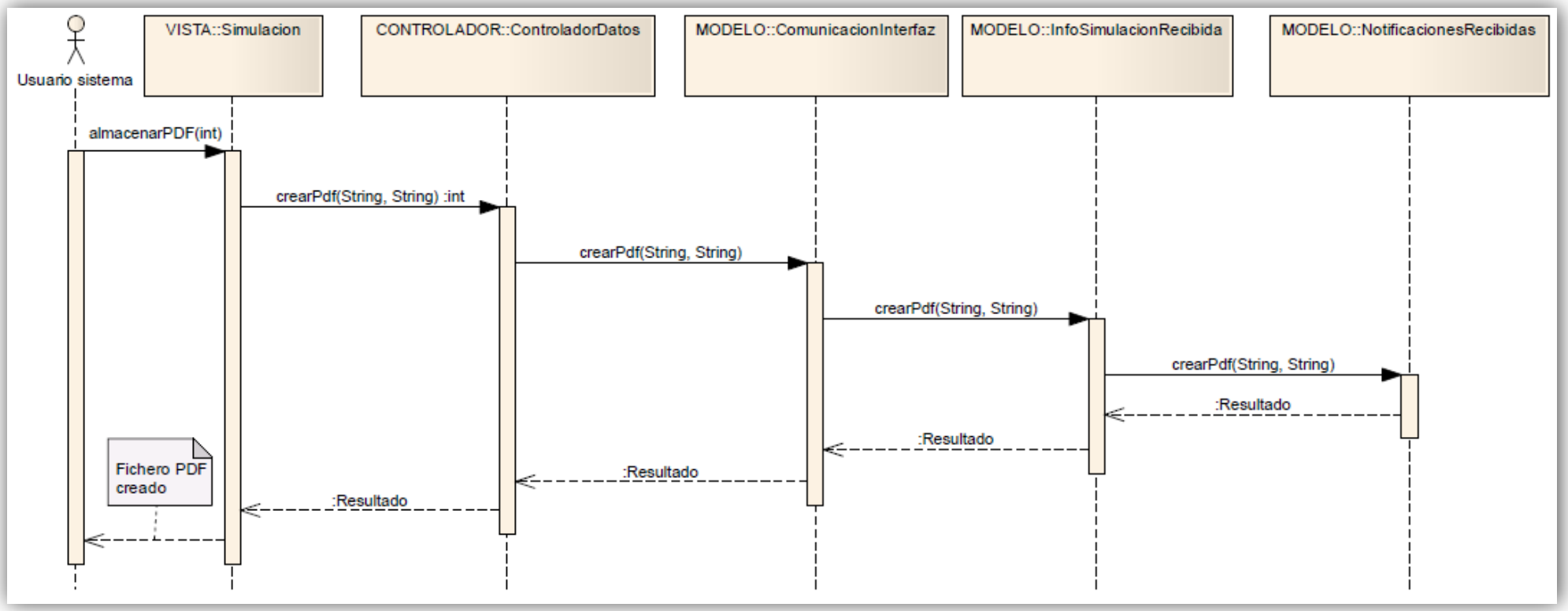


Ilustración 43: diagrama secuencia CU-05 obtener PDF.

CU-06 Limpiar pantalla

La interacción entre las clases necesarias para realizar la limpieza de la pantalla se presenta en el siguiente diagrama, Ilustración 44.

En este caso sólo es necesario pulsar en el elemento proporcionado por el programador para la realización de esta función, asociado al método "limpiar".

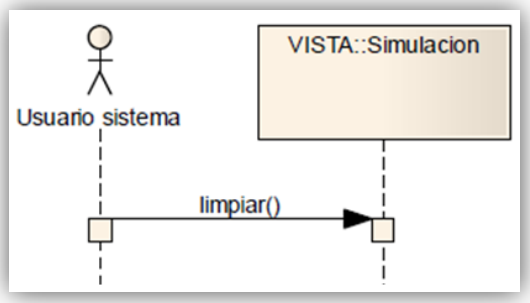


Ilustración 44: diagrama secuencia CU-06 limpiar pantalla.

CU-07 Acceder manual

La interacción entre las clases necesarias para acceder al manual de usuario se muestra en el siguiente diagrama, Ilustración 45.

Del mismo modo que se ha comentado en el diagrama anterior, sólo se tiene que pulsar el elemento indicado por el programador para la realización de esta función.

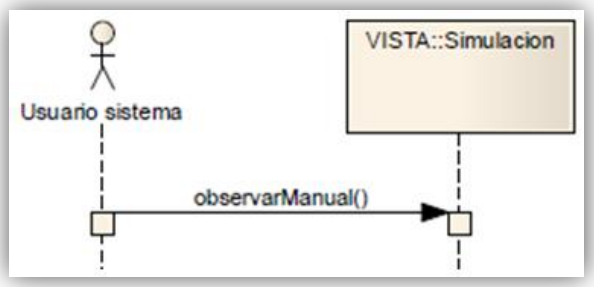


Ilustración 45: diagrama secuencia CU-07 acceder manual.

CU-08 Visualizar mensajes simulación previa

La interacción entre las clases necesarias para visualizar los mensajes intercambiados en una simulación tras la realización de una pausa se muestra en el siguiente diagrama, Ilustración 46.

Tal y como se ha descrito en otras ocasiones, sólo se requiere pulsar el elemento indicado por el programador para la realización de esta función, el cual se vincula al método "continuar".

Posteriormente se realizan las mismas operaciones que se describieron en el diagrama "CU-02 Visualizar mensajes", de modo que a la pantalla lleguen los mensajes procedentes de la ejecución de una simulación efectuada en el simulador.

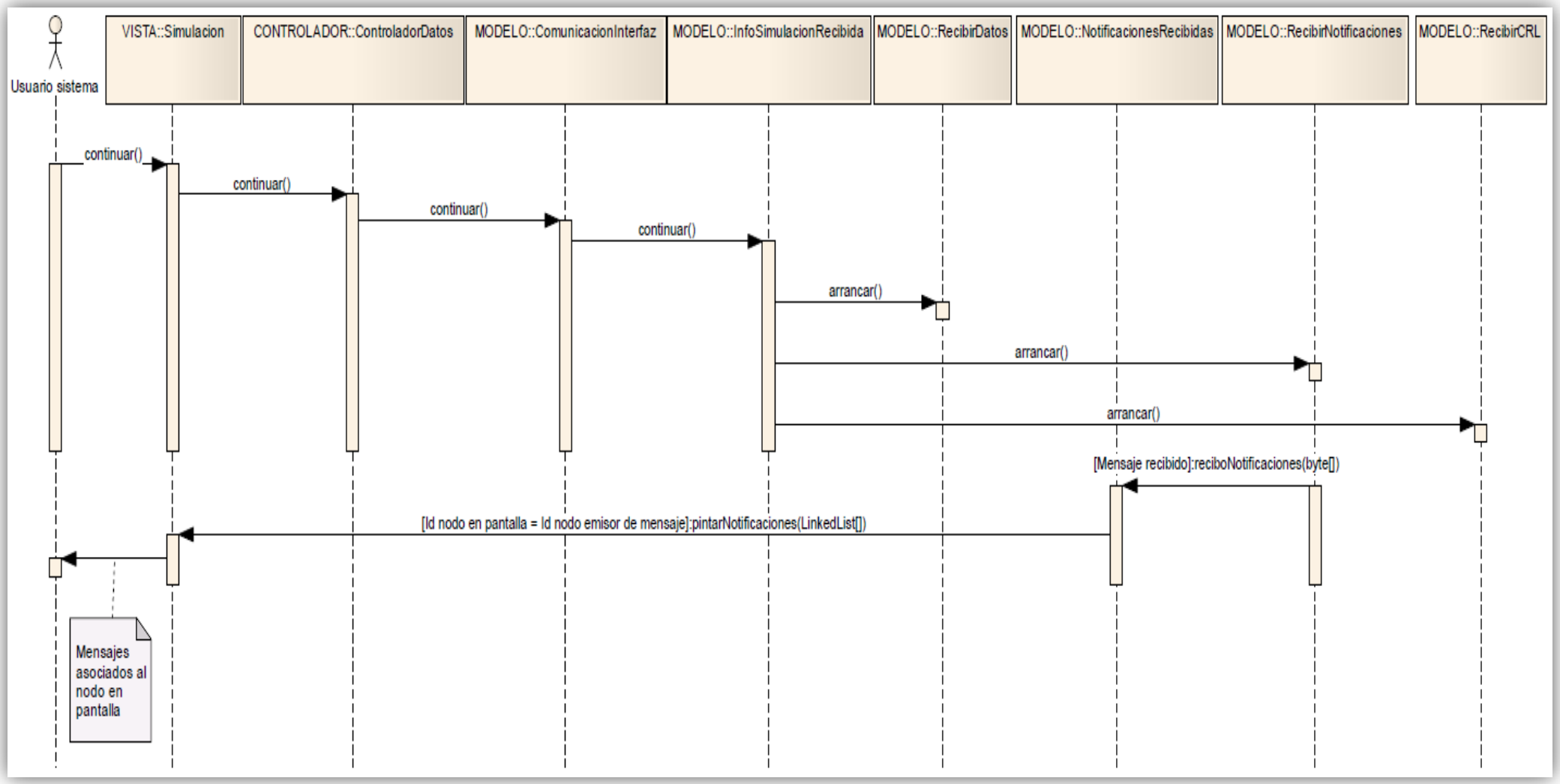


Ilustración 46: diagrama secuencia CU-08 visualizar mensajes simulación previa.

4.2.2 Diagramas de secuencia del protocolo EVIGEN

En esta sección se presenta el diagrama asociado al caso de uso del usuario NCTUns, establecido en la sección 3.8.2.2, el cual se corresponde con la ejecución completa del protocolo EVIGEN.

Sin embargo, a pesar de la existencia de un único caso de uso, el diagrama de secuencia resultante es demasiado extenso como para presentarlo y describirlo en una única imagen, de modo que se descompone en tres grandes secciones. En la primera de ellas se describe el arranque del sistema. En la segunda se detalla el intercambio de mensajes realizado entre los distintos tipos de roles, subdividida en dos, los diagramas de la recepción de datos y los diagramas del procesamiento de mensajes recibidos y el envío de los mensajes respuesta. En la tercera y última, se especifican operaciones de soporte, las cuales son el establecimiento de la configuración y la realización de las operaciones criptográficas.

Por otra parte, con anterioridad al desarrollo de esta sección, se subraya que las interacciones con las funciones vinculadas a los componentes *Criptografia* y *TomarMedidas*, desarrolladas en el proyecto complementario (2), no serán mostradas. De este modo, se simplifica y visualiza con mayor claridad las interacciones propias del desarrollo realizado en el presente proyecto. No obstante, el programador debe considerar las relaciones establecidas con ambos componentes, presentadas en la arquitectura definitiva (sección 3.6.2), y hacer uso de las funciones especificadas en el diseño de dichos componentes (sección 4.1.2.3.1 y sección 4.1.2.3.2) para contribuir tanto en el éxito del proyecto complementario (2) como en el éxito del presente proyecto.

4.2.2.1 Diagramas de secuencia del arranque del sistema

Esta sección, en la que se presentan las secuencias de las funciones ejecutadas para el arranque del protocolo EVIGEN, se divide en cuatro secciones. Por un lado, se presenta el arranque general, responsable de la inicialización de hilos y almacenamiento de variables. Otra de las secciones se corresponde con el arranque del rol Requester, el cual es realizado por todos los nodos de la simulación con rol Witness y NoEquipped. Por otra parte, se muestra el arranque del rol Witness, necesario para aquellos nodos que desempeñen este tipo de rol. Finalmente, se describe el arranque del rol NoEquipped, realizado en los nodos que asuman este rol en lugar del rol Witness.

CU-09 Parte 1.1 Arranque general del protocolo

La secuencia de clases relacionadas con el arranque general del protocolo se muestra en los diagramas de secuencia posteriores, relativos a la Ilustración 47, a la Ilustración 48, a la Ilustración 49, a la Ilustración 50 y a la Ilustración 51.

Previamente a la descripción del diagrama se ha de señalar que el nodo que en este caso se arranca se corresponde con el rol de Witness, requiriéndose el mismo diagrama pero

con el componente *NoEquipped*, para presentar el arranque general en un nodo con dicho rol.

Tras la ejecución de todas las funciones del componente *NCTUns*, el arranque general del protocolo comienza, Ilustración 47, con la lectura de las variables y puertos de configuración, así como cargando en memoria los elementos necesarios para la ejecución del protocolo, lo cual se corresponde con las distancias máximas a las que los nodos son considerados vecinos, el tiempo máximo que un mensaje puede mantenerse almacenado, el tiempo máximo tras el cual se reenvía una solicitud, el número de veces que un mensaje puede enviarse y los puertos de comunicación de datos entre los distintos nodos. Todo ello realizado por medio de las funciones necesarias pertenecientes al componente *Configuracion* y ampliado en el diagrama de la Ilustración 73, "Establecimiento de la configuración".

Posteriormente, en el segundo de los diagramas, Ilustración 48, se realiza el arranque del movimiento del nodo, mediante el arranque del hilo creado para este propósito, a través de "crearMovimiento". Asimismo, se efectúa tanto el arranque del rol correspondiente, Witness en este caso, como el arranque del rol de Requester para ejecutarse en el momento que se reciba una notificación de sanción. Todo esto continúa en los diagramas de la Ilustración 52 y de la Ilustración 53, "Arrancar Requester" y "Arrancar Witness", respectivamente.

Por otro lado, en la Ilustración 49, en la Ilustración 50 y en la Ilustración 51, se muestran los diagramas de secuencia correspondientes con el arranque del hilo de recepción de mensajes de *beaconing* de nodos con rol Requester, Witness o NoEquipped, el arranque del hilo de recepción de mensajes de *beaconing* de nodos con rol RSU y el arranque del hilo de recepción de mensajes de notificación de sanción y CRL. Considerando todo lo mencionado, dentro de cada hilo de ejecución se crea un servidor, función "servidor" de la clase "Sockets", y se inicia el proceso de espera por mensajes, haciendo uso de funciones de la clase "MsgOperations". Cada una de estas acciones lleva asociado un diagrama que continúa la descripción de la ejecución del protocolo.

Finalmente, en el diagrama Ilustración 51, es imprescindible señalar la última función presentada, "envioGeocastingTodos", requerida para realizar el envío periódico del mensaje de *beaconing* con la información de estado asociada a cada nodo.

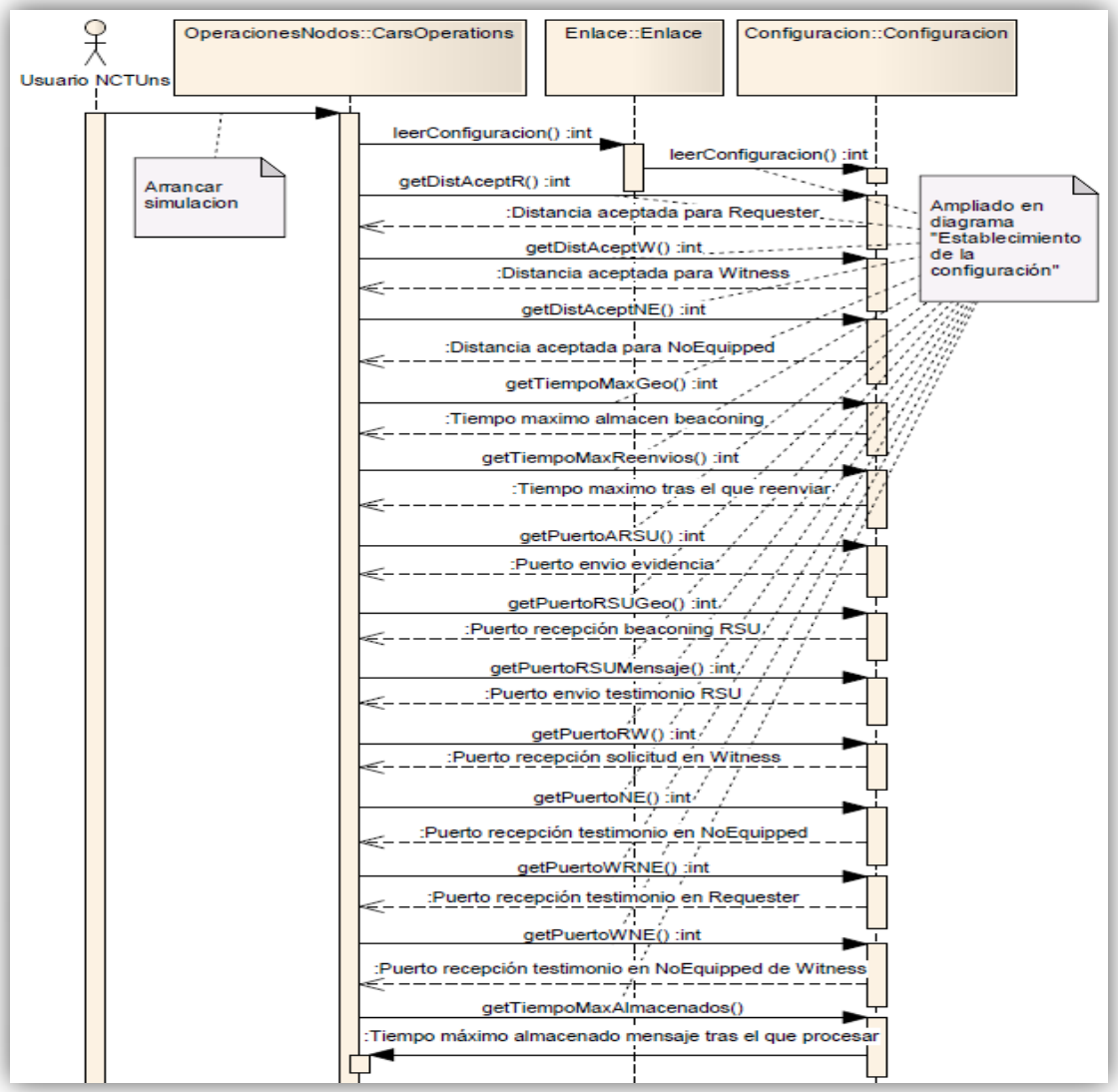


Ilustración 47: diagrama secuencia CU-09 Parte 1.1 arranque protocolo 1.

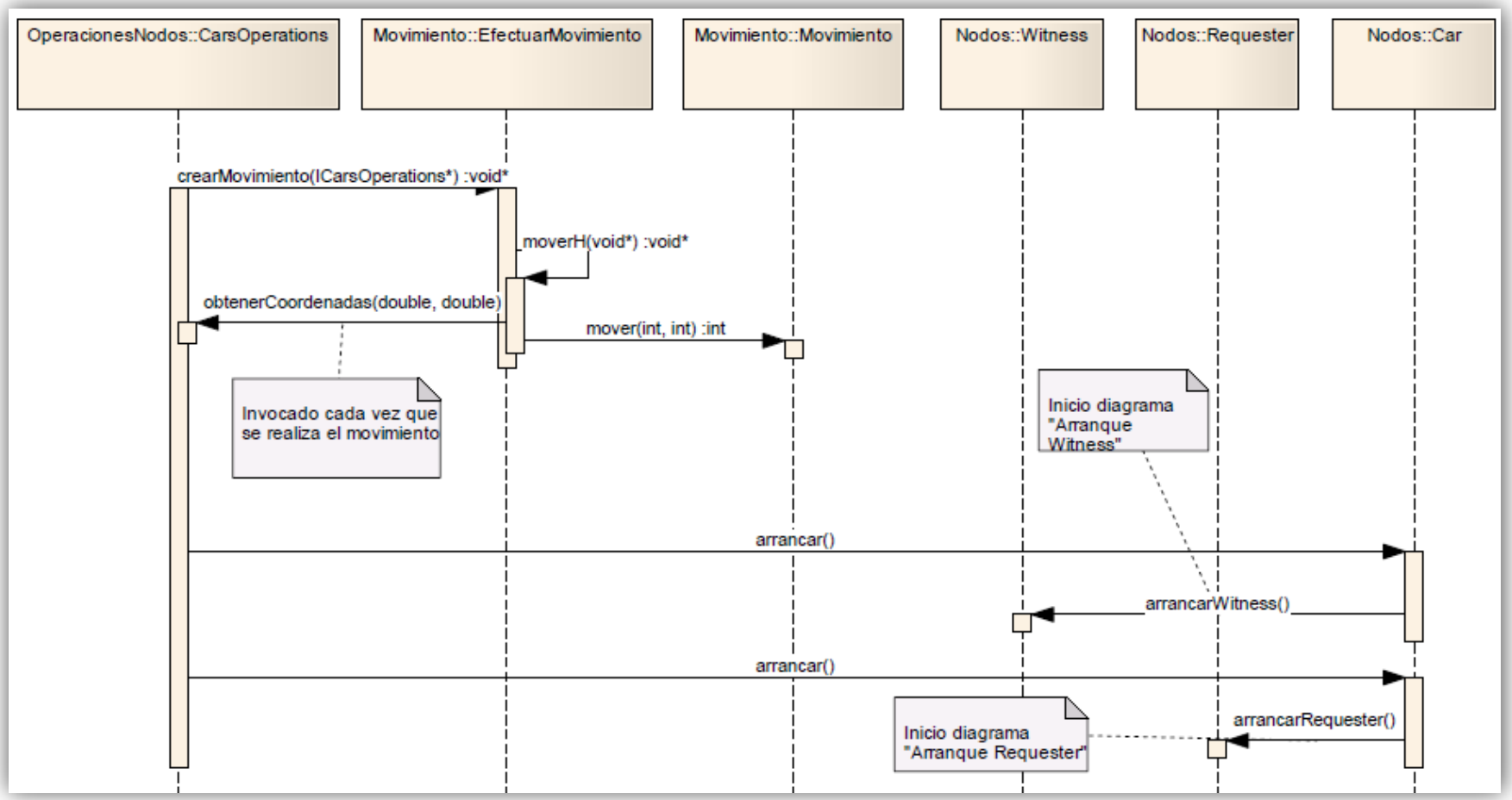


Ilustración 48: diagrama secuencia CU-09 Parte 1.1 arranque protocolo 2.

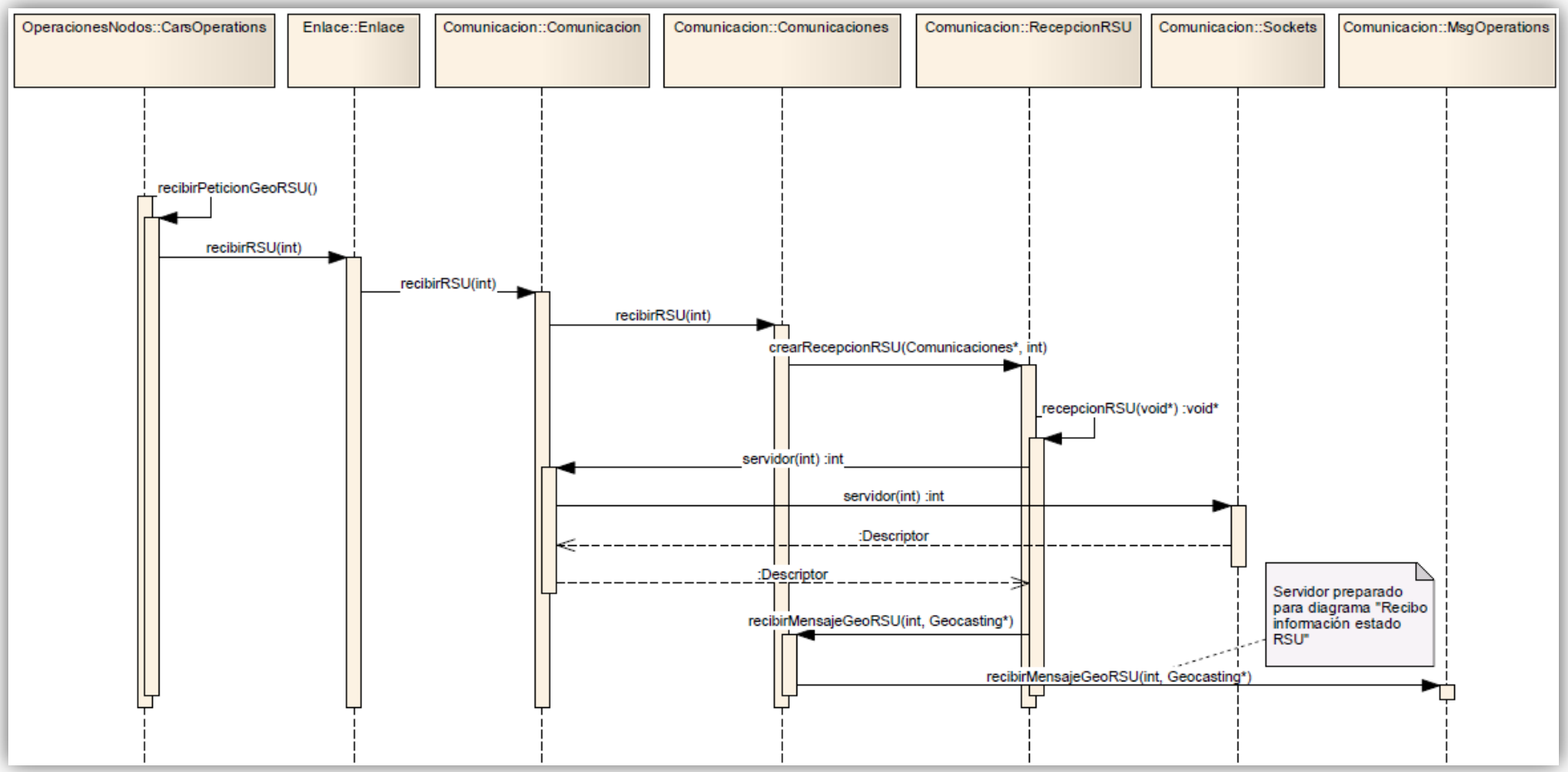


Ilustración 49: diagrama secuencia CU-09 Parte 1.1 arranque protocolo 3.

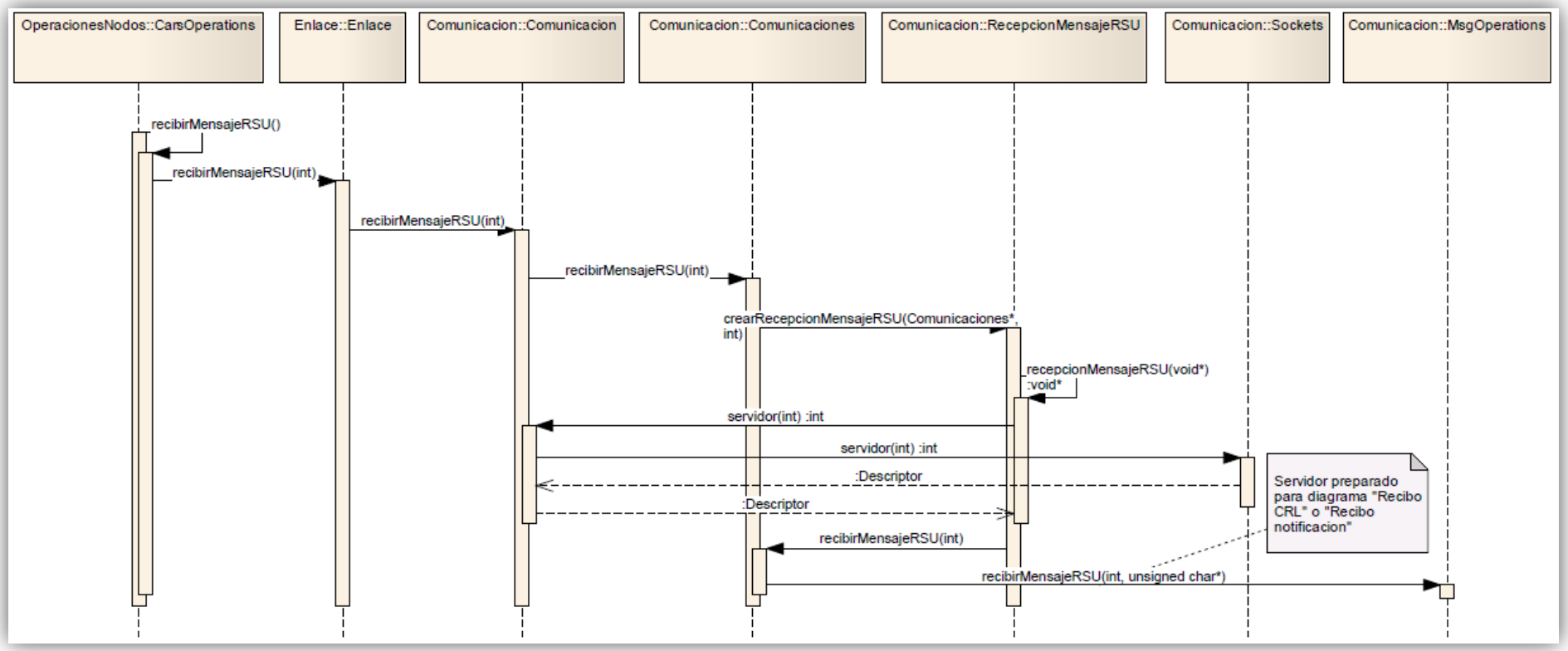


Ilustración 50: diagrama secuencia CU-09 Parte 1.1 arranque protocolo 4.

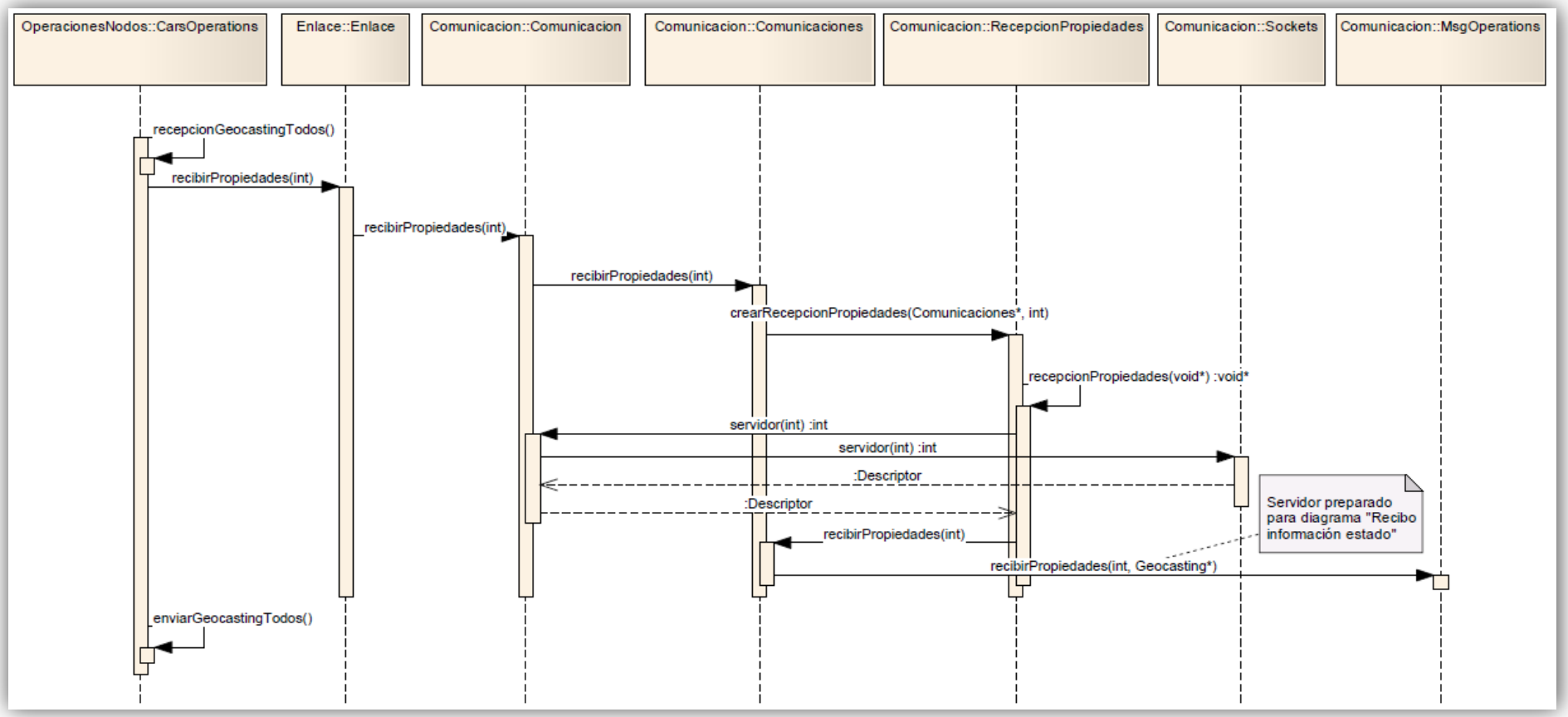


Ilustración 51: diagrama secuencia CU-09 Parte 1.1 arranque protocolo 5.

CU-09 Parte 1.2 Arranque Requester

Seguidamente al arranque general, se realiza el arranque del rol Requester, cuya secuencia de ejecución se muestra en el diagrama de la Ilustración 52.

La primera función a realizar es la carga de la variable asociada al tiempo máximo aceptado, transcurrido desde la creación de un mensaje recibido en el nodo que desempeña este rol, y el puerto necesario para la recepción de mensajes. Para realizar las acciones comentadas, se hace uso de funciones asociadas con el componente *Configuracion* y se amplía en el diagrama de la Ilustración 73, "Establecimiento de la configuración".

Posteriormente, por medio del componente *Enlace* se accede a la clase "Sockets", función "cliente", para realizar la creación de los clientes necesarios para el envío de solicitudes y de las correspondientes evidencias.

Por último, se arranca el hilo receptor de los testimonios vinculados a las solicitudes enviadas, creándose el servidor asociado, también mediante la clase "Sockets", función "servidor", e iniciando el proceso de espera por los mensajes, haciendo uso de la función "recibirTestigo" de "MsgOperations". Este proceso continúa en el diagrama de la Ilustración 60, "Recibo testimonio en Requester".

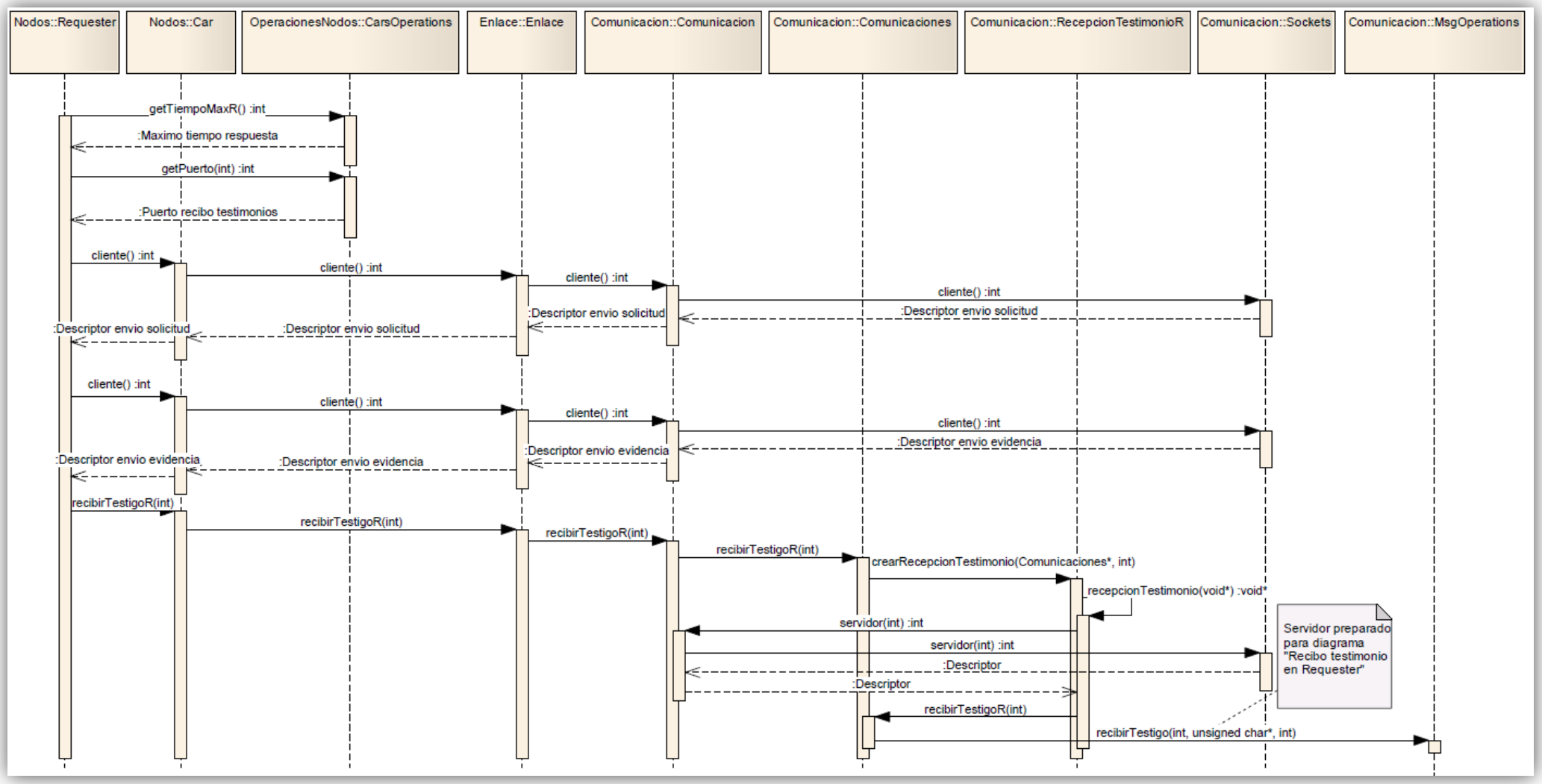


Ilustración 52: diagrama secuencia CU-09 Parte 1.2 arranque Requester.

CU-09 Parte 1.3 Arranque Witness

Otro de los arranque que han de ser realizados, excepto al realizar el arranque de un NoEquipped, es el del rol de Witness, cuya secuencia de ejecución se muestra en el diagrama de la Ilustración 53.

Al igual que en el rol Requester, lo primero a realizar es la carga de las variables y puertos necesarios para el correcto funcionamiento de este rol, los cuales se corresponden con la variable asociada al tiempo máximo aceptado, transcurrido desde la creación de un mensaje recibido en el nodo que desempeña este rol, el puerto para la recepción de solicitudes y los puertos para el envío de los testimonios creados ya sea dirección Requester, NoEquipped o RSU. Para efectuar las acciones comentadas se hace uso de funciones vinculadas al componente *Configuracion* y se amplía en el diagrama de la Ilustración 73, "Establecimiento de la configuración".

Posteriormente, por medio del componente *Enlace* se accede a la clase "Sockets", función "cliente", para efectuar los envíos de los testimonios.

Por último, se arranca el hilo receptor de los testimonios correspondientes con las solicitudes enviadas, creándose el servidor asociado, también mediante la clase "Sockets", función "servidor", e iniciando el proceso de espera de los mensajes, haciendo uso de la función "recibirTestigo" de "MsgOperations". Este proceso continúa en el diagrama de la Ilustración 58, "Recibo solicitud en Witness".

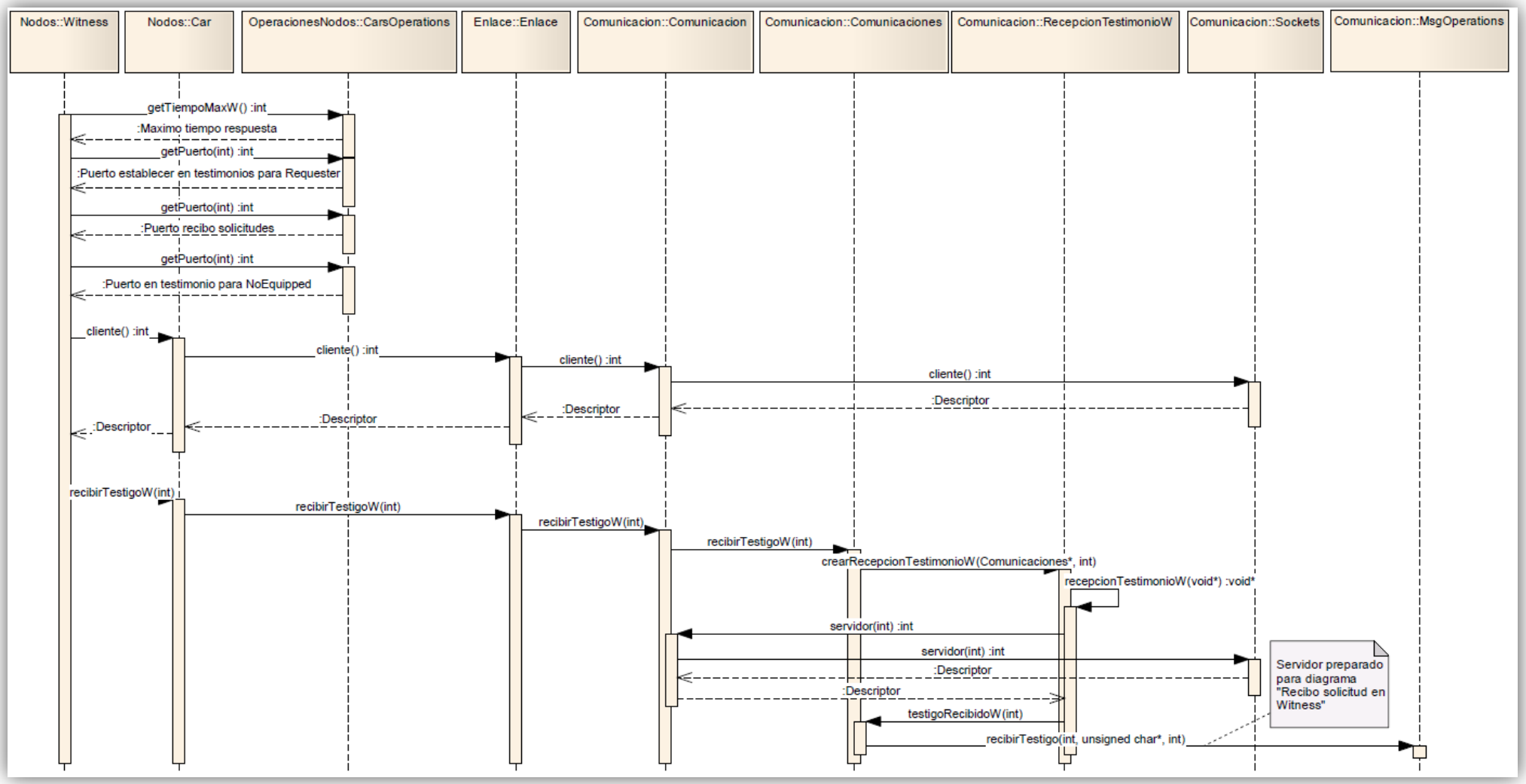


Ilustración 53: diagrama secuencia CU-09 Parte 1.3 arranque Witness.

CU-09 Parte 1.4 Arranque NoEquipped

El último de los arranques a presentar es el del rol NoEquipped, cuya secuencia de ejecución se muestra en el diagrama de la Ilustración 54 y la Ilustración 55. Tal y como se ha comentado, este arranque se produce siempre que el nodo desempeñe el rol de NoEquipped y, por ello, no se realiza el "CU-09 Parte 1.3.1 Arranque Witness".

Tal y como ocurre en el resto de roles, la primera de las funciones es la de cargar las variables y puertos necesarios para el funcionamiento del rol, los cuales se corresponden con la variable asociada al tiempo máximo aceptado, transcurrido desde la creación de un mensaje recibido en el nodo que desempeña este rol, el puerto para la recepción de testimonios procedentes de un nodo con rol Witness, el puerto para la recepción de testimonios de nodos con rol NoEquipped o RSU y el puerto para el envío de los testimonios creados dirección un nodo con rol Requester, NoEquipped o RSU. Para realizar las acciones comentadas se utilizan funciones situadas en el componente *Configuracion* y se amplía en el diagrama de la Ilustración 73, "Establecimiento de la configuración".

Posteriormente, considerando que los envíos pueden realizarse tanto a Witness como a NoEquipped o RSU, se han de crear dos clientes responsables de esta función, lo cual se efectúa haciendo uso de la clase "Sockets", función "cliente".

Finalmente, del mismo modo que se necesitan dos clientes para realizar envíos a nodos con distintos roles, se han de crear dos servidores por los que recibir tanto los testimonios procedentes de los nodos con rol Witness como los recibidos de otros NoEquipped o RSU. Esta función se realiza mediante la invocación de "servidor" de la clase "Sockets" y el establecimiento posterior de cada uno de los servidores a la espera de los mensajes, haciendo uso de la función "recibirTestigo" de "MsgOperations". Este proceso continúa en el diagrama de la Ilustración 59, "Recibo testimonio en NoEquipped".

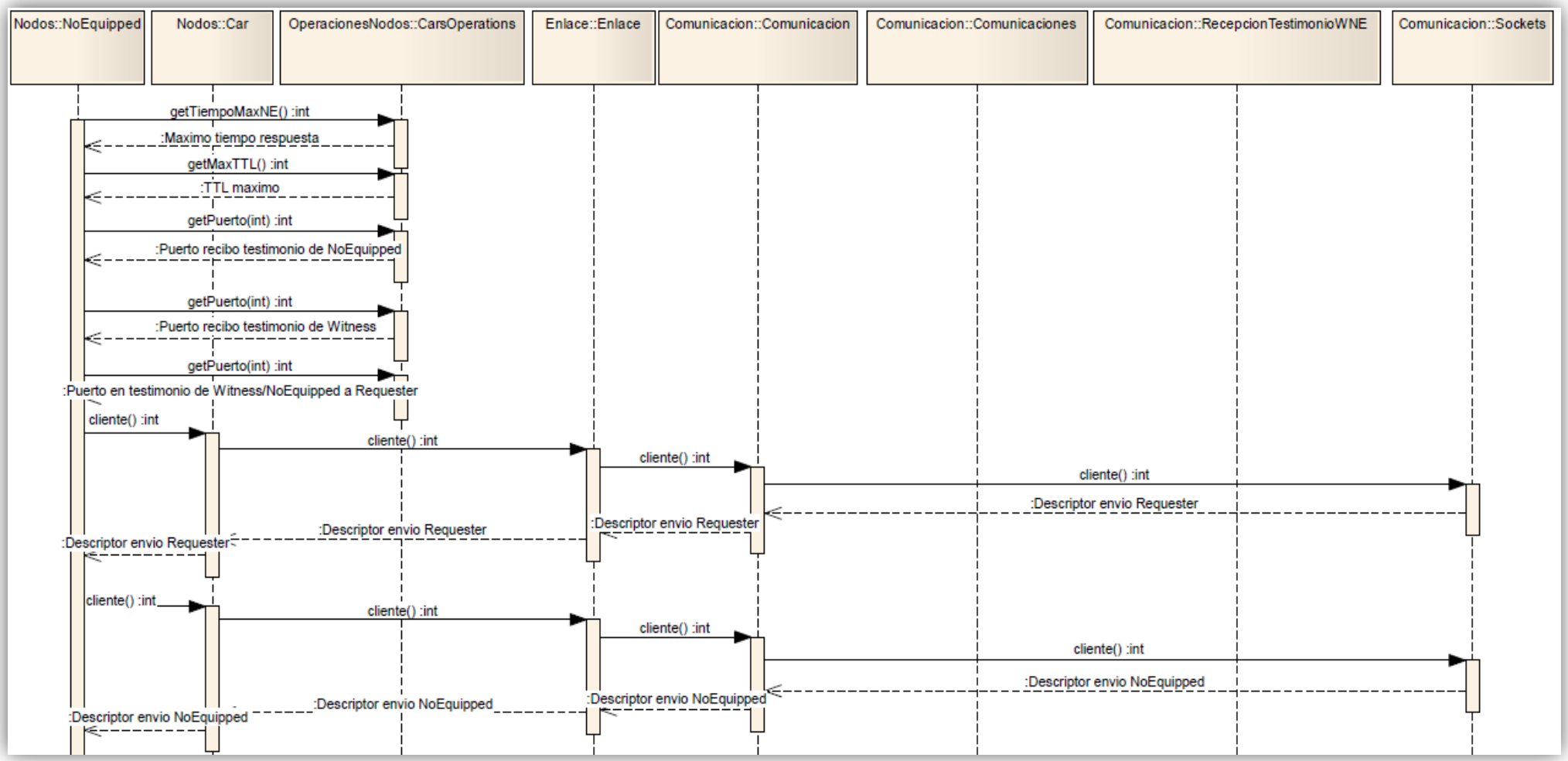


Ilustración 54: diagrama secuencia CU-09 Parte 1.4 arranque NoEquipped 1

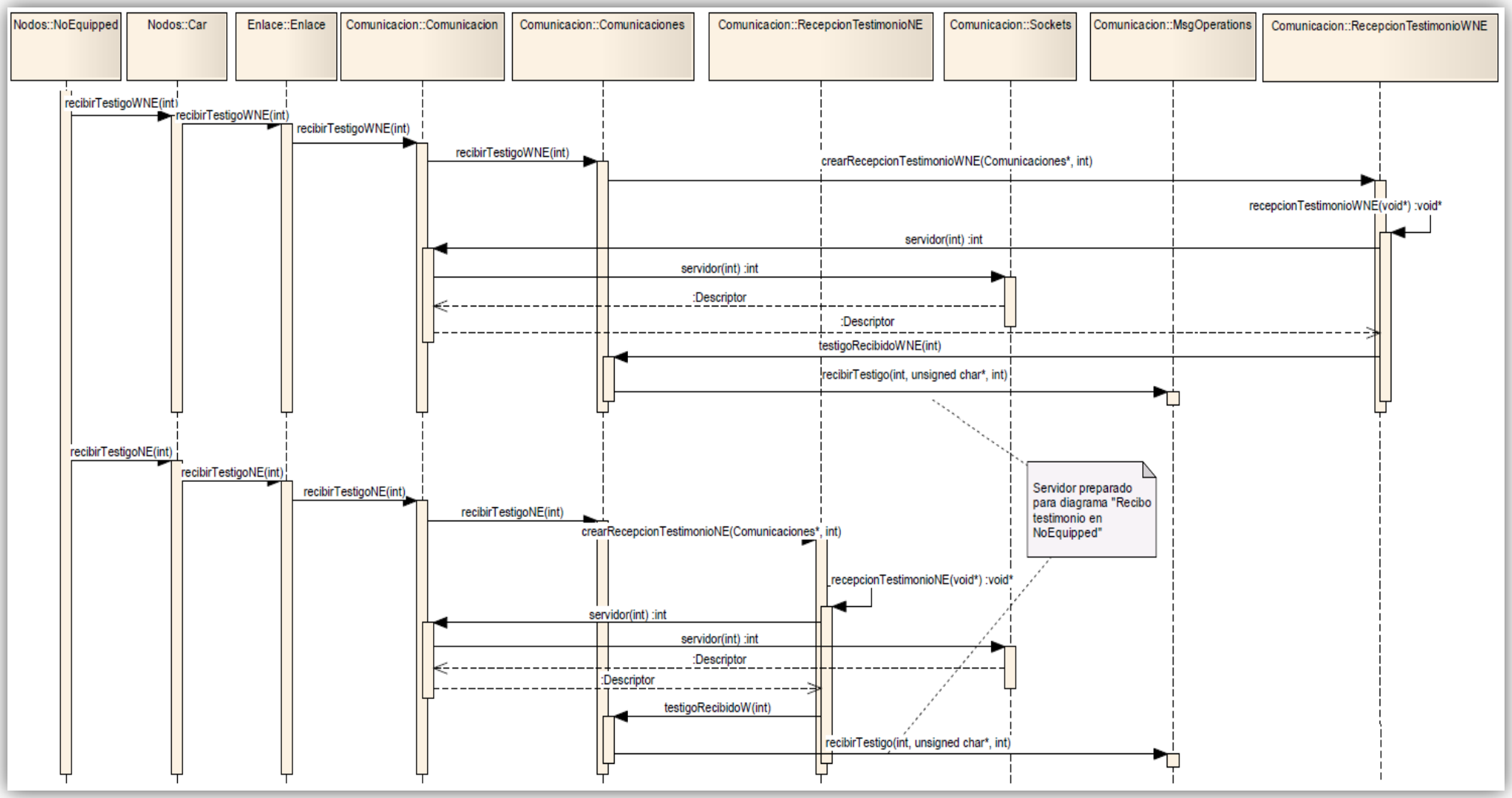


Ilustración 55: diagrama secuencia CU-09 Parte 1.4 arranque NoEquipped 2.

4.2.2.2 Diagramas de secuencia del intercambio de mensajes

En esta sección se presentan los diagramas de secuencia asociados el envío y recepción de los mensajes intercambiados en el protocolo EVIGEN.

Con el fin de facilitar la comprensión de las interacciones entre los distintos componentes, se van a mostrar diagramas de recepción de mensajes y de procesamiento, creación y envío de mensajes de respuesta asociados a la información recibida.

4.2.2.2.1 Diagramas de secuencia de la recepción de mensajes

Considerando todos los tipos de mensajes que pueden ser recibidos así como los receptores de dichos mensajes, se van a describir las interacciones asociadas con la recepción de la CRL, la recepción de las notificaciones de sanción, la recepción de solicitudes en un nodo con rol Witness, la recepción de testimonios en nodos con rol NoEquipped, la recepción de testimonios en nodos con rol Requester y las recepciones de la información de estado.

CU-09 Parte 2.1 Recibo CRL

La secuencia de clases relacionadas con la recepción de la lista de identificadores de nodo con certificado revocado se muestra en el diagrama de secuencia posterior, Ilustración 56.

Tras la recepción, en el hilo receptor de notificaciones y CRL, de un mensaje en el que se encuentre almacenada la CRL, se envía, procesa y almacena por medio de la llamada a "mensajeRecibidoRSU" de la clase "Comunicaciones". Además, en lo referente al procesamiento y almacenamiento de la lista, indicar que, tal y como queda expuesto al comienzo del análisis (sección 3.1.1), se necesita el acceso al componente *CriptografiaNodos* para realizar la operación de "verificarFirmaCRL". La ejecución de dicha operación criptográfica queda ampliada en el diagrama de la Ilustración 74, "Operaciones asociadas a la criptografía".

Por otro lado, se envía, al sistema de visión de mensajes de la simulación, la información conveniente, relativa a la CRL recibida, mediante la función "procesarCRL" de "ProcesarMsgInterfaz". Posteriormente, se realiza el envío a través del mismo componente, *ComunicacionInterfaz*, ampliándose dicha operación en el diagrama de la Ilustración 71, "Envío mensajes desde NCTUns 5.0".

Para terminar, es importante indicar la necesidad de invocar "recibirMensajeRSU", de la clase "MsgOperations", para que el servidor se mantenga a la espera de nuevos mensajes.

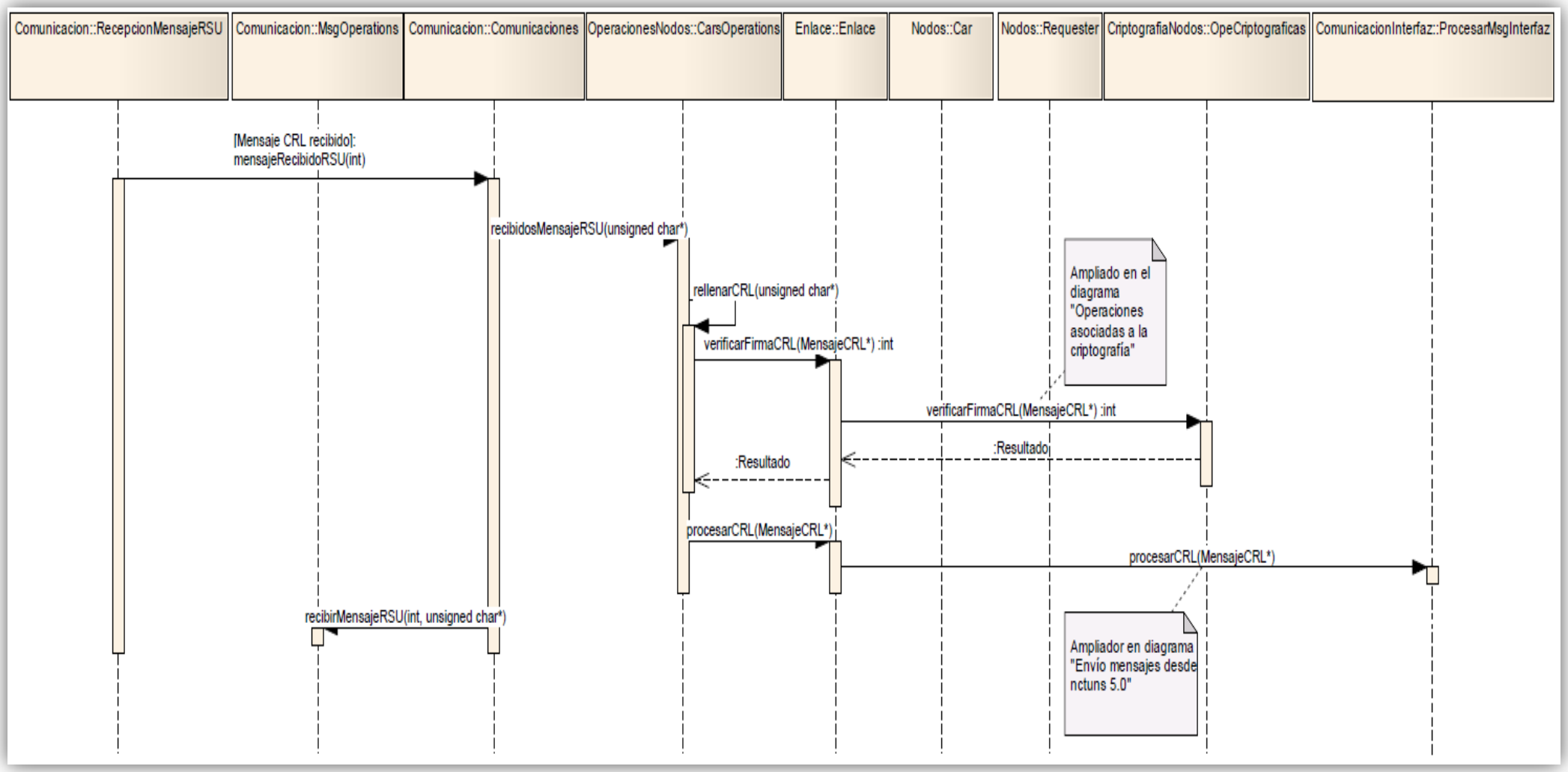


Ilustración 56: diagrama de secuencia CU-09 Parte 2.1 recibo CRL.

CU-09 Parte 2.2 Recibo notificación

La secuencia de clases relacionadas con la recepción de las notificaciones de sanción se presenta en el diagrama posterior, Ilustración 57.

Una vez recibido, en el hilo receptor de notificaciones y CRL, un mensaje correspondiente con una notificación de sanción, se envía, procesa e inicia el protocolo, comenzando por la llamada a "mensajeRecibidoRSU" de la clase "Comunicaciones".

En cuanto al procesamiento de la notificación, es imprescindible el acceso a *CriptografiaNodos* para verificar que la firma incluida en el mensaje, especificada al comienzo del análisis (sección 3.1.1), es correcta. La ejecución de dicha operación criptográfica queda ampliada en el diagrama de la Ilustración 74, "Operaciones asociadas a la criptografía".

Otra de las funciones a realizar es el envío, al sistema de visión de mensajes de la simulación, de la información conveniente, relativa a la notificación recibida, mediante la función "procesarNotificación", de la clase "ProcesarMsgInterfaz". Posteriormente, se ejecuta el envío en el mismo componente, *ComunicacionInterfaz*, ampliándose dicha operación en el diagrama de la Ilustración 71, "Envío mensajes desde NCTUns 5.0".

Por otro lado, debido a la recepción de una notificación y de acuerdo con los pasos a realizar en este protocolo, se inicia la ejecución del rol Requester a través de la invocación de la función "enviarPetición" de la clase "Car". Esto continúa en el diagrama de la Ilustración 65, "Creación y envío de solicitud".

Por último, es importante indicar la necesidad de invocar "recibirMensajeRSU", de la clase "MsgOperations", para que el servidor se mantenga a la espera de nuevos mensajes.

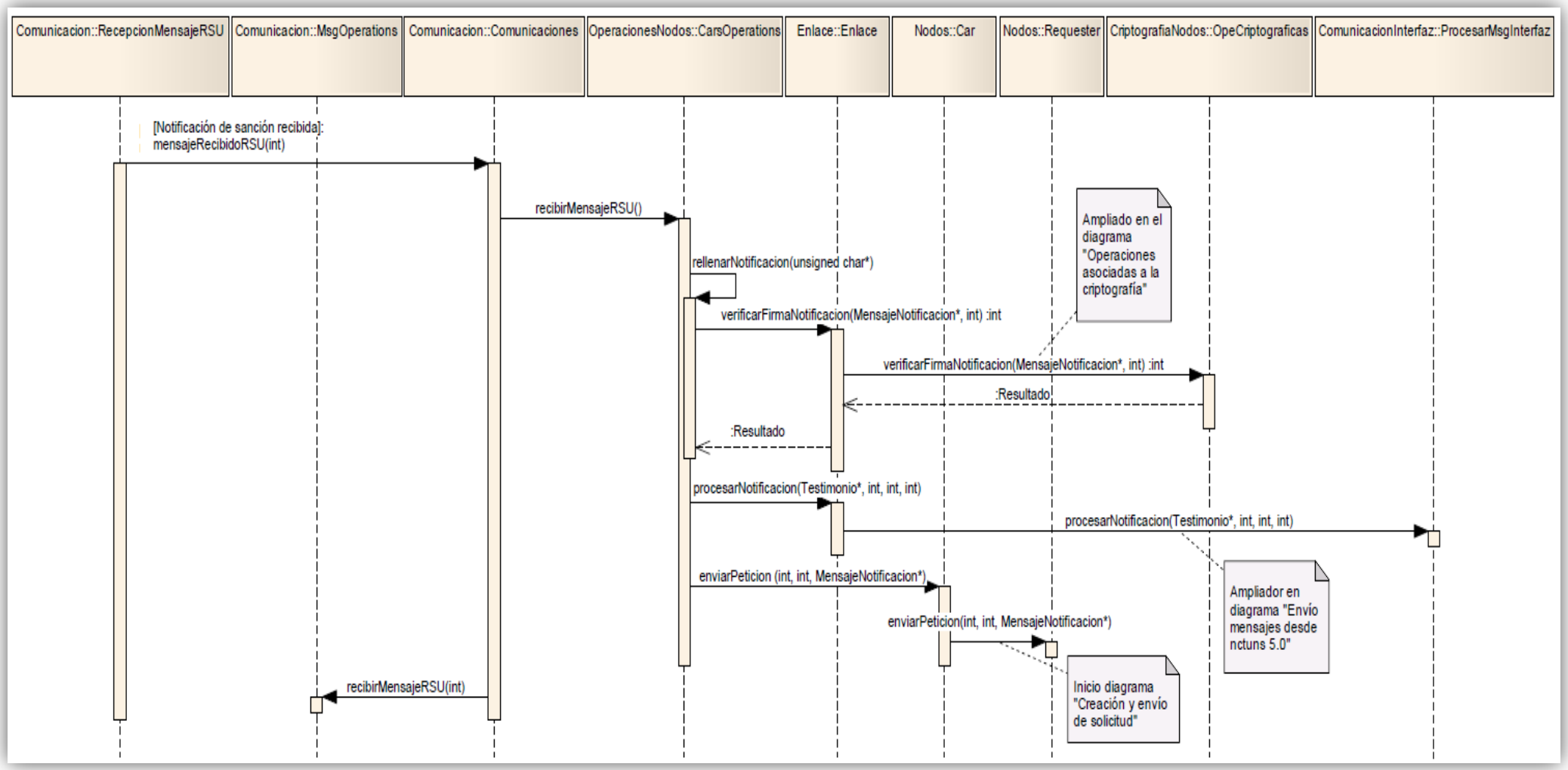


Ilustración 57: diagrama de secuencia CU-09 Parte 2.2 recibo notificación.

CU-09 Parte 2.3 Recibo solicitud en Witness

La secuencia de clases relacionadas con la recepción de solicitudes en nodos con rol Witness se muestra en el diagrama posterior, Ilustración 58.

Una vez recibido, en el hilo receptor de solicitudes procedentes de un nodo con rol Requester, un mensaje correspondiente con una solicitud, se procesa y se responde de acuerdo a los datos disponibles, lo cual será presentado en el diagrama de la Ilustración 67 y de la Ilustración 68, "Creación y envío testimonio desde Witness".

Además, se envía la información conveniente, relativa a la solicitud recibida, al sistema de visión de mensajes de la simulación. Esto se realiza haciendo uso de "procesarNotificacion", de la clase "ProcesarMsgInterfaz", para posteriormente ejecutar el envío en el mismo componente, *ComunicacionInterfaz*. La operación de envío es ampliada en el diagrama de la Ilustración 71, "Envío mensajes desde NCTUns 5.0".

Finalmente, es imprescindible establecer que el servidor se mantenga a la espera de nuevos mensajes, haciendo uso de "recibirTestigo" de la clase "MsgOperations".

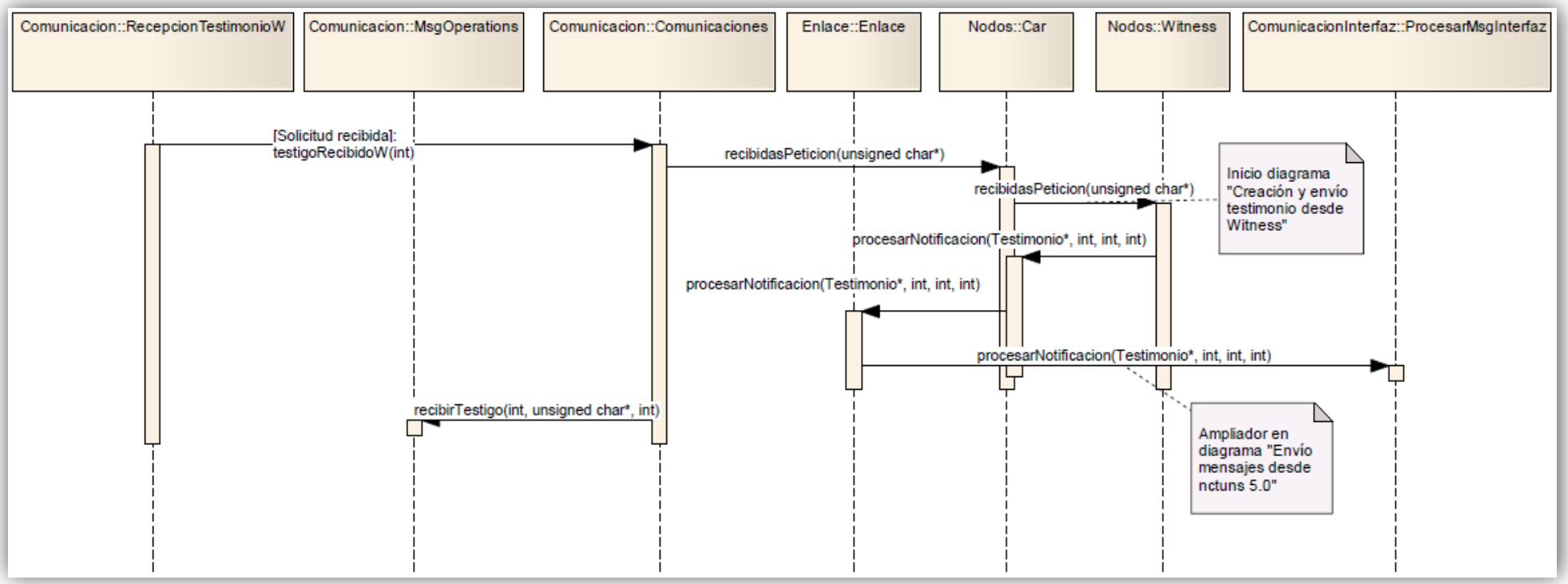


Ilustración 58: diagrama de secuencia CU-09 Parte 2.3 recibo solicitud en Witness.

CU-09 Parte 2.4 Recibo testimonio en NoEquipped

La secuencia de clases relacionadas con la recepción de testimonios en nodos con rol NoEquipped se muestra en el diagrama posterior, Ilustración 59.

Las recepciones se pueden obtener por medio del hilo receptor de testimonios procedentes de un nodo con rol Witness o mediante el hilo receptor de testimonios de nodos con rol NoEquipped o RSU.

Tras la recepción de un testimonio, por alguno de los hilos mencionados, se procesa y se crea un testimonio respuesta de acuerdo a los datos disponibles, lo cual será presentado en el diagrama de la Ilustración 69 y de la Ilustración 70, "Creación y envío testimonio desde NoEquipped".

Por otra parte, se envía la información conveniente, relativa al testimonio recibido, al sistema de visión de mensajes de la simulación. Esto se realiza haciendo uso de "procesarNotificacion", de la clase "ProcesarMsgInterfaz", para posteriormente ejecutar el envío en el mismo componente, *ComunicacionInterfaz*. La operación de envío es ampliada en el diagrama de la Ilustración 71, "Envío mensajes desde NCTUns 5.0".

Para terminar, es imprescindible establecer que el servidor se mantenga a la espera de nuevos mensajes, haciendo uso de "recibirTestigo" de la clase "MsgOperations".

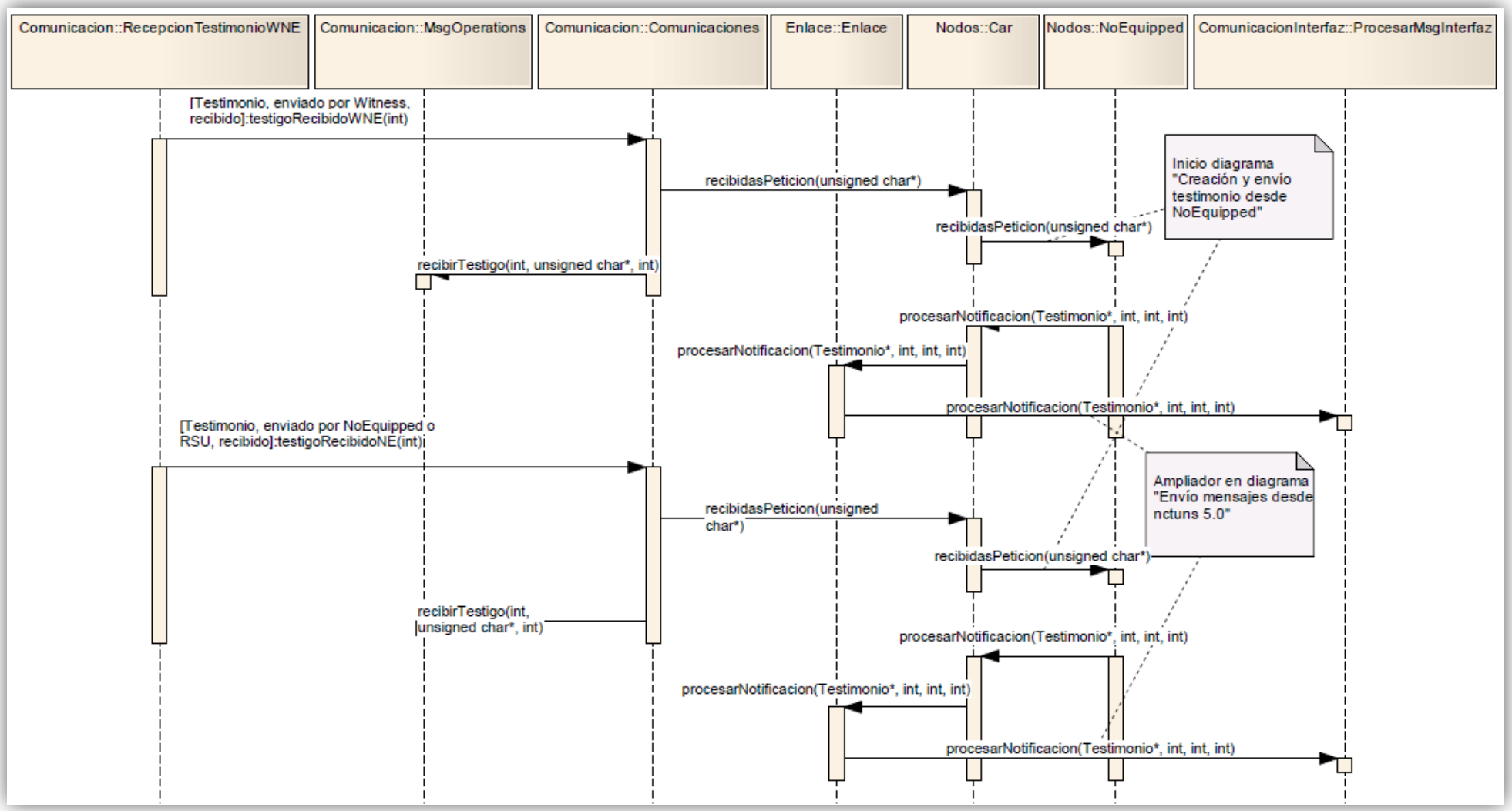


Ilustración 59: diagrama de secuencia CU-09 Parte 2.4 recibo testimonio en NoEquipped.

CU-09 Parte 2.5 Recibo testimonio en Requester

La secuencia de clases relacionadas con la recepción de testimonios en nodos con rol Requester se muestra en el diagrama posterior, Ilustración 60.

Una vez recibido, en el hilo receptor de testimonios, un mensaje correspondiente con un testimonio vinculado con una solicitud enviada previamente, se han de procesar y almacenar los datos adecuados, conforme a lo establecido al comienzo del análisis (sección 3.1.1), para la correcta creación de la evidencia asociada.

En lo referente al procesamiento, es necesaria la verificación de la corrección del mensaje recibido y, puesto que para la creación del mensaje se ha hecho uso de operaciones criptográfica, para efectuar la comprobación se necesita acceder al componente *CriptografiaNodos*, función "verificarRespuesta". La ejecución de dicha operación criptográfica queda ampliada en el diagrama de la Ilustración 74, "Operaciones asociadas a la criptografía".

Por otra parte, se envía la información conveniente, relativa al testimonio recibido, al sistema de visión de mensajes de la simulación. Esto se realiza haciendo uso de "procesarNotificacion", de la clase "ProcesarMsgInterfaz", para posteriormente ejecutar el envío en el mismo componente, *ComunicacionInterfaz*. La operación de envío es ampliada en el diagrama de la Ilustración 71, "Envío mensajes desde NCTUns 5.0".

Para concluir, es imprescindible establecer que el servidor se mantenga a la espera de nuevos mensajes, haciendo uso de "recibirTestigo" de la clase "MsgOperations".

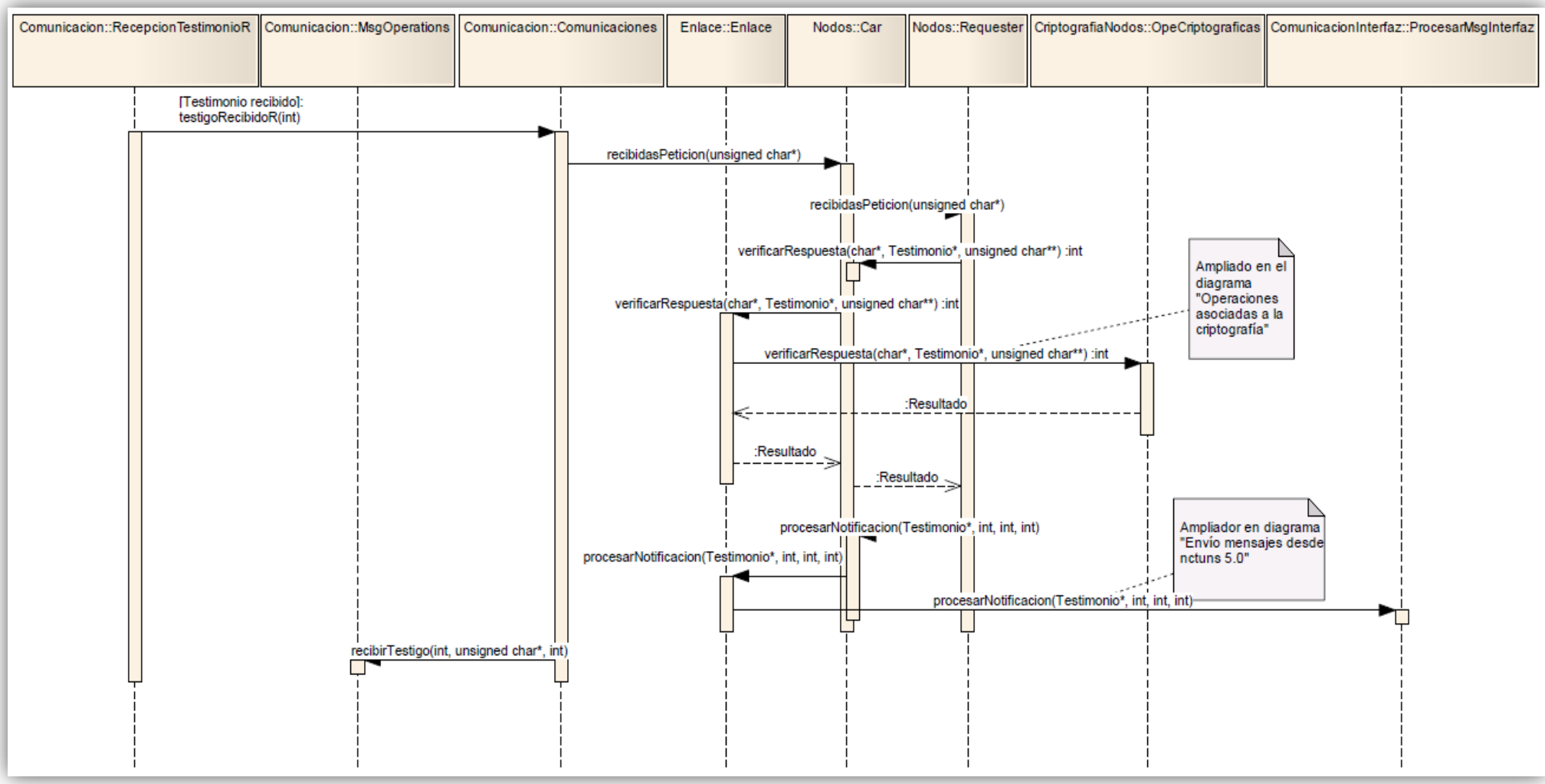


Ilustración 60: diagrama de secuencia CU-09 Parte 2.5 recibo testimonio en Requester.

CU-09 Parte 2.6 Recibo información estado nodos Requester, Witness y NoEquipped

La secuencia de clases relacionadas con la recepción de la información de estado, intercambiada en los mensajes de *beaconing* por los nodos con rol Requester, Witness y NoEquipped, se presenta en el diagrama posterior, Ilustración 61.

Una vez recibido, en el hilo receptor de la información de estado procedente de un nodo con rol Requester, Witness o NoEquipped, un mensaje correspondiente con la información de estado de un nodo, se procesa y almacena en caso de ser conveniente.

Por un lado, se garantiza que el emisor del mensaje se considera cercano, haciendo uso de "comprobarCercanía", en cuyo caso se procede al almacenamiento de la información pertinente por medio de la función "anyadirCocheAceptado".

Por otra parte, se envía la información conveniente, asociada al mensaje recibido, al sistema de visión de mensajes de la simulación. Esto se realiza haciendo uso de "procesarInfoEstado", de la clase "ProcesarMsgInterfaz", para posteriormente ejecutar el envío en el mismo componente, *ComunicacionInterfaz*. La operación de envío es ampliada en el diagrama de la Ilustración 71, "Envío mensajes desde NCTUns 5.0".

Para concluir, es imprescindible establecer que el servidor se mantenga a la espera de nuevos mensajes, haciendo uso de "recibirPropiedades" de la clase "MsgOperations".

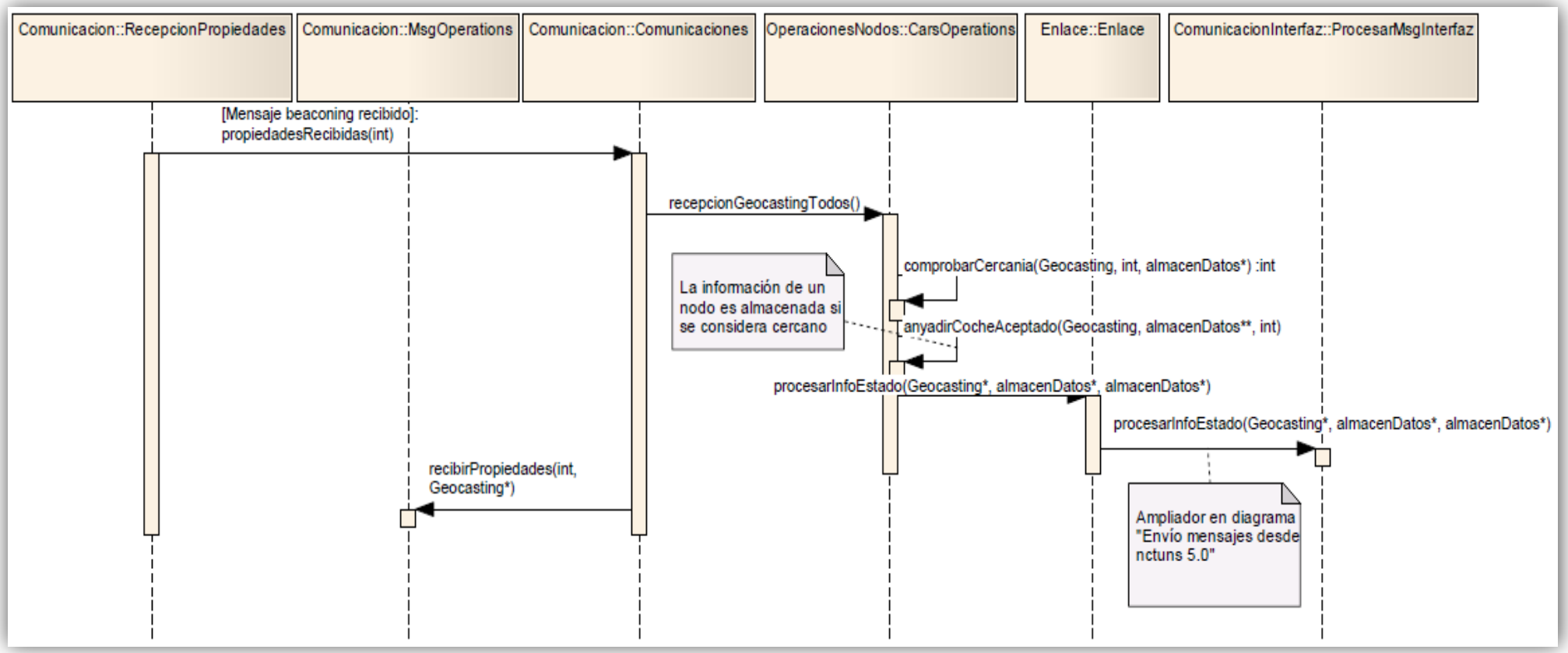


Ilustración 61: diagrama de secuencia CU-09 Parte 2.6 recibo estado nodos Requester, Witness y NoEquipped.

CU-09 Parte 2.7 Recibo información estado nodo RSU

La secuencia de clases relacionadas con la recepción de la información de estado, intercambiada en los mensajes de *beaconing*, por los nodos con rol RSU, se muestra en el diagrama posterior, Ilustración 62.

Una vez recibido, en el hilo receptor de la información de estado procedente de un nodo con rol RSU, un mensaje correspondiente con la información de estado, se procesa y almacena invocando de la función "anyadirCocheAceptado".

Finalmente, es imprescindible establecer que el servidor se mantenga a la espera de nuevos mensajes, haciendo uso de "recibirMensajeGeoRSU" de la clase "MsgOperations".

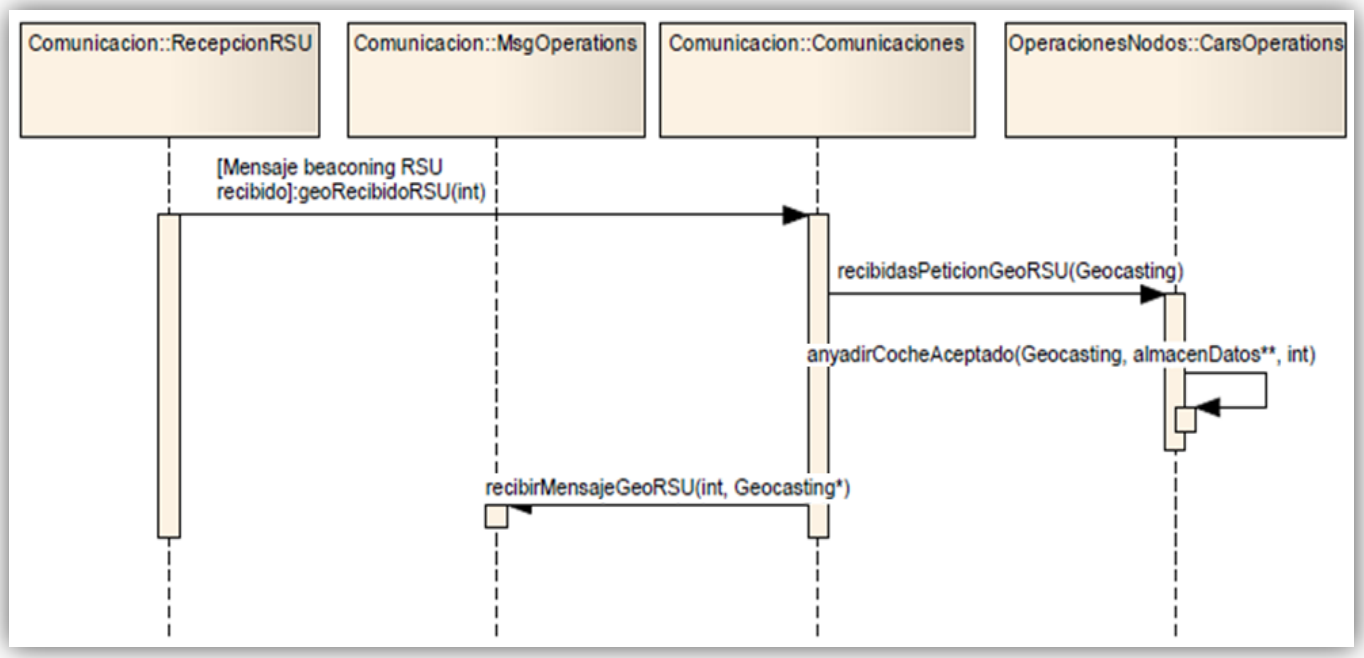


Ilustración 62: diagrama de secuencia CU-09 Parte 2.7 recibo estado nodo RSU.

4.2.2.2.2 Diagramas de secuencia del procesamiento de los mensajes recibidos y envío de mensajes de respuesta

Una vez descritas las secuencias de interacción entre los distintos componentes y clases para la recepción de cada uno de los datos del sistema, se presenta el procesamiento y la creación de los mensajes de respuesta asociados a la información recibida.

CU-09 Parte 3.1 Creación y envío de evidencia

La secuencia de clases relacionadas a la creación y al envío de una evidencia se presenta en los diagramas de secuencia posteriores, Ilustración 63 e Ilustración 64.

Cada vez que finaliza el tiempo establecido en *TIEMPO_MAX_REENVIOS* se ejecuta el hilo de comprobación de testimonios recibidos, presentado en el diseño de los

componentes Nodos, Requester, Witness y NoEquipped (sección 4.1.2.1.2), realizando una invocación a "comprobarRespuestas". En dicha función se efectúa un filtro que permite la obtención de la velocidad a incluir en la evidencia, teniendo en cuenta cada uno de los intervalos recibidos en los testimonios.

Posteriormente, haciendo uso de operaciones criptográficas, mediante la función "crearEvidencia" del componente CriptografíaNodos, se efectúa la creación de la evidencia. La ejecución de dicha operación criptográfica queda ampliada en el diagrama de la Ilustración 74, "Operaciones asociadas a la criptografía".

Por otro lado, como se muestra en la parte inferior de la Ilustración 63, se realiza el envío al sistema de visión de mensajes de la simulación de la información conveniente, asociada a la evidencia creada. Esto se efectúa haciendo uso de "procesarNotificacion", de la clase "ProcesarMsgInterfaz", para posteriormente ejecutar el envío en el mismo componente, *ComunicacionInterfaz.* La operación de envío es ampliada en el diagrama de la Ilustración 71, "Envío mensajes desde NCTUns 5.0".

Seguidamente, Ilustración 64, de todos los nodos con rol RSU se busca el más cercano y se envía la evidencia a dicho nodo, a través de las funciones "enviarTestigo" y "enviarMensaje", pertenecientes al componente *Comunicación*.

Para terminar, se finaliza el hilo de comprobación de testimonios recibidos, mediante "finHiloComprobaciones". Esta función es ejecutada puesto que al finalizar el procesamiento de una notificación de sanción no es necesario continuar realizando comprobaciones de recepción de testimonios, las cuales ya no serían de utilidad, mejorándose así el rendimiento del protocolo.

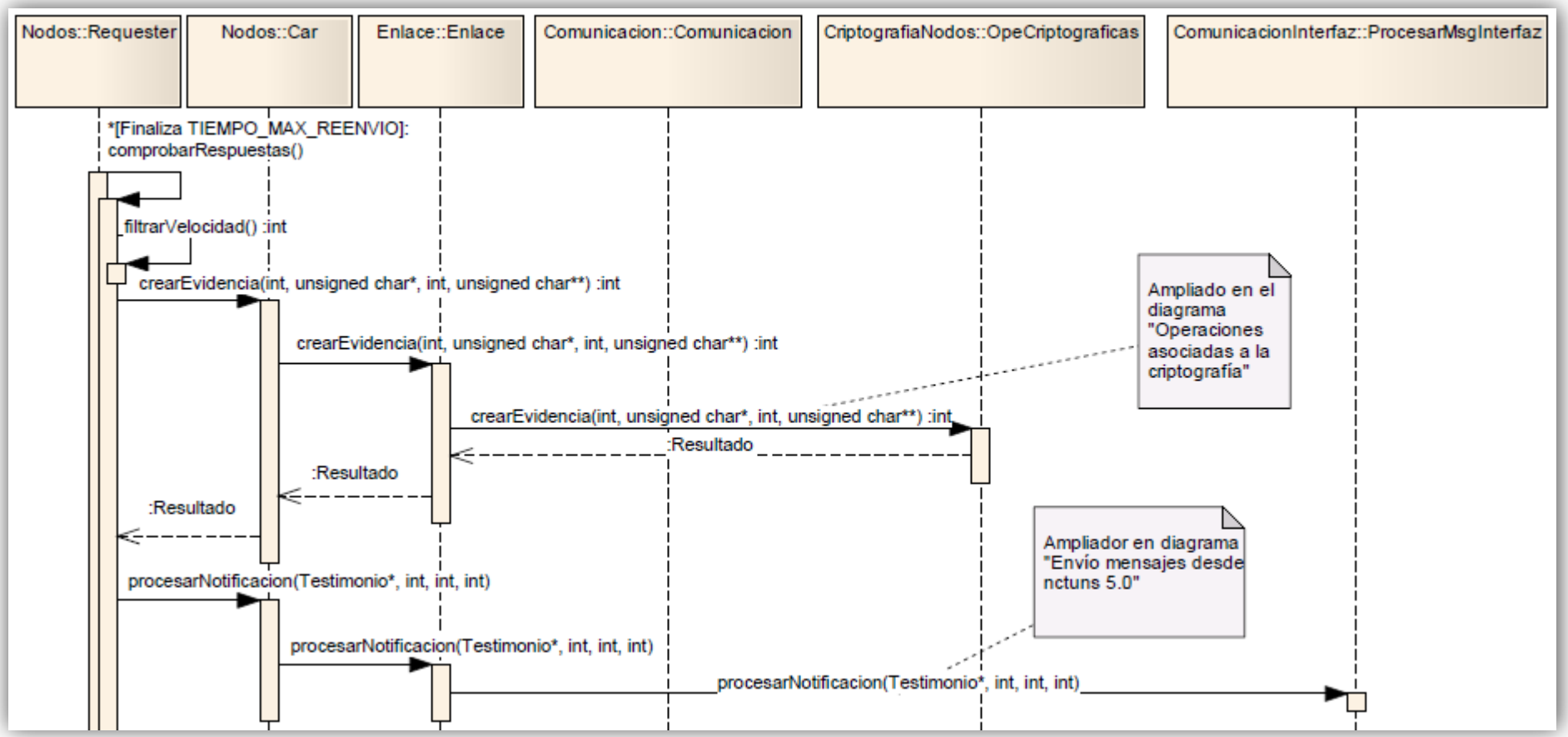


Ilustración 63: diagrama de secuencia CU-09 Parte 3.1 creación y envío de evidencia 1.

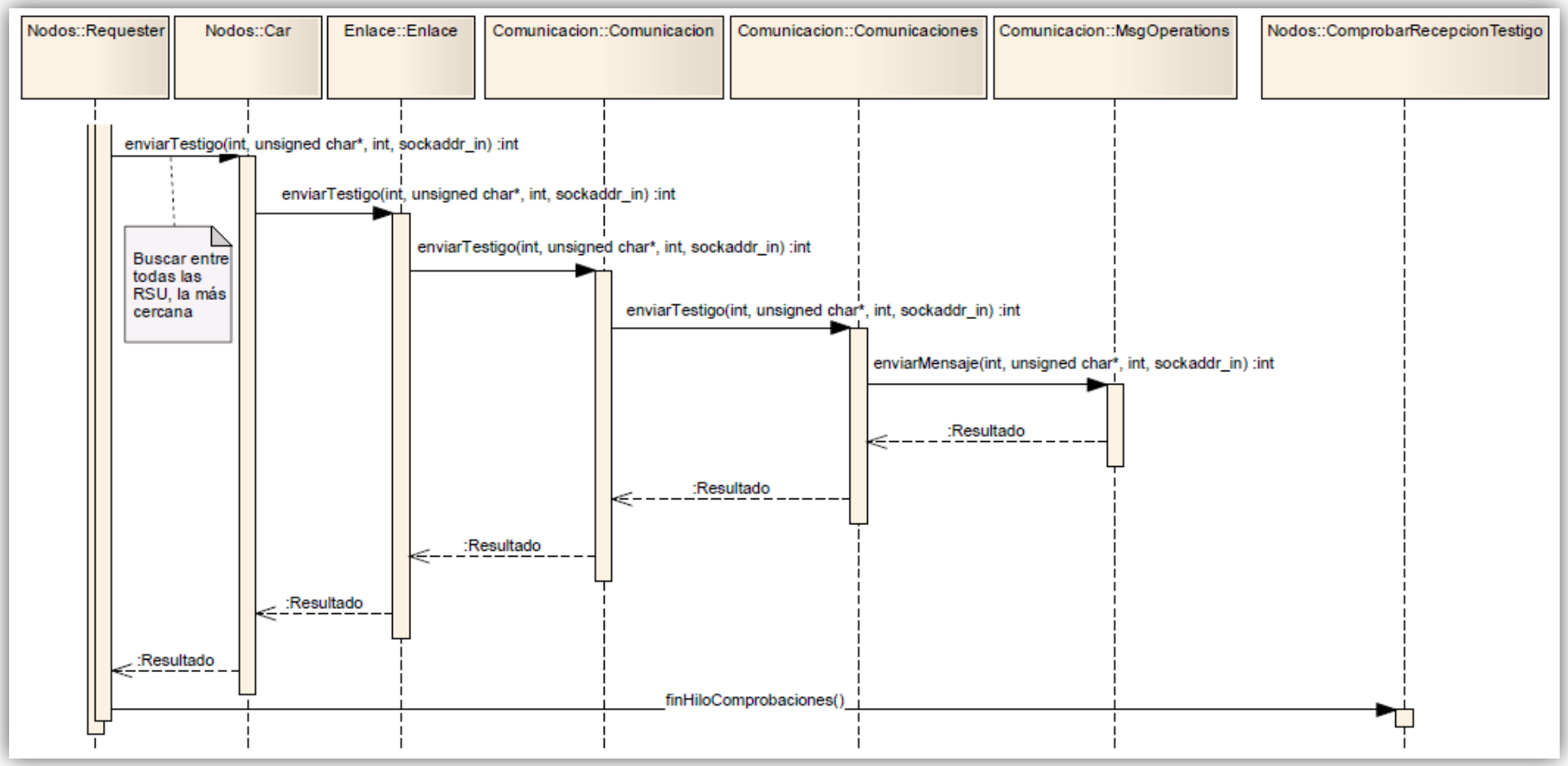


Ilustración 64: diagrama de secuencia CU-09 Parte 3.1 creación y envío de evidencia 2.

CU-09 Parte 3.2 Creación y envío de solicitud

La secuencia de clases relacionadas con la creación y el envío de una solicitud se muestra en los diagramas de secuencia posteriores, Ilustración 65 e Ilustración 66.

Tras la recepción de una notificación se arranca el hilo de comprobación de testimonios, recibidos a través de la función "crearComprobacionRecepcionTestigo", puesto que aquí se inicia el procesamiento y creación de una solicitud.

Seguidamente, utilizando las operaciones criptográficas descritas en la 3.1.1, por medio de "crearPeticion", se crea la solicitud. La ejecución de dicha operación criptográfica queda ampliada en el diagrama de la Ilustración 74, "Operaciones asociadas a la criptografía".

Posteriormente, se verifica la cantidad de envíos que se han realizado asociados a la notificación recibida, función "comprobarPeticion", de modo que si ha sido enviada una cantidad inferior a *NUMERO_MAX_REENVIOS* se permita realizar el envío/reenvío.

Por otro lado, se realiza el envío al sistema de visión de mensajes de la simulación de la información conveniente, asociada a la solicitud creada. Esto se efectúa haciendo uso de "procesarNotificacion" de la clase "ProcesarMsgInterfaz", para posteriormente ejecutar el envío en el mismo componente, *ComunicacionInterfaz*. La operación de envío es ampliada en el diagrama de la Ilustración 71, "Envío mensajes desde NCTUns 5.0".

Para terminar, considerando todos los nodos con rol Witness, previamente almacenados, se han de escoger aquellos que se consideran cercanos, función "comprobarCercanía, y realizar el envío de la solicitud a cada uno de ellos, por medio del componente *Comunicacion*, función "enviarTestigo". Este proceso es continuado en el diagrama de la Ilustración 58, "Recibo solicitud en Witness".

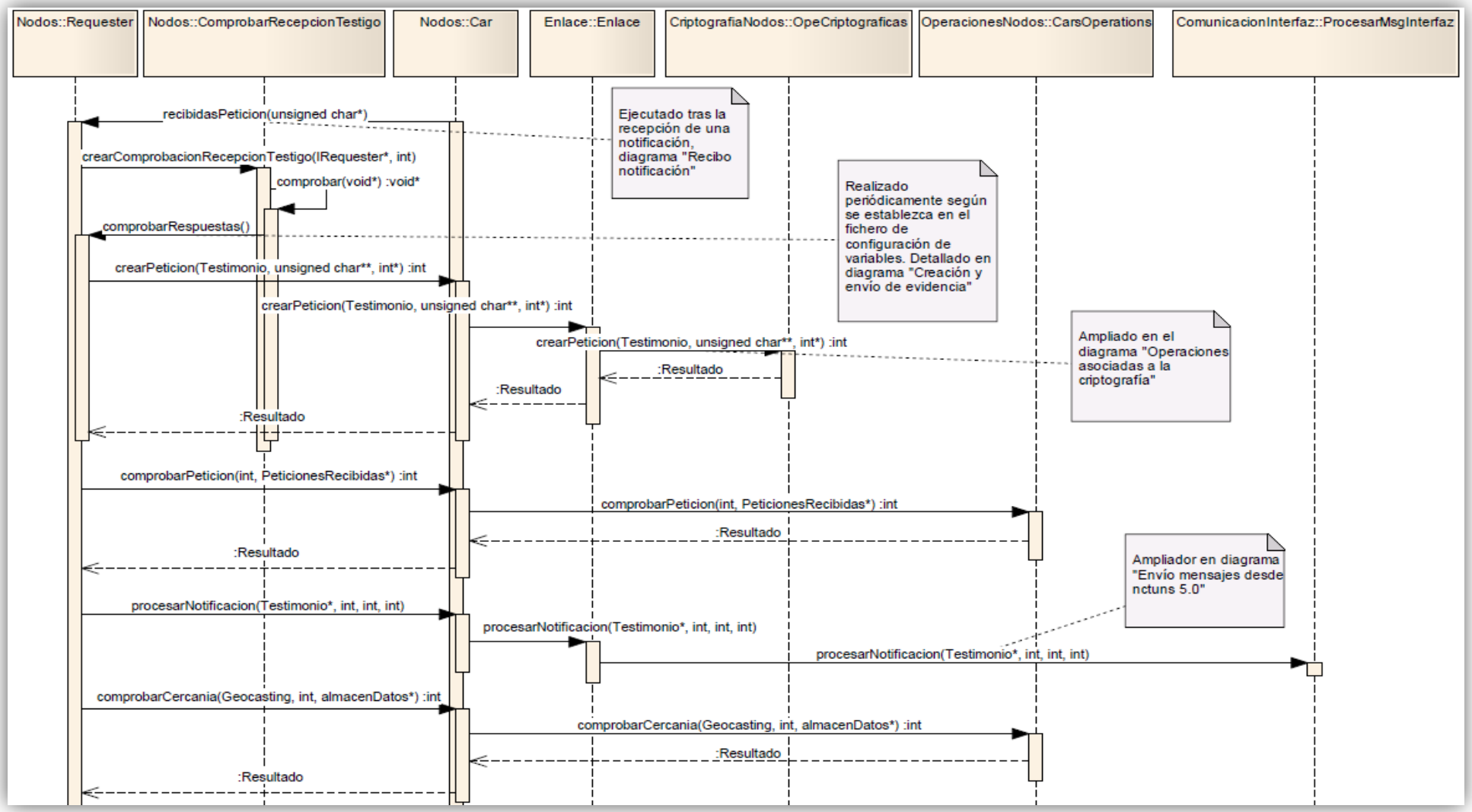


Ilustración 65: diagrama de secuencia CU-09 Parte 3.2 creación y envío de solicitud 1.

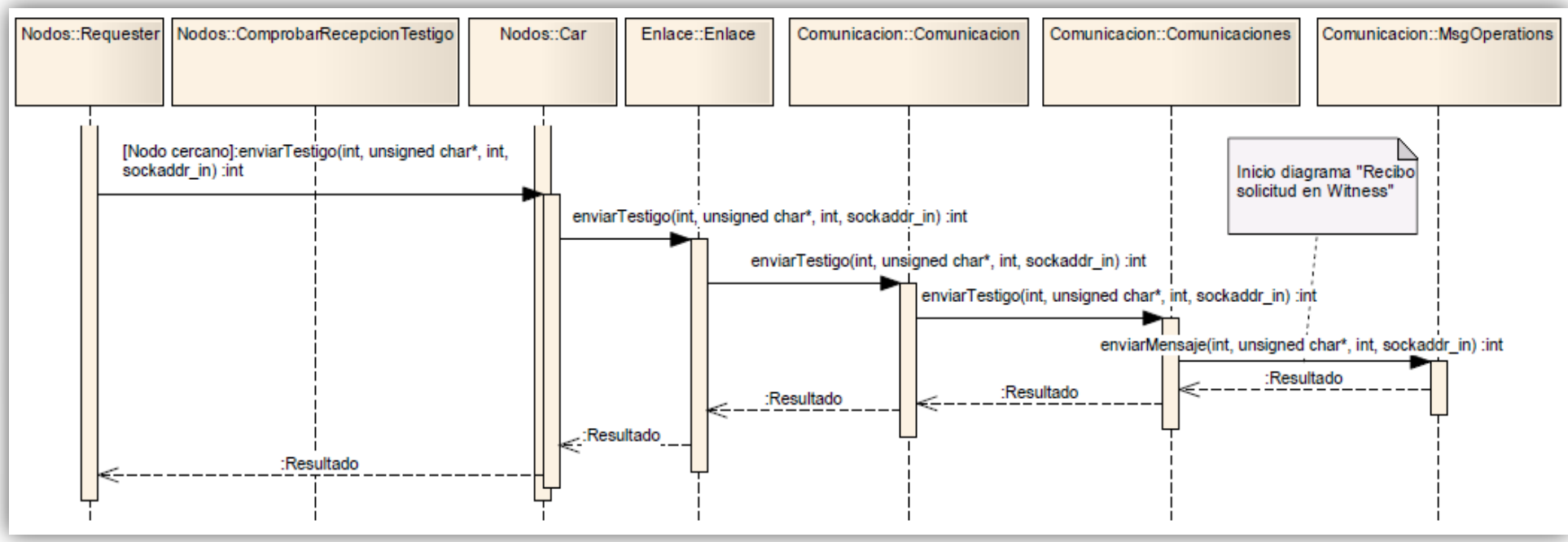


Ilustración 66: diagrama de secuencia CU-09 Parte 3.2 creación y envío de solicitud 2.

CU-09 Parte 3.3 Creación y envío de testimonio desde Witness

La secuencia de clases relacionadas con la creación y el envío de un testimonio desde un nodo Witness hacia el nodo emisor de la solicitud o hacia nodos con roles NoEquipped o RSU, lo cual es dependiente de la distancia a la que se encuentre cada uno de ellos, se muestra en los diagramas de secuencia posteriores, Ilustración 67 e Ilustración 68.

Una vez recibida una solicitud se comprueba la validez de la misma y, al igual que en otras ocasiones, se accede al componente *CriptografiaNodos*, esta vez haciendo uso de la función "verificarPeticion".

Posteriormente es necesaria la creación del intervalo a incluir en el testimonio, invocando "establecerRangoVelocidad", para obtener la respuesta a la solicitud recibida a través de la función "crearRespuesta", de nuevo situada en el componente *CriptografiaNodos*. La ejecución de ambas operaciones criptográficas queda ampliada en el diagrama de la Ilustración 74, "Operaciones asociadas a la criptografía".

Seguidamente, es conveniente constatar que el nodo emisor de la solicitud es confiable, verificándose su certificado en "comprobarPertenenciaCRL".

Por otro lado, se envía al sistema de visión de mensajes de la simulación de la información conveniente, asociada a la solicitud recibida. Esto se efectúa haciendo uso de "procesarNotificacion", de la clase "ProcesarMsgInterfaz", para posteriormente ejecutar el envío en el mismo componente, *ComunicacionInterfaz*. La operación de envío es ampliada en el diagrama de la Ilustración 71, "Envío mensajes desde NCTUns 5.0".

Después, teniendo en cuenta la información almacenada relacionada con los nodos participantes en la simulación, se verifica si el Requester solicitante se encuentra lo suficientemente cerca para realizar el envío, puesto que de lo contrario habría que obtener los nodos NoEquipped considerados cercanos y realizar el envío a cada uno de ellos y, en caso de no disponer de nodos NoEquipped próximos, el envío se realizaría a todas las RSU almacenas. Todo esto se realiza por medio de la función "comprobarCercanía" y las funciones "enviarTestigo" y "enviarMensaje" del componente *Comunicacion*. Este proceso es continuado en el diagrama de la Ilustración 60, "Recibo testimonio en Requester", o en el diagrama de la Ilustración 59, "Recibo testimonio en NoEquipped".

Finalmente, tras finalizar las funciones de procesamiento de un mensaje recibido, se verifica la existencia de otro en la lista de mensajes que han de ser procesador por haber sido recibidos cuando había uno en proceso. Por tanto, por medio de la función "comprobarListaMensajesAlmacenados" se verifica la existencia de un mensaje, en cuyo caso se realizará, nuevamente, cada una de las funciones presentadas en la Ilustración 67 y en la Ilustración 68, "Creación y envío de testimonio desde Witness 1" y "Creación y envío de testimonio desde Witness 2", respectivamente.

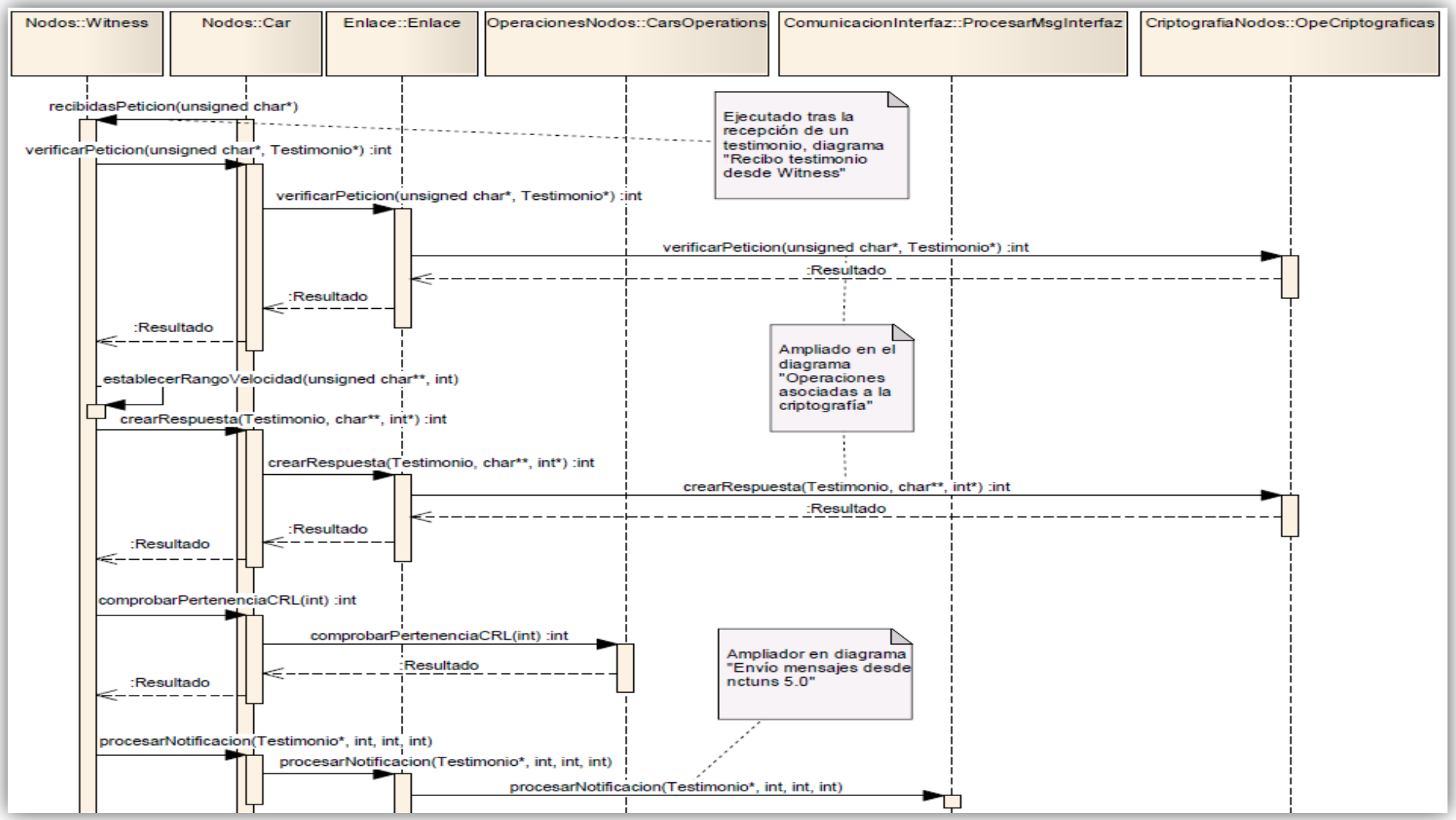


Ilustración 67: diagrama de secuencia CU-09 Parte 3.3 creación y envío de testimonio desde Witness 1.

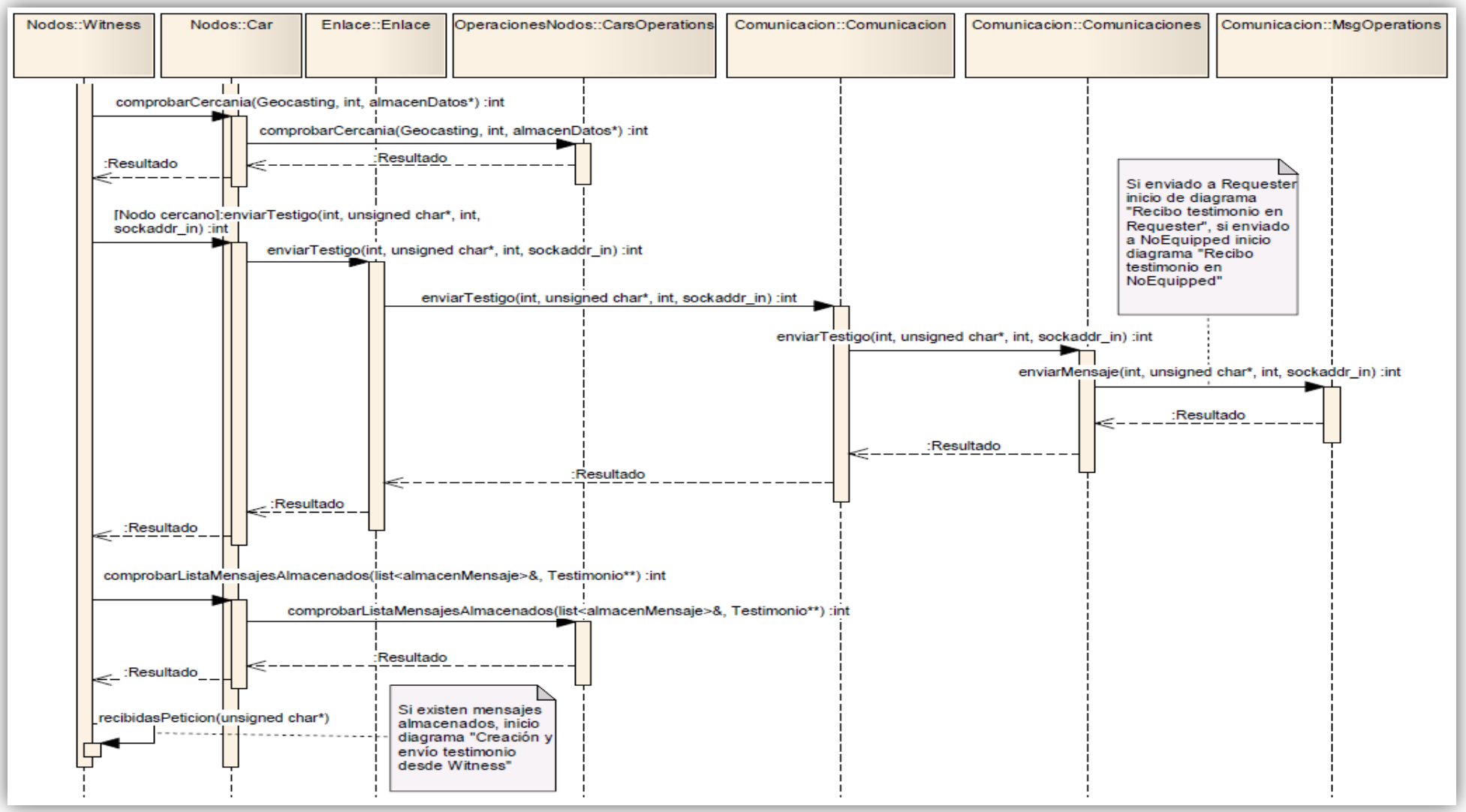


Ilustración 68: diagrama de secuencia CU-09 Parte 3.3 creación y envío de testimonio desde Witness 2.

CU-09 Parte 3.4 Creación y envío de testimonio desde NoEquipped

La secuencia de clases relacionadas con la creación y el envío de un testimonio desde un nodo NoEquipped hacia el nodo emisor de la solicitud o nodos con roles NoEquipped o RSU, lo cual es dependiente de la distancia a la que se encuentre cada uno de ellos, se muestra en los diagramas de secuencia posteriores, Ilustración 69 e Ilustración 70.

Tras la recepción de un testimonio y la modificación de los campos oportunos, tal y como se ha expuesto en la sección 3.1.1, se verifica que la cantidad de recepciones de dicho testimonio sea inferior a NUMERO_MAX_REENVIOS, siendo descartado en caso contrario, y realizándose esta operación a través de la función "comprobarPeticion".

A partir de este punto se realizan las mismas operaciones de envío a la interfaz de la información conveniente asociada al testimonio recibido, de comprobación de cercanía, de envío al nodo correspondiente y de comprobación de la existencia de mensajes almacenados, que fueron descritas en la "Creación y envío de testimonio desde Witness".

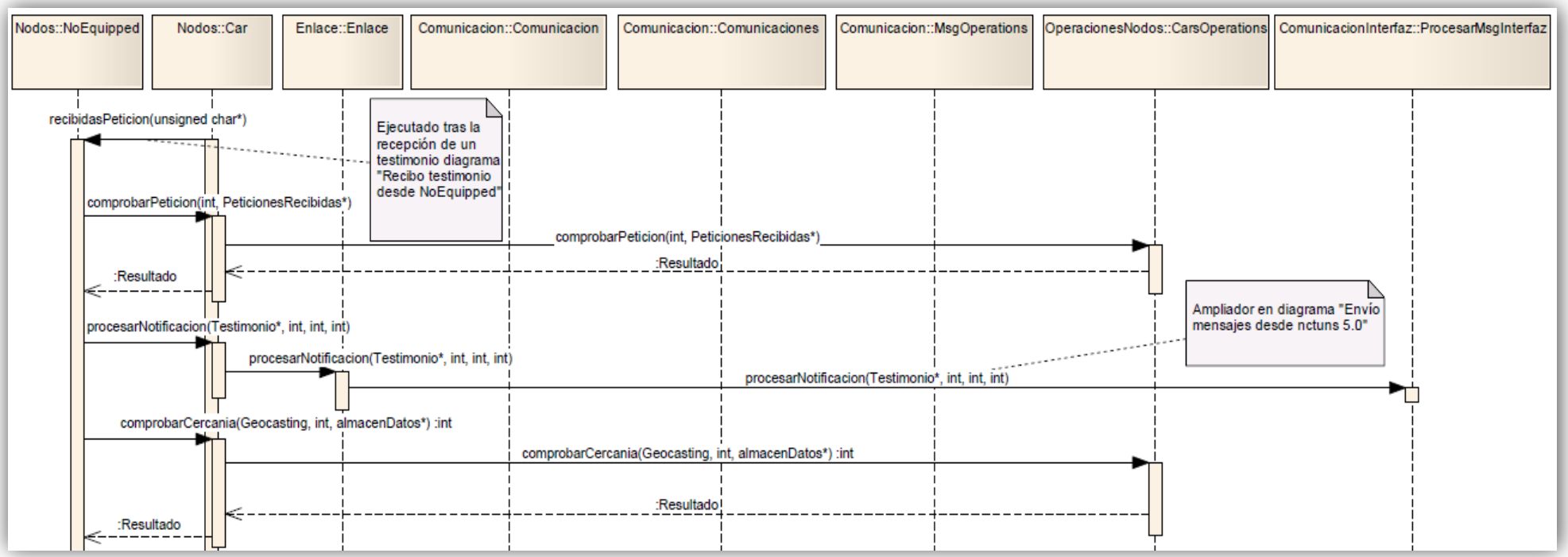


Ilustración 69: diagrama de secuencia CU-09 Parte 3.4 creación y envío de testimonio desde NoEquipped 1.

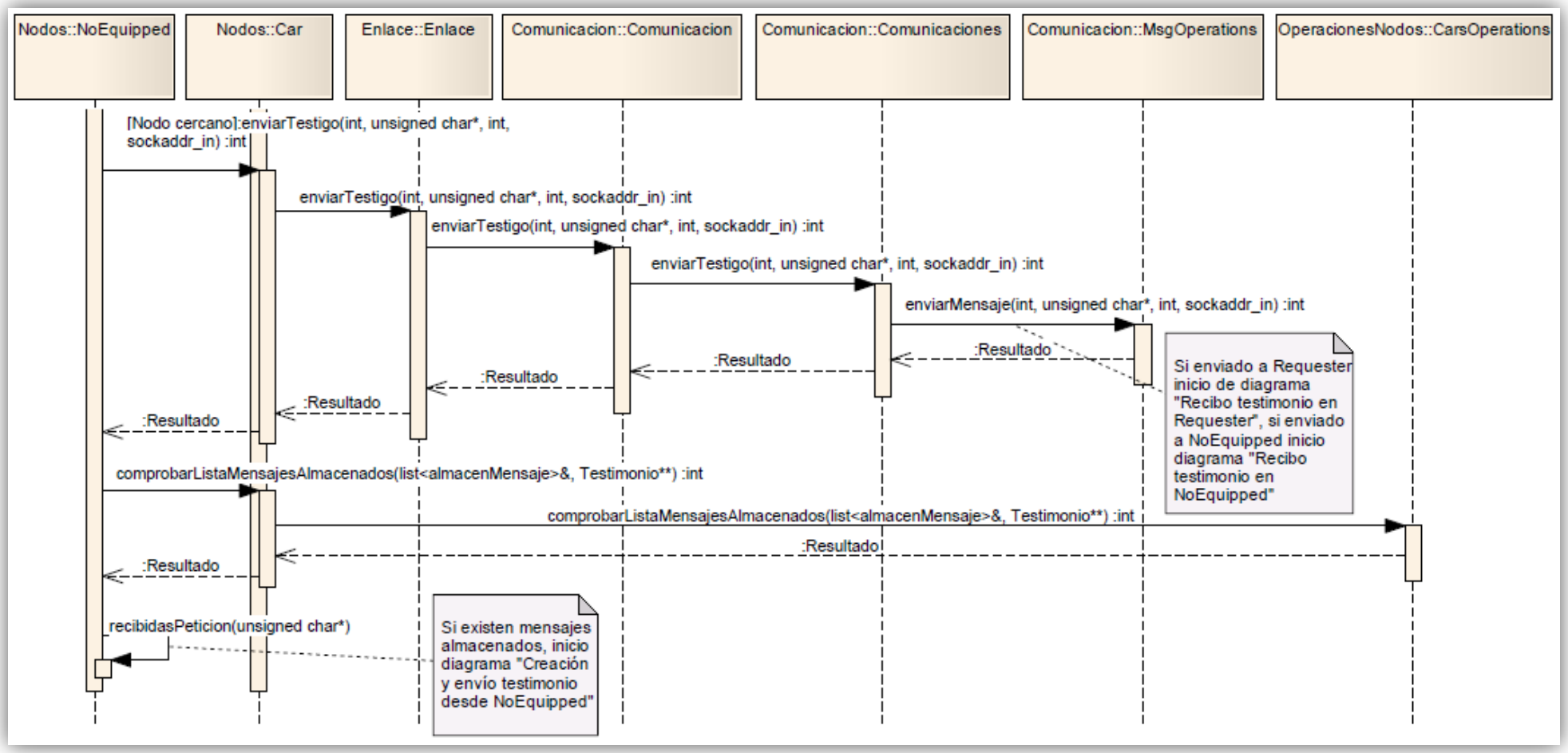


Ilustración 70: diagrama de secuencia CU-09 Parte 3.4 creación y envío de testimonio desde NoEquipped 2.

4.2.2.3 Diagrama de secuencia del envío de datos al sistema de visión de mensajes de la simulación

En esta sección se va a realizar la descripción de las clases utilizadas para el envío de mensajes y nodos participantes en la simulación al sistema de visión de mensajes de la simulación.

CU-09 Parte 4.1 Envío mensajes desde NCTUns 5.0

La secuencia de las interacciones producidas entre clases para enviar los mensajes utilizados en el protocolo al sistema de visión de mensajes de la simulación se presenta en el siguiente diagrama, Ilustración 71.

Para conseguir este propósito se requiere la participación del componente *ComunicacionInterfaz*, de modo que a partir de los nodos participantes en la simulación se efectúen llamadas a las funciones "procesarCRL", "procesarNotificacion", "procesarInfoEstado" y "procesarInfoEstadoRSU". En la implementación de cada una de las funciones indicadas se invoca la función adecuada de la clase "EnviarInterfaz" para que toda la información sea recibida correctamente en el sistema de visión de mensajes de la simulación.

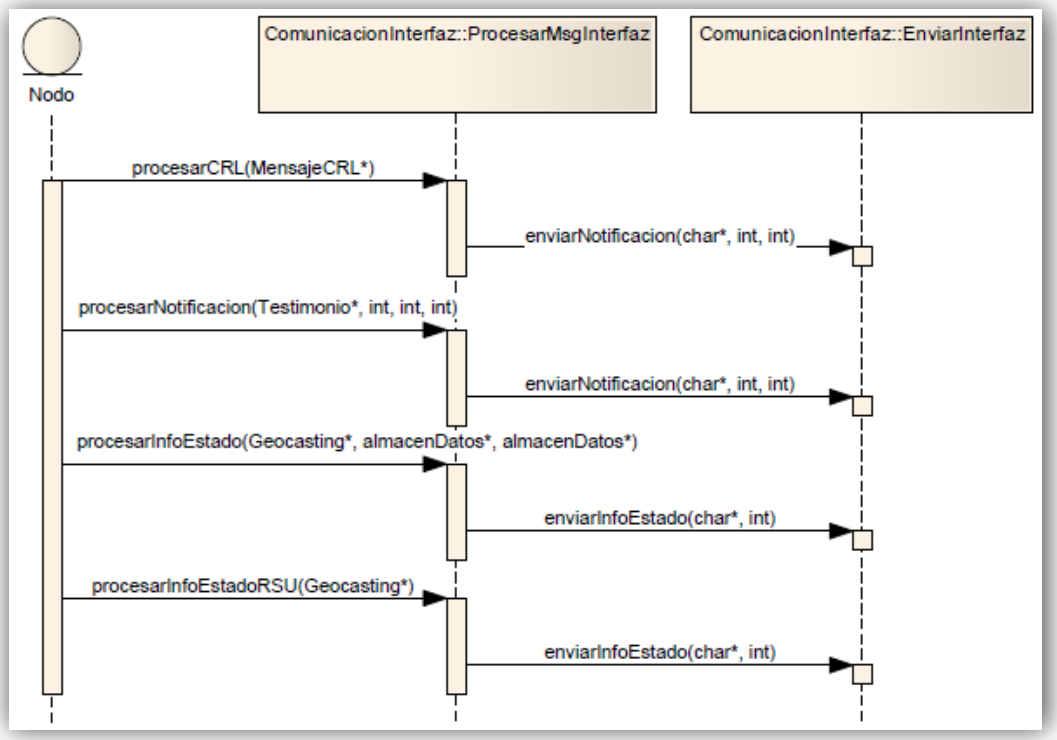


Ilustración 71: diagrama de secuencia CU-09 Parte 4.1 envío de mensajes desde NCTUns 5.0.

CU-09 Parte 4.2 Envío nodos participantes desde NCTUns 5.0

La secuencia de funciones ejecutas para el envío de los nodos participantes en la simulación al sistema de visión de mensajes de la simulación se muestra en el siguiente diagrama, Ilustración 72.

Posteriormente a todas las operaciones realizadas en los componentes del simulador NCTUns 5.0 (1), se produce una llamada a "read_trafficGen", función en la que se han de realizar la obtención, el procesamiento y el envío de los nodos participantes.

Para comenzar, se realiza la lectura del puerto al que se debe enviar la lista de nodos, función "leerPuertoInterfaz". Después se realiza el procesamiento de dicha lista, función "procesarInfoNodos" y, finalmente, se envía al sistema de visión de mensajes de la simulación, función "enviarInfoEstado".

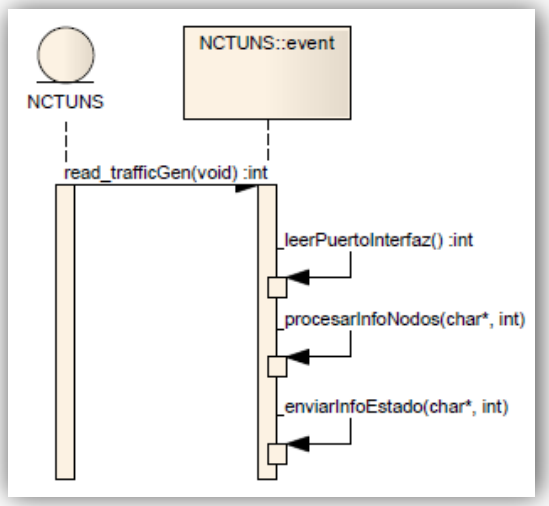


Ilustración 72: diagrama de secuencia CU-09 Parte 4.2 envío nodos participantes desde NCTUns 5.0.

4.2.2.4 Diagrama de secuencia de operaciones de soporte

En esta sección se procede a la descripción de las interacciones producidas en el establecimiento de la configuración y en la utilización de las operaciones criptográficas.

CU-09 Parte 5.1 Establecimiento de la configuración

La secuencia de las interacciones producidas entre clases para realizar el establecimiento de la configuración de puertos, variables y rutas, se muestra en el siguiente diagrama de secuencia, Ilustración 73.

Esta funcionalidad consta de dos partes diferenciadas. Por un lado, se realiza la lectura tanto de los puertos como de las variables de configuración, puesto que ambos tipos de datos son necesarios simultáneamente a lo largo de la ejecución del protocolo. Sin embargo, por otro lado se sitúa la lectura de las rutas en las que se localizan los

certificados y la clave privada ya que únicamente son necesarios en la realización de operaciones criptográficas.

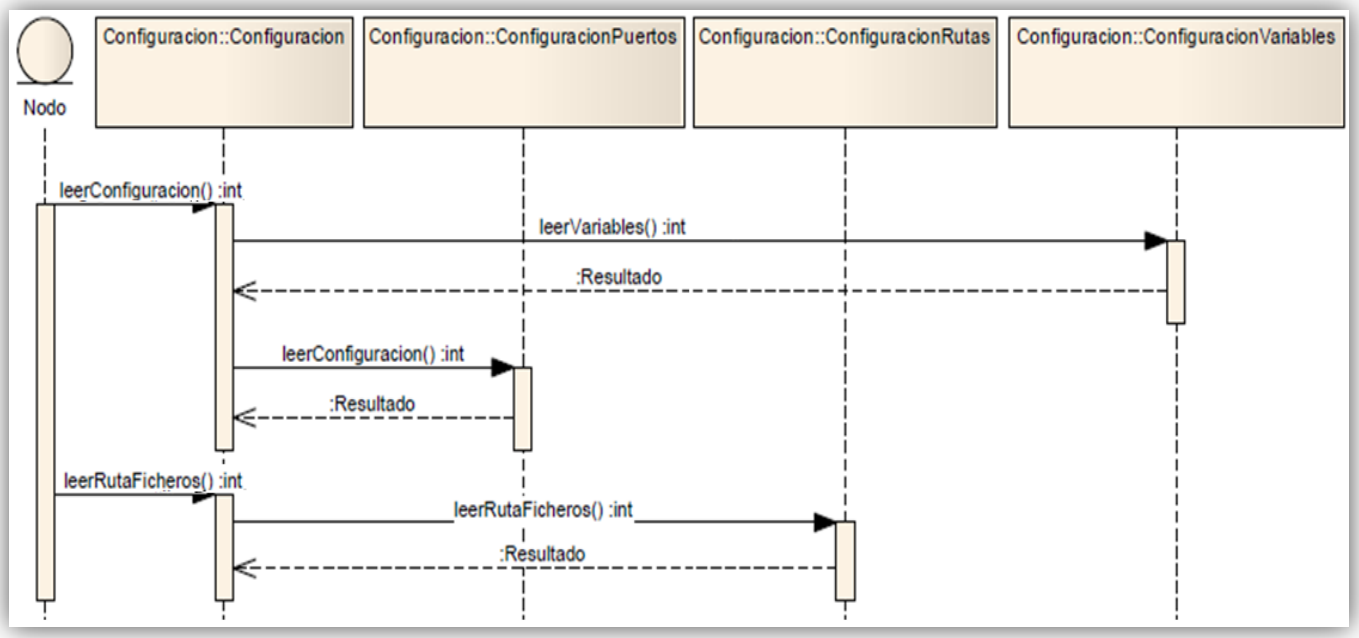


Ilustración 73: diagrama de secuencia CU-09 Parte 5.1 establecimiento de la configuración.

CU-09 Parte 5.2 Operaciones asociadas a la criptografía

La secuencia de las interacciones producidas entre clases para realizar la creación o verificación de mensajes que requieren el uso de operaciones criptográficas se presenta en el siguiente diagrama de secuencia, Ilustración 74.

Para comenzar, es imprescindible cargar las rutas de los certificados y la clave privada, lo cual se presenta ampliado en el diagrama de la Ilustración 73, "Establecimiento de la configuración".

Por otra parte, se muestran todas las invocaciones que se realizan por los distintos nodos participantes en la simulación para realizar tanto la creación de mensajes en los que se incluyen operaciones criptográficas como para realizar comprobaciones de mensajes en los que se hayan incluido dicho tipo de operaciones.

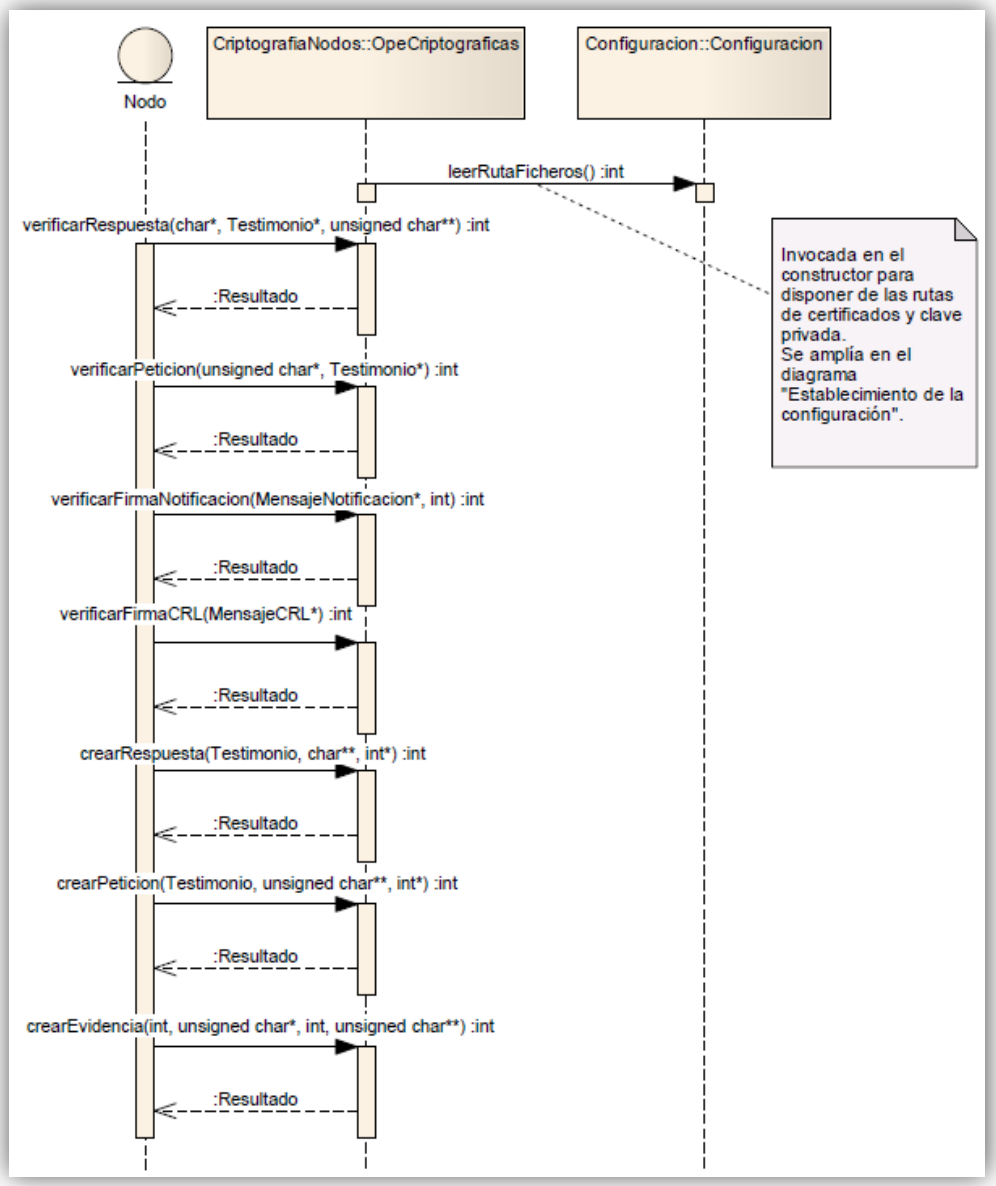


Ilustración 74: diagrama de secuencia CU-09 Parte 5.2 operaciones asociadas a la criptografía.

5 Implementación

Una vez analizado y diseñado el sistema se implementan cada una de las funcionalidades especificadas. Tras ello, se señalan las decisiones tomadas para efectuar la implementación (sección 5.1) y se presentan los resultados obtenidos tras la realización de las pruebas de aceptación (sección 5.2).

5.1 Decisiones de implementación

La implementación de los objetivos del proyecto, integración del protocolo EVIGEN en los modelos de comportamiento establecidos en el simulador NCTUns 5.0 (1) y visualización de los mensajes intercambiados en el mismo, aun habiendo realizado un preciso análisis y diseño, requieren la decisión de determinados aspectos relativos a la implementación, los cuales son presentados en los párrafos posteriores.

Certificados y claves privadas

La primera de las decisiones se basa en la consideración de que en los mensajes intercambiados se utilizan operaciones criptográficas en las que se incluye la utilización de certificados y claves privadas, de modo que es necesaria la elección de la herramienta de creación de los mismos y de la cantidad requerida.

Por un lado, los certificados y las claves privadas han sido creados mediante OpenSSL (19) por ser una aplicación potente, muy utilizada y que comprende los requisitos necesarios para el propósito a conseguir.

Por otro lado, en cuanto a la cantidad realizada, se podría haber decidido crear un certificado y una clave privada por cada nodo, de modo que se necesitasen tantos certificados y claves como nodos participantes en la simulación. Sin embargo, por simplicidad, se ha creado un mismo certificado y clave privada para todos los nodos con rol Requester, Witness y NoEquipped.

Ésta decisión se ha tomado teniendo en cuenta que la elección de un certificado u otro no afectaría al rendimiento ni al propósito del protocolo y, por ello, se piensa que es la opción más acertada, a la vez que la opción más sencilla.

Sockets UDP

Otra de las decisiones a tomar, teniendo en cuenta la utilización de sockets UDP, es la cantidad de datagramas UDP que se han de enviar por cada mensaje, ya que considerando que es un protocolo en el que la entrega no es ordenada, en caso de utilizar más de un datagrama en cada envío, habría que tener en cuenta los posibles desordenamientos de los datos.

Para tomar esta decisión es imprescindible realizar un estudio de los tamaños de los mensajes intercambiados en el protocolo. Analizados los mensajes, el único que puede producir problemas, debido a la variabilidad de su tamaño, se corresponde con la

evidencia creada tras la recepción de los distintos testimonios, presentada al inicio del análisis, en la sección 3.1.1.

$$\underline{\underline{ticket}} = E_{KpubCA}(Cert_B + S_{KpvB}(claveAES) + claveAES)$$

$$\underline{\underline{condTesti}} = E_{KAES}(v_A)$$

$$\underline{ticketsCondTesti} = \underline{\underline{ticket1}} + \underline{\underline{condTesti1}}(+\underline{\underline{ticket2}} + \underline{\underline{condTesti2}} + ... + \underline{\underline{ticketN}} + \underline{\underline{condTestiN}})$$
$$Evidencia = Cert_A + S_{KpvA}(v_F) + v_F + \underline{ticketsCondTesti}$$

Seguidamente, indicar que los tamaños obtenidos empíricamente para los siguientes campos son:

Tamaños (en bytes)					
Certificado	Clave	Firma	Intervalo velocidad	Cifrado con clave	Resumen
827	45	256	8	44	20

Tabla 22: tamaño campos.

Realizados los cálculos, se indica que el tamaño de este mensaje es de (1091 + cadenaTicketsCondTesti) bytes, siendo el tamaño aproximado de un ticket de 1128 bytes y de un condTesti de 256 bytes.

Por tanto, dado que en un paquete UDP pueden enviarse un máximo de 65535 bytes, se ofrece la posibilidad de adjuntar, aproximadamente, un total de 46 tickets y condTesti, lo cual se considera una cantidad lo suficientemente razonable como para establecer que cada mensaje será enviado en un único datagrama UDP.

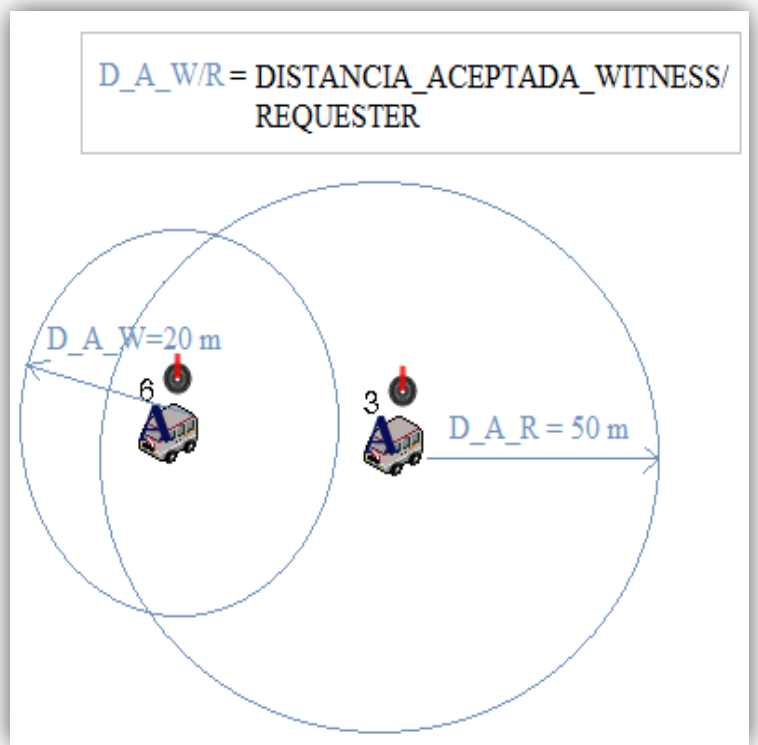
Implementación del intercambio de mensajes

Es imprescindible indicar que la implementación sobre la que el creador de NCTUns desarrolló el intercambio de mensajes, se basa en establecer una potencia de antena configurable para cada nodo participante en la simulación y, de este modo, aumentar o disminuir el radio de envío y recepción de mensajes. La potencia de la antena puede ser modificada mediante la propiedad "*Transmission Power*" de cada uno de los nodos.

Sin embargo, a la vista del análisis y del diseño presentado, se puede observar la propia implementación realizada para efectuar el envío y la recepción de los mensajes, de modo que se toma como base la posición de cada uno de los nodos, emisor y receptor, y se determinada, de acuerdo a las variables *DISTANCIA_ACEPTADA_REQUEESTER*, *DISTANCIA_ACEPTADA_WITNESS* y *DISTANCIA_ACEPTADA_NOEQUIPPED*, la aceptación o el rechazo de los mensajes. Hay que subrayar que en los mensajes de *beaconing*, al enviarse por inundación, la comprobación sobre la cercanía o lejanía de un determinado nodo se realiza en el receptor, descartando o aceptando el mensaje recibido. Sin embargo, en el resto de mensajes, con el fin de disminuir el tráfico de los

mismos, es el nodo emisor el responsable de efectuar la comprobación y realizar los envíos únicamente a los nodos considerados cercanos.

En la siguiente imagen, Ilustración 75, se presenta un ejemplo para comprender la implementación realizada.



Si mensaje *beaconing*:
- Nodo3 envía mensaje a nodo6
- Nodo6 recibe y descarta mensaje de nodo3
- Nodo6 envía mensaje a nodo3
- Nodo3 recibe y procesa mensaje de nodo6

Si no mensaje *beaconing*:
- Nodo3 sólo recibe mensajes de nodo6
- Nodo6 sólo envía mensajes a nodo3

Ilustración 75: implementación intercambio de mensajes.

El motivo de realizar la implementación de algo que estaba desarrollado previamente, ha sido el desconocimiento. Fue un proceso laborioso y que requirió mucho esfuerzo el concluir que el creador del simulador NCTUns, por medio del parámetro "*Transmission Power*", especificaba la potencia de las antenas y, con ello, aumentaba o disminuía el radio en el que se aceptaban o rechazaban mensajes, implementando de este modo el intercambio de los mismos.

Teniendo en cuenta lo comentado y dado que el hallazgo se efectuó con posterioridad al análisis, se decidió realizar el envío de los mensajes de acuerdo con el planteamiento inicialmente expuesto en este proyecto, estableciendo "*Transmission Power*" a un valor elevado (i.e. 15000) y proporcionando libertad para desarrollar la implementación. No obstante, para realizar este procedimiento tal y como lo proporciona el simulador, únicamente sería necesaria la eliminación de la llamada a la función "comprobarCercanía", presentada en el diseño del componente *Operaciones* (sección 4.1.2.1.4), en los lugares en los que se haga uso de ella y establecer el valor de "*Transmission Power*" a la cantidad deseada.

Envío de los nodos participantes en la simulación al sistema de visión de mensajes

La implementación del envío, al sistema de visión de mensajes de la simulación, de los nodos participantes en la simulación, se puede llevar a cabo mediante distintas técnicas.

Primeramente, hay que tener presente que es en la clase "event" donde se realizan modificaciones para conseguir este propósito, lo cual ha sido especificado en la sección 4.1.2.2.

Por un lado, se podría crear la lista de nodos en la clase "event" y, posteriormente, hacer llamadas a clases ya implementadas como son "Socket", "Comunicacion", etc., y realizar el envío adecuado.

Por otro lado, se podría realizar tanto la creación de la lista como el envío, en la propia clase "event", sin necesidad de acceder a otro tipo de clases.

Considerando que la realización de llamadas a clases implementadas implicaría el establecimiento de dependencias entre el protocolo desarrollado y uno de los componentes creados para la ejecución del simulador NCTUns 5.0 (1), se piensa que es más acertada la inclusión de todo el código, tanto de procesamiento como de envío, en esta misma clase.

5.2 Resultado de las pruebas de aceptación

Los resultados obtenidos tras la realización de las pruebas de aceptación se presentan en la Tabla 23.

RESULTADOS PRUEBAS DE ACEPTACIÓN		
Id	**Ver.**	**Resultado**
P-01	1.0	Superada
P-02	1.0	Superada
P-03	1.0	Superada
P-04	1.0	Superada
P-05	1.0	Superada
P-06	1.0	Superada
P-07	1.0	Superada
P-08	1.0	Superada
P-09	1.0	Superada
P-10	1.0	Superada
P-11	1.0	Superada
P-12	1.0	Superada
P-13	1.0	Superada
P-14	1.0	Superada
P-15	1.0	Superada
P-16	1.0	Superada
P-17	1.0	Superada
P-18	1.0	Superada
P-19	1.0	Superada
P-20	1.0	Superada
P-21	1.0	Superada
P-22	1.0	Superada
P-23	1.0	Superada
P-24	1.0	Superada

RESULTADOS PRUEBAS DE ACEPTACIÓN		
Id		
P-25	1.0	Superada
P-26	1.0	Superada
P-27	1.0	Superada
P-28	1.0	Superada
P-29	1.0	Superada
P-30	1.0	Superada
P-31	1.0	Superada
P-32	1.0	Superada
P-33	1.0	Superada
P-34	1.0	Superada
P-35	1.0	Superada
P-36	1.0	Superada
P-37	1.0	Superada
P-38	1.0	Superada
P-39	1.0	Superada
P-40	1.0	Superada
P-41	1.0	Superada
P-42	1.0	Superada
P-43	1.0	Superada
P-44	1.0	Superada
P-45	1.0	Superada
P-46	1.0	Superada
P-47	1.0	Superada

Tabla 23: resultados pruebas de aceptación.

Sin embargo, a pesar de haberse superado satisfactoriamente todas las pruebas, conviene indicar el coste en la elaboración de las mismas puesto que al realizarse en un simulador, se requiere de un número aproximado de 5 o 6 simulaciones por prueba para verificar la salida.

6 Conclusiones y líneas futuras

Tras la finalización del proyecto se presentan las conclusiones obtenidas (sección 6.1) y las posibles vías de desarrollo por las que este proyecto puede ser ampliado (sección 6.2).

6.1 Conclusiones sobre el proyecto

Concluido el desarrollo del proyecto, en esta sección se exponen las dificultades que han tenido que superarse y las aportaciones que pueden obtenerse tras la finalización del mismo.

6.1.1 Dificultad del proyecto

A lo largo del proyecto ha sido necesario superar ciertas barreras, relacionadas tanto con la integración del protocolo como con el modo de implementación del mismo, así como determinados aspectos de la implementación del sistema de visualización de mensajes y aspectos vinculados a la creación de un proyecto colaborativo.

Para comenzar, el principal problema encontrado a lo largo del proyecto fue el análisis de la implementación interna del simulador NCTUns, especialmente apreciar que está creado sobre un núcleo especialmente desarrollado para su funcionamiento. Por ello, identificar que la dirección de *broadcast* es 1.0.1.255, en lugar de 255.255.255.255, requirió un gran esfuerzo. Por otra parte, vinculado al simulador, tampoco fue tarea fácil conseguir identificar la clase "event" y en concreto la función "read_trafficGen" como el lugar en el que se obtiene la información de los nodos participantes en la simulación, para poder ofrecer la identidad de dichos nodos al sistema de visualización de mensajes. Por otro lado, en lo referente a la implementación de la transmisión de los mensajes realizada por el creador de NCTUns, la detección del empleo del parámetro "*Transmission Power*", presentado en la sección de implementación (sección 5.1), fue otra de las cuestiones difíciles de resolver.

En relación con el sistema de visión de mensajes de la simulación, fue laboriosa la implementación de una interfaz en la que, presentando mucha información, la visualización fuese clara y sin retardos en la actualización de los elementos.

Por otra parte, teniendo en cuenta la inexperiencia en el desarrollo de sistemas, la creación de componentes y clases que, en la medida de lo posible fuesen reutilizables, no ha resultado fácil.

Finalmente, no hay que olvidar citar los conflictos de realizar un trabajo de integración con otro proyecto, especialmente en un caso como éste, en el que se necesita una elevada sincronización para que ambos proyectos se realicen de forma continuada, sin necesidad de que uno de ellos se detenga por retardos en el desarrollo del complementario. Por tanto, el problema principal no es debido a errores en la integración de ambas partes sino al hecho de tener que realizar un trabajo en paralelo, puesto que en determinados momentos surgen problemas que dificultan el mantenimiento de dicho paralelismo.

6.1.2 Aportaciones obtenidas

Muchas son las conclusiones que se pueden obtener tras la realización de un proyecto como el presentado en este documento, sin embargo, se pueden resumir realizando cuatro consideraciones, las cuales se corresponden con la visión obtenida tras la creación de un sistema completo, el aporte personal adquirido tras la finalización del mismo, el producto obtenido mediante un desarrollo en paralelo y el contraste entre lo esperado y lo realizado.

Por una parte, se puede afirmar que la creación de un sistema, desde el estudio del problema hasta la realización de las pruebas y con independencia de sus objetivos, es un proceso laborioso que supone muchas horas de trabajo y sacrificio. Además, aquellos proyectos a los que se les asocia una investigación previa a la realización del mismo, requieren un esfuerzo adicional para estudiar la situación actual. Dicho estudio es imprescindible y anterior al verdadero comienzo del desarrollo del proyecto, lo cual es un trabajo altamente gratificante y valioso pero con un gran esfuerzo asociado.

Por otro lado, es indiscutible el aporte personal obtenido tras la realización de este proyecto puesto que se adquiere una visión realista de lo que es la creación de un sistema en su completitud. Se consigue, también, una percepción sobre la habilidad personal para la distribución y estructuración del trabajo y se aumenta la fluidez tanto en la propuesta de soluciones como en el modo de llevarlas a cabo, bien programando o bien buscando caminos alternativos.

Por otra parte, considerando la integración de este proyecto con otro complementario, para el completo funcionamiento del protocolo EVIGEN en el simulador NCTUns 5.0 (1), hay que subrayar la coordinación y el trabajo en equipo como principios fundamentales para el éxito de los objetivos. De hecho, es indispensable el desarrollo en paralelo en gran parte del tiempo planificado para que ambos proyectos avancen adecuadamente. Además, la creación de un proyecto de forma colaborativa conduce a obtener un resultado más llamativo, por aportar mayores funcionalidades, que de ser un desarrollo individual no hubiesen podido efectuarse por motivos de tiempo y esfuerzo.

Finalmente, hay que afirmar la satisfacción de los objetivos puesto que se ha evaluado, con éxito, el correcto funcionamiento de la integración del protocolo y del sistema de visión de mensajes de la simulación. Por este motivo, los resultados obtenidos son equiparables a los deseados.

6.2 Líneas futuras

Aunque finalizado el presente proyecto se dejan varias puertas abiertas que pueden ser cerradas en proyectos futuros.

Por una parte, actualmente los nodos móviles, correspondiéndose con los roles Requester, Witness y NoEquipped, recogen información del entorno por medio de mensajes de *beaconing* en los que se informa, individualmente, de la identificación, la velocidad, la posición y el tipo de vehículo/rol de cada uno de ellos. Sin embargo, en el mundo real los vehículos están dotados de sensores que recopilan la información de su entorno de forma independiente, siendo esto un proceso pendiente en este proyecto.

Por otra parte, desde el comienzo del proyecto se ha considerado la utilización del protocolo EVIGEN como herramienta para la revocación de sanciones por exceso de velocidad, pero podría utilizarse para la revocación de otro tipo de sanciones, como por adelantos indebidos o por no realizar una parada en un determinado semáforo.

Otra de las posibles líneas de desarrollo se relaciona con el estudio empírico de la corrección, validez y satisfacción del protocolo EVIGEN, de modo que realizando un número sustancial de pruebas, modificando los parámetros más relevantes de todos los presentados, se determine detalladamente el éxito o fracaso de la implantación del protocolo.

Por otro lado, considerando la gran carga computacional requerida para el funcionamiento de EVIGEN, se podría modificar la implementación realizada para que el sistema fuese ejecutado de forma distribuida, contribuyendo a la repartición del trabajo entre distintos computadores. De este modo, se podrían realizar escenarios más variados, con multitud de combinaciones en cuanto a la creación de circuitos y a la cantidad de nodos participantes.

Finalmente, verificándose la existencia de una nueva versión del simulador NCTUns, versión 6.0, una posible vía de desarrollo se basaría en la migración de toda la implementación correspondiente con el desarrollo del protocolo EVIGEN al simulador NCTUns 6.0.

7 Bibliografía y referencias

Finalizado el desarrollo del presente proyecto, se expone la bibliografía y las referencias utilizadas a lo largo del documento.

1. Página principal NCTUns, Wang, S.Y. Network Simulator and Emulator. [En línea] http://nsl.csie.nctu.edu.tw/nctuns.html.
2. **Rueda, Sergio García.** Extensión de NCTUns 5.0 para simular el entorno de infraestructura y desarrollo del sistema de representación de indicadores para EVIGEN.
3. **Manzano, Lorena González.** ANEXOS, Extensión de NCTUns 5.0 para simular el entorno inalámbrico y desarrollo del visor de mensajes intercambiados para EVIGEN.
4. *Protocolo de creación de evidencias en entornos vehiculares. Congreso Iberoamericano de Seguridad de la Información (CIBSI), Montevideo (Uruguay), 2009.* **J.M. de Fuentes, A.I. González-Tablas, A. Ribagorda, B. Ramos.** págs. http://www.fing.edu.uy/inco/eventos/cibsi09/docs/Papers/CIBSI-Dia3-Sesion6%282%29.pdf.
5. **Wang, Shie-Yuang, Chih-Liang, Chou y Chih-Che, Lin.** Manual NCTUns 5.0, NCTUns Network Simulator and Emulator.
6. Patrones de diseño en arquitecturas java. [En línea] http://java.sun.com/blueprints/patterns.
7. Fedora10. [En línea] http://fedoraproject.org/.
8. Java Swing . [En línea] http://download.oracle.com/javase/tutorial/uiswing/.
9. Windows Forms. [En línea] http://windowsclient.net/default.aspx.
10. GTK+. [En línea] http://www.gtk.org/.
11. iText. [En línea] http://itextpdf.com/.
12. iTextSharp. [En línea] http://www.elguille.info/colabora/NET2005/Percynet_Funcionalidad_itextsharp_con_csharp.htm.
13. AXIS. [En línea] http://ws.apache.org/axis/.
14. CORBA. [En línea] http://www.corba.org/.
15. Microsoft Visual Studio. [En línea] http://www.microsoft.com/spain/visualstudio.
16. NetBeans. [En línea] http://netbeans.org/.
17. **Erich Gamma, Richard Helm, Palph Jhonson y John M. Vlissides.** *Design Patterns: Elements of Reusable Object-Oriented Software.*
18. Guía ESA para la definición de requisitos. [En línea] http://cisas.unipd.it/didactics/STS_school/Software_development/Guide_to_the_SW_requirements_definition_phase-0503.pdf.
19. OpenSSL. [En línea] http://www.openssl.org/.
20. **Soto.C, R.** Redes Vehiculares ad hoc - VANET. [En línea] http://ciiaas.org/pdf/meses/20%20abr%2009/CIIAAS.20.04.2009.pdf.

8 Glosario de términos

Este glosario se compone de las definiciones y acrónimos utilizados a lo largo del documento, junto con su correspondiente descripción.

Las definiciones se presentan en la Tabla 24 posterior.

Definiciones	Descripción
AES	También conocido como Rijndael, es uno de los algoritmos más populares para las operaciones de cifrado simétrico. El algoritmo está basado, principalmente, de una cadena de sustituciones y permutaciones.
Beaconing	Envíos periódicos de mensajes informativos, en este caso información de estado de cada nodo. Estos envíos se realizan a un número indeterminado de nodos, los cuales se encuentran dentro de la zona de transmisión.
CondTesti	Cadena cifrada que contiene el mensaje solicitado, en este caso se corresponde con el intervalo de velocidad a la que se indica que circulaba un determinado nodo.
CORBA	Estándar que establece una herramienta para el desarrollo de aplicaciones distribuidas desarrolladas utilizando los mismos o distintos lenguajes de programación.
CRL	Lista correspondiente con los identificadores de nodo cuyo certificado está revocado. Es el equivalente a la propia definición de CRL, lista de los certificados revocados.
Evidencia	Mensaje enviado a una RSU tras haber recibido una notificación, de sanción en este caso, y realizado por medio de las respuestas recibidas de nodos con rol Witness y NoEquipped.
GTK+	Librería, desarrollada por el equipo GTK+, para el desarrollo de interfaces de usuario. Permite la programación en multitud de lenguajes como son C, C++, C#, etc.
HTTP	Protocolo que define el intercambio de datos en internet. Se caracteriza por ser un protocolo sin estado y orientado a transacciones.
IDL	Es una parte del estándar CORBA. Se corresponde con el lenguaje de definición de interfaces para la comunicación de componentes desarrollados en distintos lenguajes de programación.
Información estado	Referente a un nodo con rol Requester, Witness o NoEquipped, esta información se corresponde con la posición en la que se sitúan en la pantalla del simulador NCTUns, la velocidad, el tipo de vehículo/rol (Witness o NoEquipped) y el identificador. Referente a un nodo con rol RSU, esta información se refiere a la posición e identificador de cada uno de ellos.

Definiciones	Descripción
Javadoc	Documentación en HTML, proporcionada por Sun Microsystems, que describe cada una de las librerías disponibles para la programación en lenguaje Java.
Java RMI	Estándar que establece una herramienta para el desarrollo de aplicaciones distribuidas desarrolladas utilizando Java como lenguaje de programación.
Java Swing	Librería desarrollada para la creación de interfaces de usuario haciendo uso del lenguaje Java.
MVC	Patrón de diseño basado en el aislamiento de la parte lógica del sistema con respecto a la interfaz de usuario, permitiendo un desarrollo independiente. Consta de tres partes: modelo, el cual almacena los datos, controlador, encargado de las solicitudes al modelo, y vista, responsable de la interfaz de usuario.
Netbeans	Plataforma desarrollada para la creación de aplicaciones de escritorio, ofrece servicios como la administración de interfaces de usuario, la administración de la configuración del usuario, la administración del almacenamiento o la administración de ventanas.
Nodo	Entidad que hace referencia a un elemento de la interfaz presentada en el simulador NCTUns. En este caso corresponde con cada uno de los iconos con la imagen de un vehículo, a los cuales se les asocian roles del protocolo, Witness, NoEquipped, Requester, RSU, DGT o AC.
NoEquipped	Rol, tomado por cualquiera nodo partícipe en la simulación, encargado de realizar los reenvíos de testimonios que reciba de Witness u otros NoEquipped o RSU.
Notificación	Mensaje asociado a la recepción de una sanción por medio del cual se comienza la ejecución del protocolo.
NVT ASCII	Formato de representación de mensajes para la comunicación de distintas máquinas.
PHP	Lenguaje interpretado diseñado principalmente para el desarrollo de páginas web. Normalmente se sitúa del lado del servidor, utilizándose como entrada y creándose páginas web como salida.
Requester	Rol, tomado por cualquier nodo partícipe en una simulación, encargado de recibir las notificaciones, sanciones en este caso, y enviar la solicitud de un testimonio a nodos con rol Witness para poder revocar la sanción.
RSU	Rol, tomado por cualquiera nodo partícipe en la simulación, encargado de realizar la función de intermediario entre los distintos nodos partícipes en la misma.
SHA-1	Función criptográfica para la creación de resúmenes de 20 bytes. Es una de las funciones más utilizadas tanto en aplicaciones de seguridad como en protocolos.

Definiciones	Descripción
Simulador NCTUns	Sistema en el que se pueden visualizar las simulaciones y en el que se ha desarrollado el protocolo.
SOAP	Protocolo desarrollado para el intercambio de información en un entorno distribuido, con independencia del lenguaje y del sistema operativo. El transporte de los mensajes se produce mediante HTTP y el formato de los mensajes está basado en XML.
Socket	Elemento utilizado tanto para la comunicación de sistemas como para el desarrollo de aplicaciones distribuidas.
TCP	Protocolo de nivel de transporte utilizado para el intercambio de segmentos. Proporciona servicios que garantizan que los datos son entregados, en el orden adecuado y retransmitidos en caso de pérdida. Es un protocolo fiable.
Testimonio	Mensaje construido por nodos con rol Witness, NoEquipped y RSU (actuando como NoEquipped), tras la recepción de una solicitud.
Ticket	Cadena cifrada que contiene la clave utilizada en CondTesti para realizar la operación de cifrado.
UDP	Protocolo de nivel de transporte, utilizado para el intercambio de datagramas. No se proporcionan mecanismos para garantizar la llegada de los mensajes ni el orden de los mismos. No es un protocolo fiable.
Vecino	Nodo que se sitúa dentro de la zona de transmisión de mensajes.
Vehículo	Nodo correspondiente con la ejecución del rol Witness, NoEquipped o Requester.
Witness	Rol, tomado por cualquiera nodo partícipe en una simulación, encargado de crear el testimonio con el intervalo de velocidad a la iba el nodo que realiza la solicitud.
XML	Modo de definir lenguajes, para distintas necesidades, basado en el intercambio de información estructurada entre distintas plataformas.

Tabla 24: definiciones

Los acrónimos utilizados se muestran en la Tabla 25 posterior.

Acrónimos	Descripción
AC	Autoridad de Certificación.
AES	Advance Encryption Standard
CORBA	Common Object Request Broker Architecture (Arquitectura común de intermediarios en peticiones a objetos)
CRM	Certificate Revocation List (Lista de certificados revocados).
DGT	Dirección General de Tráfico.
ESA	European Space Agency (Agencia Espacial Europea)
HTTL	Hypertext Transfer Protocolo (Protocolo de transferencia de hipertexto).
IDL	Interface Definition Language (Lenguaje de definición de interfaces).
JMV	Java Virtual Machine (Máquina virtual de Java)
MVC	Model-View-Controller (Modelo Vista Controlador).
NVT ASCII	Network Virtual Terminal ASCII.
PDF	Portable Document Format (Formato portable para documentos).
RSU	Road Side Unit.
SHA-1	Secure Hash Algorithm
SOAP	Simple Object Access Protocol.
TCP	Transmission Control Protocol (Protocolo de control de transmisión).
TTL	Time To Live (Tiempo de vida).
UDP	User Datagram Protocol.
XML	Extensible Markup Language (Lenguaje de marcado extensible).

Tabla 25: acrónimos.