UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN TECNOLOGÍAS DE LA TELECOMUNICACIÓN

Entorno de simulación de redes distribuido basado en ns-3 y computación en nube

Autor:

D. Sergio Serrano Iglesias

Tutores:

Dr. D. Miguel Luis Bote Lorenzo

Dr. D. Eduardo Gómez Sánchez

Valladolid, 14 de Julio de 2016

TÍTULO:	**Entorno de simulación de redes distribuido basado en ns-3 y computación en nube**
AUTOR:	**D. Sergio Serrano Iglesias**
TUTOR:	**Dr. D. Miguel Luis Bote Lorenzo**
	Dr. D. Eduardo Gómez Sánchez
DEPARTAMENTO:	**Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

TRIBUNAL	
PRESIDENTE:	**Dr. D. Yannis Dimitriadis Damoulis**
VOCAL:	**Dr. D. Manuel Rodríguez Cayetano**
SECRETARIO	**Dr. D. Juan Ignacio Asensio-Pérez**
FECHA:	**19 de Julio de 2016**
CALIFICACIÓN:	

Resumen

El uso de los simuladores tiene especial interés en entornos educativos, donde los alumnos pueden emplearlos para reforzar los conocimientos adquiridos en las sesiones de teoría, con un bajo coste y una considerable flexibilidad. Sin embargo, su tiempo de ejecución es elevado, sobre todo en las simulaciones de barrido de parámetros. En este trabajo se propone diseñar, desarrollar y evaluar un entorno de simulación distribuido de redes TCP/IP que haga uso de la nube computacional para la ejecución distribuida de las simulaciones basadas en el simulador ns-3. Esta aplicación parte de una versión anterior, el DNSE (Distributed Network Simulation Environment), que ofrecía un entorno similar basado en grid computacional, y de un trabajo teórico de traslación de esta solución a la nube computacional, llamado DNSE3. En este TFG se realiza un rediseño, aprovechando servicios de infraestructura de nubes populares (AWS y OpenStack), incluyendo mecanismos para la escalabilidad automática. Tras desarrollar un prototipo completamente funcional, se evalúa su rendimiento en pruebas sintéticas y se discute su posible uso en un escenario académico.

Palabras clave

Simulación, barrido de parámetros, nube computacional, REST, servicios, escalado automático

Abstract

Simulations have some interest in educational environments, as the students can reinforce the knowledge acquired in the theory lectures, with low cost and a considerable flexibility. One of their main issues is the amount of time required for their realization, especially in the parameter sweep simulations. In this project we attempt to design, develop and evaluate a distributed network simulation environment based on cloud computing that uses the ns-3 network simulator. This application is based on previous attempts, such as the DNSE (Distributed Network Simulation Environment), which could execute multiple distributed simulations based on grid technology, and the analysis of its migration to cloud computing. In this project we will redesign the application, taking advantage of some services offered by well-known cloud infrastructures (AWS and OpenStack) and including facilities for its automatic scaling. When we have raised a completely functional prototype, we will test its performance in some predefined test as well as we discuss its deployment in academic scenarios.

Keywords

Simulations, parameter sweep simulations, cloud computing, REST, services, automatic scaling

Índice

Índice de figuras

Índice de tablas

Agradecimientos

Son demasiadas las personas a las que debo de dar gracias por haber llegado hasta aquí. En primer lugar, gracias a Miguel y a Eduardo, por ofrecerme este proyecto. Les tengo que dar gracias por no mostrar demasiadas quejas cuando mandaba los informes, tediosos al principio pero útiles al final.

También quiero dar las gracias a todos los miembros del GSIC/EMIC, por dejarme entrar en este fantástico grupo de trabajo. En especial quería dar las gracias a Alejandro, Juan, Osmel, Rafael y Sonia, al ser las primeras personas que conocí del grupo y ayudarme a integrarme. Demasiados son los recuerdos que tengo de todo este tiempo.

No me voy a olvidar de todos los compañeros que he ido conociendo a lo largo de los estudios. Demasiados por nombrar y no quiero olvidarme de ninguno de ellos. No sé como habría sido todos estos años sin el apoyo y la amistad que se han ido forjando estos años. También quiero agradecer en especial a aquellas personas que se den por aludidas por el apoyo durante este último curso y desagradecerlas por la corrupción a la que me han llevado.

Otro agradecimiento especial se lo llevan Adrián, Adrián, Diego, Víctor y Víctor, por seguir manteniendo esta amistad desde hace tantos años y no haber perdido nunca el contacto, compartiendo buenos ratos y experiencias mientras podía estar en Burgos.

Por último, pero no menos importante, quería agradecer a mi familia: mis padres, mi hermana, mi abuela y mis diferentes tíos y primos. Gracias por preocuparse por cómo estaba llevando el curso, por interesarse en las numerosas chapas en las que les explicaba lo que hacía y, en la mayoría de los casos, no enterarse de misa la media. Gracias por intentar animarme en esos moemntos que tenñia el ánimo mas bajo de lo normal.

Gracias a todos de corazón.

No importa dónde vayas,

siempre recuerda de dónde vienes.

Anónimo

Capítulo 1. Introducción

1.1. Motivación

El uso de simulaciones es muy común en entornos académicos y de investigación. Permiten aplicar diversos modelos matemáticos para experimentar en diferentes entornos o elementos que serían muy difíciles de tratar en la realidad, ya sea por coste o por su inviabilidad. Es por esto que las simulaciones abarcan múltiples campos de estudio: desde el estudio de diferentes modelos físicos hasta el test de impacto de coches de lujo [Law07]. Los modelos simulados permiten la entrada de parámetros de configuración que permiten ajustar las condiciones evaluadas, ofreciendo una gran flexibilidad en su uso.

Esta versatilidad de las simulaciones ha propiciado su uso en entornos educativos. Los alumnos pueden aprovechar las simulaciones para contrastar los resultados obtenidos con los que se obtienen de los modelos teóricos vistos en las sesiones teóricas, reforzando los conocimientos adquiridos [Win95]. Si además añadimos la posibilidad de modificarlos con los parámetros de entrada, se puede estudiar en detalle los diferentes modelos teóricos en multitud de escenarios sin obligar al alumno a tener que comprender nuevos modelos de simulación.

Dentro de los diferentes campos en los que resulta de utilidad el uso de las simulaciones podemos encontrar el estudio de las redes telemáticas. Diversos simuladores como ns-2, su sucesor ns-3 u OMNet++, entre otros, permiten estudiar el comportamiento de diferentes redes ante generación y cursado del tráfico entre los nodos que conforman dichas redes. La simulación de redes telemáticas ha sido utilizada para, entre otras cosas, buscar las condiciones en las cuales se produce congestión en determinados puntos de la red, determinar el retardo que se produce en la comunicación tras atravesar toda la red o estudiar el comportamiento de un nuevo protocolo en diferentes entornos, entre otros [WvLW09].

El uso tradicional de simulaciones en un contexto educativo conlleva estudiar un escenario concreto, con unos valores particulares de sus parámetros de entrada, analizando cada uno de los aspectos que lo conforman. Sin embargo, otras veces tiene mucho más interés ver la evolución de un modelo al ir variando alguno de sus parámetros de entrada. En el caso de las simulaciones de redes telemáticas, podría ser interesante observar para qué valores se produce congestión en alguno de los nodos o cuando hay un incremento nada despreciable del retardo extremo a extremo de la comunicación. Para realizar este tipo de simulaciones se pueden emplear las simulaciones de barrido de

parámetros. En estas simulaciones se define el rango de posibles valores que puede tomar alguno de los parámetros de entrada del modelo a simular y, en el momento de ejecución, se llevan a cabo simulaciones para cada una de las combinaciones posibles de valores de los parámetros a "barrer". Lo habitual es que el procesamiento de las simulaciones se realice de forma iterativa.

Este tipo de simulaciones presentan un problema por la forma en la que son procesadas. Como cada simulación puede tener un tiempo de ejecución significativo y, si además añadimos que generalmente el número de simulaciones necesario para un barrido puede ser elevado, se requiere de una gran cantidad de tiempo para completar las simulaciones de barrido de parámetros y, por consiguiente, se tarda más en obtener los resultados con los que poder trabajar. Este problema no es para nada desconocido y se ha tratado en diferentes artículos, como puede ser el escrito por Fujimoto [Fuj16]. De hecho, se proponen dos aproximaciones para minimizarlo en la medida de lo posible: la computación en paralelo y la computación distribuida. Ambas soluciones se basan en la ejecución en paralelo de diversos procesos que no necesitan compartir recursos de computación. Dado que las simulaciones de un barrido de parámetros son totalmente independientes entre sí en la ejecución, existe un alto nivel de paralelismo que puede explotarse mediante sistemas multihilo o multiproceso.

En los sistemas multiprocesador se busca aprovechar los recursos disponibles en una única máquina (múltiples núcleos o procesadores) para ejecutar el mayor número de simulaciones en procesos distintos. Esta solución la han incorporado algunos de los simuladores de forma nativa, como son ns-3 y OMNet++ [Seg10, Ltd07]. El problema fundamental de esta aproximación es la falta de escalabilidad, limitada por el hardware disponible. El número de hilos o procesos que se pueden ejecutar en paralelo está limitado, y es muy costoso ampliar el hardware para conseguir aumentar esta capacidad.

La computación distribuida pretende solventar este problema utilizando una red de ordenadores. En lugar de utilizar los recursos de una única máquina, el trabajo a computar se reparte entre distintos nodos de la red. Un problema que puede surgir con este tipo de solución es la necesidad de sincronismo entre los distintos nodos del sistema, que conllevaría esperas forzadas por las altas latencias de la comunicación sobre la red. Sin embargo, en un barrido de parámetros, dado que las simulaciones son independientes unas de otras, no supone un problema. Sin embargo, sí hay que tener presente que surgirán retardos debido a la comunicación a través de la red, lo cual hace que el mayor nivel de paralelismo sólo sea rentable cuando el número de tareas es elevado y su tiempo de computación significativamente mayor que el de la comunicación entre nodos.

Dentro de los planes de estudio de los Grados en Tecnologías y Tecnologías Específicas de Telecomunicación de la Universidad podemos encontrar algunas asignaturas en las cuales se pretende aprovechar las ventajas del uso de las simulaciones para reforzar los temas tratados en los contenidos de la asignatura. En particular, en las asignaturas "Ingeniería de Tráfico en Redes Telemáticas" y "Teletráfico", impartidas respectivamente en los grados mencionados, se hace uso del simulador ns-3 para comprobar el comportamiento de diversas redes desde diferentes puntos de vista, como puede ser bajo el modelado del sistema de colas o desde la calidad de servicio (*Quality of Service*, QoS). En alguna de las prácticas desarrolladas se hace uso del barrido de

parámetros para ver bajo qué condiciones se puede aplicar cada uno de los modelos de colas estudiados y, tal y como se ha estado explicando, se requiere de una gran cantidad de tiempo para completar las simulaciones y poder realizar las tareas pedidas. En ocasiones, el tiempo necesario para la obtención de dichos resultados puede llegar a superar el tiempo disponible en las sesiones de laboratorio.

En el Plan Antiguo del Título de Ingeniero de Telecomunicación e Ingeniero Técnico de Telecomunicación también se utilizaba un simulador de redes, en este caso el ns-2, para realizar estudios similares en redes telemáticas. Para solventar el problema mencionado, Bote et al. [BLAPGS+12] desarrollaron la aplicación DNSE (*Distributed Network Simulation Environment*), la cual se basaba en un paradigma de computación distribuida para realizar las simulaciones, el grid computacional, pretendiendo repartir los trabajos en una red de nodos formada por una federación de distintas organizaciones[1] para realizar una distribución de las simulaciones en función de su demanda. El DNSE se utilizó en las sesiones prácticas de las asignaturas impartidas en los planes antiguo con gran éxito, hasta que se cambió de simulador en las asignaturas por el ns-3. Debido a los problemas que acarreaba el uso de esta solución, se optó por no utilizarla con el nuevo simulador.

Si nos fijamos en la demanda de simulaciones realizada por los alumnos en uno de los cursos anteriores, veríamos que, de forma general, la demanda realizada es relativamente baja. Solamente en aquellas prácticas que se requiere de un elevado número de simulaciones se producen picos de demandas, ya que así lo requiere la práctica. Si se decidiera desplegar un nuevo sistema distribuido para la realización de las simulaciones, debería de ser lo suficientemente escalable como

En los últimos años se ha establecido un nuevo paradigma de computación distribuida: la nube computacional. Este paradigma consiste en la utilización de recursos virtuales aparentemente infinitos con una gestión flexible y remota. Dentro de los recursos que puede ofrecer la nube podemos encontrar el almacenamiento remoto y de cómputo, mediante máquinas virtuales de características flexibles capaces de realizar diferentes tareas, como puede ser el acceso a diferentes servicios remotos [AFG+10]. Una de las ventajas que proporciona la nube computacional es la fácil escalabilidad de los recursos. En uno o dos minutos se pueden generar nuevas instancias de máquinas virtuales que sean capaces de cubrir los picos de demanda de los servicios y destruirlas cuando no sean necesarias. En el caso de las nubes públicas, el coste que se aplica al uso de la nube depende del uso que se haga de ella (tiempo de funcionamiento de las instancias, tráfico que haya circulado por la red de las instancias, cantidad de almacenamiento empleado…).

Estas capacidades que ofrece la nube computacional se podrían aprovechar para solventar el problema detectado en el barrido de las simulaciones, al ofrecernos la flexibilidad necesaria para cubrir los picos de demanda sin necesidad de la provisión de una gran cantidad previa de recursos. Esta propuesta se ha considerado en trabajos anteriores, en los cuales se estudiaba la viabilidad de una aplicación capaz de aprovechar esta flexibilidad para permitir la computación distribuida de las simulaciones [Can12]. A

[1] A pesar de ello, los nodos de cómputo que formaban el grid en el que estaba desplegado el DNSE pertenecían todos a la Universidad de Valladolid.

esta aplicación se la denomina DNSE-3 (*Distributed Network Simulation Environment-3*).

En el estudio de dicha aplicación, se analizó el uso que se podría hacer de la nube para su despliegue, donde se destacó la gran adaptabilidad de los recursos tanto a nivel económico como logístico y medioambiental [Can12]. Por otro lado, se diseñó una arquitectura orientada a servicios para la aplicación, de forma que cada uno de los servicios realizara una parte de la funcionalidad de la aplicación, permitiendo una gestión y mantenimiento más cómodo y manejable al igual que presentaban un diseño que permitiera su uso en otros proyectos. Estos servicios, a su vez, se diseñaron con un diseño RESTful, exponiendo su funcionalidad a través de unos recursos o entidades más relevantes de cada uno de estos servicios y que utilizan una API (*Application Programming Interface*) REST (*Representational State Transfer)* basada en la capa de aplicación HTTP (*HyperText Transfer Protocol*).

El diseño de esta arquitectura se intentó realizar de forma que tuviera el mínimo acoplo posible con el entorno de nube empleado, entre otras cosas porque la mayoría de funcionalidades que se debían implementar de gestión de nube aún no se ofrecían en los gestores empleados. Sin embargo, a día de hoy, estos servicios presentan ya una integración propia en cada uno de estos entornos, la cual está mejor adaptada para dicha infraestrutura y presentan un mayor número de funcionalidades y un mejor soporte, por lo que se considera adecuado intentar aprovechar estos servicios, a expensas de perder dicha portabilidad. En este Trabajo Fin de Grado (TFG) se pretende realizar un rediseño de dicha aplicación, así como una primera implementación para comprobar que realmente se consigue los beneficios esperados del sistema distribuido.

1.2. Objetivos

Tal y como se comentó en la sección anterior, en este trabajo se pretende continuar con el trabajo realizado con la aplicación DNSE3. Es por esto que el objetivo principal de este trabajo será el siguiente:

- Rediseñar, implementar y desplegar una aplicación que aproveche las propiedades de la nube computacional para la ejecución de simulaciones de redes telemáticas TCP/IP.

Esta aplicación se diseñará de forma que pueda ser utilizada en el mayor número de entornos de nube computacional posible. Para ello, se restringirá las herramientas y *software* empleado en su implementación a aquellas que presenten licencias gratuitas. No obstante, deberá de poder ser compatible con herramientas con necesidad de licencia en aquellas situaciones que sean necesarias.

Esta aplicación estará orientada tanto a entornos académicos, para uso y disfrute por parte de los alumnos, como a entornos de investigación, de forma que ambos se puedan aprovechar del rendimiento esperado de la aplicación. Muestra de ello es el posible uso que se hará de ella en las asignaturas de “Ingeniería de Teletráfico en Redes Telemáticas” y “Teletráfico”, impartidas en los Grados en Tecnologías y Tecnologías Específicas de Telecomunicación en la Universidad de Valladolid. Con esta aplicación se pretende obtener los resultados procedentes de las simulaciones de forma más rápida al

utilizar la capacidad de computo de la nube computacional, así como la flexibilidad que ofrece.

1.3. Metodología de trabajo

A lo largo del desarrollo de la aplicación DNSE3 se ha seguido la metodología ofrecida por el Proceso Unificado, proceso iterativo en el que el desarrollo del proyecto sigue una serie de iteraciones o proyectos cortos de duración fija (en nuestro caso de 2 semanas), las cuales originan una serie de sistemas incrementales en los cuales se amplía y refina el proyecto [Lar03]. Las ventajas que aporta el desarrollo iterativo en la producción de software van desde la detección y mitigación temprana de riesgos altos hasta la recogida de retroalimentación tras completar cada una de las iteraciones y la mejor gestión de complejidad del desarrollo al trabajar en ciclos más cortos [Lar03].

El flujo de trabajo del Proceso Unificado está dirigido principalmente por los casos de uso, los cuales incluyen las funcionalidades o requisitos funcionales que debe ofrecer el sistema a cada tipo de usuario que lo utilice [JBR00]. Estos casos de uso, por una parte, influye en el diseño de la arquitectura del sistema, que establece las relaciones entre los diferentes elementos que componen el sistema, y, por otra, pueden establecer los diferentes objetivos que se deben alcanzar en las diferentes iteraciones [JBR00].

La organización del trabajo propuesta por el Proceso Unificado se divide en 4 fases [Lar03, JBR00]:

- Fase de inicio: en esta fase se establece una visión aproximada de cómo debe ser el sistema final, además de su viabilidad. Para ello se definen los principales casos de uso, una arquitectura básica que los cubra y una previsión de la planificación y coste del proyecto. Adicionalmente se identifican los riesgos más importantes en cuanto a su desarrollo.
- Fase de elaboración: se especifican en detalle los diferentes requisitos que debe cumplir el sistema por medio de los casos de uso y se realiza una implementación iterativa del núcleo de la arquitectura y los riesgos más importantes. Además, se estiman los recursos necesarios para terminar el proyecto.
- Fase de construcción: en esta fase se completa la implementación de la arquitectura al igual que el resto de casos de uso. Al finalizar esta etapa se dispone de una versión del producto orientada a su despliegue.
- Fase de transición: en esta fase se realizan pruebas del sistema desarrollado, detectando los posibles fallos o carencias que presente. Estos fallos se irán corrigiendo y ampliando las funciones ofrecidas por el sistema hasta conseguir la versión final del producto.

Dentro de cada fase se realizan una serie de tareas o flujos de trabajo, entre las cuales se presentan [Lar03, JBR00]:

- Captura de requisitos: en esta etapa se establecen una serie de conversaciones con los clientes que solicitan el proyecto para recoger las diferentes funcionalidades que debe satisfacer el sistema, así como una serie de condiciones que debe cumplir sin influir en su funcionamiento, también conocidos como requisitos no funcionales. Todos estos requisitos deben ser expuestos en el documento de

requisitos, en un lenguaje familiar para el usuario. Se suele generar también un modelo del dominio, que muestra las relaciones entre las distintas entidades que conforman el sistema.

- Análisis: se sintetizan los diferentes requisitos recogidos en la etapa anterior, estructurándolos y obteniendo una arquitectura del sistema. En esta síntesis se definen los diferentes casos de uso del sistema, esta vez en un lenguaje más técnico y preciso y menos orientado a su presentación al cliente.
- Diseño: en esta etapa se modela el sistema, incluyendo su arquitectura, para que soporte todos los requisitos planteados. Crea un punto de partida para las tareas de implementación, al definir cómo deben funcionar cada uno de los casos de uso. También facilita la estructuración de las iteraciones de la etapa de implementación, permitiendo descomponer los distintos trabajos en tareas más manejables.
- Implementación: partiendo del resultado de la etapa de diseño, se realiza la implementación software de los diferentes componentes del sistema, para obtener su código ejecutable.
- Pruebas: tras realizar la implementación del sistema, se verifica cada uno de los elementos, así como las versiones finales del sistema, detectando los posibles fallos o carencias que presente. Con los resultados de estas pruebas se vuelve a la etapa de implementación para solventar dichos problemas.

Tal y como se comentó anteriormente, este trabajo continúa con el estudio realizado sobre la viabilidad del DNSE3 en un entorno de nube computación. En este estudio se realizaron las etapas de captura de requisitos, análisis y diseño. Por nuestra parte, se retomará el flujo de trabajo realizando las tareas de implementación y pruebas, no sin antes realizar un rediseño del sistema, para adaptarlo a las condiciones actuales de trabajo.

Para ello, se realizarán artefactos UML que incluyan tanto los modelos de diseño de la arquitectura del sistema, los diagramas de secuencia que muestren la realización de cada uno de los casos de uso planteados, perteneciendo a la etapa de diseño, y el resultado de las diferentes pruebas realizadas tras la implementación del sistema.

1.4. Estructura del documento

En lo que queda de documento se ilustrará sobre el trabajo realizado con la aplicación DNSE3. Para ello, se dividirá en una serie de capítulos centrados cada uno en un tema relevante del desarrollo de la aplicación.

En el capítulo 2 se hablará sobre el estado del arte del DNSE3. Esto incluye, por un lado, las diferentes tecnologías y arquitecturas empleadas en su diseño y, por otro, el trabajo previo sobre el que se trabajará. Esto último incluye tanto la aplicación DNSE como el estudio realizado acerca del DNSE3.

En el capítulo 3 se mostrará la arquitectura final empleada. Se dividirán los servicios que conforman el sistema en aquellos que requieren de su implementación y en aquellos servicios que vienen integrados en el *middleware* de gestión de nube computacional empleado.

En el capítulo 4 se tratarán las pruebas realizadas con el núcleo del sistema. En primer lugar, se detallará la implementación realizada hasta el momento de la aplicación. Posteriormente, se detallarán cada una de las pruebas realizadas, definiendo sus condiciones y el objetivo que se desea alcanzar. Tras presentar los resultados obtenidos, se analizarán y se compararán con el rendimiento esperado.

Finalmente, en el último capítulo nos centraremos en las conclusiones extraídas de la implementación del DNSE3. Adicionalmente, se indicará la línea de trabajo futuro a seguir para obtener una versión final del sistema.

Capítulo 2. Estado del Arte

En el capítulo anterior se estableció el objetivo de este proyecto: el desarrollo de una aplicación que aproveche las capacidades de la nube computacional para permitir la ejecución de simulaciones de redes TCP/IP. Para alcanzar dicho objetivo, debemos utilizar diferentes herramientas, como son los simuladores de redes telemáticas y los servicios ofertados de nube. No obstante, antes de poder empezar a trabajar con ellos necesitamos conocer su utilidad y las diferentes opciones hay disponibles para cada uno.

En este capítulo se mostrarán las diferentes virtudes de las tecnologías en las que se basa nuestra aplicación, en particular, el uso de los simuladores, la nube computacional y el diseño de servicios RESTful. Adicionalmente, se mostrarán los trabajos previos realizados sobre esta misma aplicación, siendo la versión anterior de la misma, el DNSE, y el estudio realizado acerca de su despliegue en un entorno de nube computacional.

2.1. Simuladores de redes telemáticas

Según establecen Guizani et al., *las simulaciones son imitaciones de un sistema del mundo real a través de una recreación digital de su comportamiento según las reglas descritas por un modelo matemático* [GRKAF10]. Gracias a esta recreación, se pueden realizar estudios de los comportamientos de dichos sistemas. Por supuesto, no es la única forma de estudiar estos sistemas. Otras alternativas a las simulaciones se presentan a continuación [Law07, CS03]:

- Modelos analíticos: permiten comprender el comportamiento del sistema si se entiende correctamente el modelo analítico y, si es lo suficientemente sencillo, permite obtener resultados válidos rápidamente. Sin embargo, si se debe modelar un sistema más complejo, los modelos se vuelven demasiado complejos como para poder trabajar con ellos.
- Maquetas: permiten trabajar con equipamiento real, con lo que se obtienen resultados más cercanos a los reales en comparación con estudios analíticos. Por otra parte, las maquetas están preparadas para trabajar bajo unas determinadas condiciones, dificultando el estudio bajo otras condiciones distintas sin elevar su coste.
- Entornos reales: presentan la ventaja de obtener resultados fieles, pero presentan diferentes problemas que dificultan su uso, como son el coste, disponibilidad, seguridad... También hay que tener presente la poca flexibilidad de esta

alternativa, al estar sujeto a las condiciones de dicho entorno, como sucedía con las maquetas, aunque estás tiene mayor versatilidad.

- Emulación: utilizan los modelos de las simulaciones, generando entidades virtuales, que trabajan de forma conjunta con implementaciones reales, ya sean *hardware* o *software*. Combina las ventajas de las simulaciones y de los entornos reales, pero está más centrado en imitar el comportamiento del modelo emulado más que en su rendimiento.

Las simulaciones, por su parte, presentan una serie de ventajas frente a estas otras opciones, según indica Law [Law07]:

- Permiten modelar modelos mucho más complejos que los modelos analíticos.
- Permite comparar diferentes diseños de sistemas para determinar cuál cumple con los requisitos buscados.
- Se tiene un mejor control sobre las condiciones experimentales que si se tuviera que hacer en un modelo real.
- Permite evaluar grandes periodos de tiempo en un corto periodo de tiempo o estudiar el sistema detalladamente en un tiempo expandido.

Sin embargo, las simulaciones no son perfectas, sino que tienen también una serie de problemas asociados [Law07]. En primer lugar, con las simulaciones no se pueden obtener resultados exactos, sino estimaciones. Esto implica que se necesita realizar múltiples simulaciones para poder tener resultados fiables con los que trabajar, al igual que aumenta el tiempo necesario para obtenerlos. Por otra parte, el uso de los simuladores no es universal. Aunque puedan compartir ciertas pautas de uso o diseño, cada uno de ellos presenta un funcionamiento distinto que, además, requiere de unos conocimientos mínimos para su uso. Aunque este problema solo sucede en el primer acercamiento a la herramienta, mientras se aprende a utilizarla, se sigue necesitando de un tiempo hasta que se adquiere un manejo suficiente. Finalmente, aunque pueda parecer bastante obvio, otro problema relevante es el gran nivel de dependencia de la simulación con el modelo empleado. Si el modelo tiene algún fallo de diseño o de configuración, los resultados obtenidos tras su ejecución son erróneos y se deben de descartar, implicando un consumo de recursos y de tiempo desperdiciados. Este problema puede ser en ocasiones difícil de encontrar, por lo que se requiere de una evaluación exhaustiva del modelo antes de sacar conclusiones.

A pesar de estos hándicaps, dentro del estudio de redes telemáticas es el método de evaluación predominante, sobre todo en el desarrollo de nuevas arquitecturas y protocolos, tal y como comenta Weingärtner et al. [WvLW09]. En este aspecto destaca la flexibilidad ofrecida por los simuladores de ofrecer múltiples escenarios de simulación de una forma más cómoda que en una maqueta o despliegue real, donde las evaluaciones a efectuar se ven limitadas a los escenarios para los que han sido diseñadas.

2.1.1. Uso de los simuladores en entornos educativos

Además del uso de las simulaciones en investigaciones, se puede extender su uso a entornos académicos, donde los alumnos utilicen las simulaciones para experimentar los diferentes modelos estudiados en las sesiones de teoría. En esta clase de entornos se aprovechan especialmente la capacidad de controlar el modelo de forma sencilla y el

control del espacio de tiempo empleado en la simulación. Por otra parte, las desventajas de su uso como la obtención de simulaciones o el correcto diseño del modelo empleado no son especialmente relevantes. Su uso tiene el propósito de reforzar los conocimientos adquiridos por los alumnos en las sesiones teóricas, por lo que no se necesita una gran precisión en los resultados de las simulaciones.

De forma adicional, el uso de simulaciones en esta clase de entornos presenta otra serie de ventajas frente a otros modelos de enseñanza [Win95]. En comparación a los modelos analíticos se evita tener que trabajar con su formulación matemática. Por otra parte, evitan las distracciones que presentan las maquetas al dejar jugar a los alumnos con el material, además de que no impide su uso más allá del laboratorio. Sin embargo, a base de la repetición de las simulaciones y del ensayo-error, se puede tener una noción del comportamiento del modelo ignorando por completo el modelo empleado. Aunque se entiende en parte su funcionamiento, no se consiguen reforzar los conocimientos impartidos en las sesiones teóricas. Aun así, se ha corroborado el refuerzo positivo que suponen las simulaciones dentro de estos entornos [ZA03].

A la hora de utilizar las simulaciones en estos entornos, se pueden utilizar tanto las simulaciones individuales como las de barrido de parámetros. El uso de las primeras permite comprender tanto el funcionamiento tanto global como particular del modelo. Poniendo como ejemplo las simulaciones de redes telemáticas, se puede estudiar la evolución temporal de la comunicación transmitida o bien analizar la aplicación de los diferentes protocolos y algoritmos que intervienen en ella. El otro tipo de simulación, por su parte, no se centra tanto en cómo evoluciona un modelo sino ver cómo afectan sus parámetros en su comportamiento. El problema que se produce con el uso de estas simulaciones, tal y como se comentó en la introducción, es el elevado tiempo necesario para su ejecución, debido a la ejecución sucesiva de las diferentes simulaciones efectuadas.

2.1.2. Ejemplos de simuladores comerciales

En el ámbito de las redes telemáticas, disponemos de un gran número de simuladores disponibles, los cuales están basados de forma general en las simulaciones de eventos discretos. En este tipo de simulaciones, los distintos componentes que forman el modelo generan eventos a medida que se producen los sucesos, como por ejemplo la generación de un mensaje por parte de una aplicación. Estos eventos se almacenan dentro de una cola de eventos en función de su tiempo de ejecución y en la simulación se van procesando por orden [WvLW09].

Uno de los primeros simuladores en utilizar este tipo de simulaciones fue el simulador ns-2[2], desarrollado en 1995. Este simulador está desarrollado en C++, además de los nodos que componen el modelo a simular. Para configurar los modelos, como puede ser la introducción de los parámetros o la conexión de los nodos, se utiliza un *script* oTcL (*Object Tcl*) aparte, el cual evita tener que volver a compilar el modelo cuando se requieran realizar cambios en la topología [WvLW09, FV07]. Este simulador ha ganado gran prestigio dentro de los entornos de investigación y desarrollo, convirtiéndose en el

[2] `http://www.isi.edu/nsnam/ns/`

simulador de referencia dentro de dichos entornos. Por otra parte, presenta unos problemas como son el elevado consumo de memoria necesario cuando los modelos simulados tienen numerosos elementos y el tiempo de simulación necesario para su ejecución, directamente proporcional al tamaño del modelo [WvLW09]. Aunque su interfaz se basa en el uso de la línea de comandos, ofrece una herramienta gráfica de visionado de las animaciones del funcionamiento de los modelos llamada Network Animator (nam).

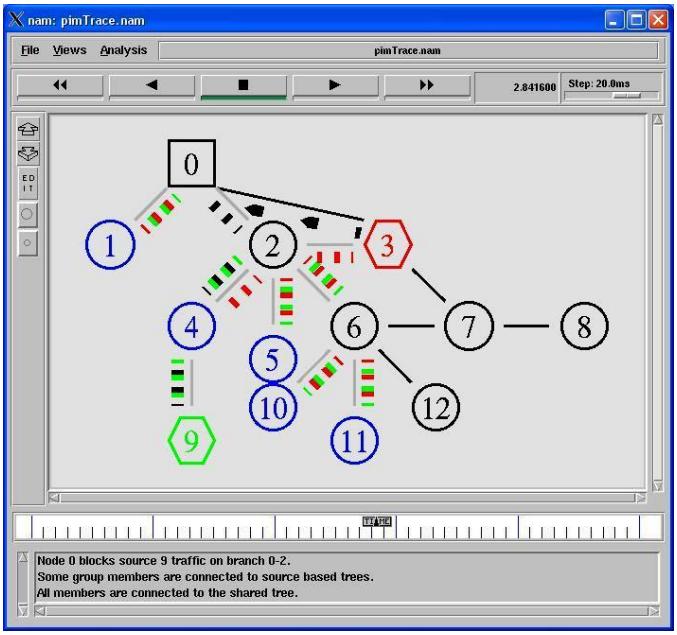

Figura 2.1: *Interfaz gráfica de la herramienta Nam ofrecida por el simulador ns-2.*

La siguiente versión de este simulador, ns-3[3], intentó solventar estos problemas. Este simulador, desarrollado en 2005, está escrito en C++, con ciertas partes desarrolladas en Python. A diferencia del ns-2, los modelos están escritos en C++, incluida su configuración. Esto obliga a tener que compilar el modelo cada vez que se haga un cambio, aunque se mejoran los problemas provenientes del uso del *script* oTcL [WvLW09]. Debido a la falta de compatibilidad con la versión anterior, el simulador dispone de una herramienta para adaptar los modelos anteriores. Otra de las características que lo diferencian de su predecesor es la posibilidad de ejecutar simulaciones en paralelo, aunque está más orientado a aprovechar al máximo el rendimiento de la máquina que en reducir el tiempo de ejecución y no permite una ejecución distribuida de forma nativa [Seg10]. También permite generar ficheros de trazas de la evolución de la simulación, incluyendo capturas de tráfico en formato *pcap*, las cuales se pueden utilizar en analizadores de tráficos de terceros como puede ser WireShark [WvLW09]. Al igual que el ns-2, este simulador presenta una interfaz basada en la línea de comandos, pero ofrece un entorno gráfico para el visionado de las animaciones. En esta ocasión, esta herramienta se conoce como NetAnim.

[3] `https://www.nsnam.org/`

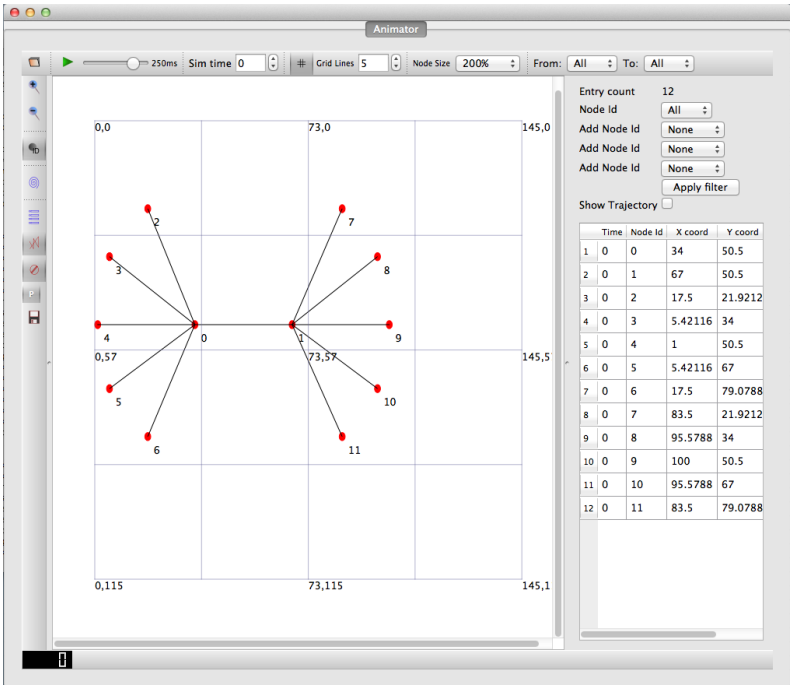

Figura 2.2: *Interfaz gráfica de la herramienta NetAnim ofrecida por el simulador ns-3.*

Otro simulador bastante empleado en la actualidad es el simulador OMNet++[4]. Éste, a diferencia de los anteriores, es un simulador de eventos discretos de propósito general, aunque se está empleando de forma habitual para simular toda clase de redes. A diferencia de los otros simuladores, éste proporciona un entorno de desarrollo integrado (IDE, *Integrated Development Environment*) gráfico. Está también desarrollado en C++, pero dispone de su propio lenguaje para la configuración de los modelos, NED (*Network Description*), que se transcribe a C++ antes de su ejecución [WvLW09]. De forma similar a lo que pasaba en el simulador ns-3, permite la ejecución en paralelo de las simulaciones, aún sin ser de forma distribuida.

[4] `https://omnetpp.org/`

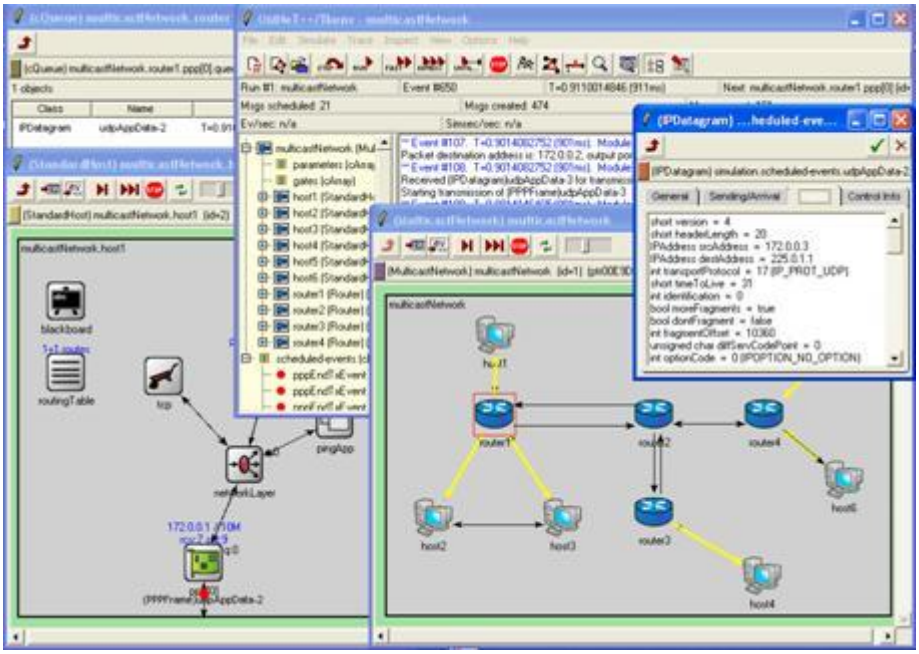

Figura 2.3*: Interfaz gráfica del simulador OMNet++.*

Los tres simuladores presentados anteriormente son de licencia gratuita y de código abierto, pero también hay presentes alternativas de pago. Una de las más conocidas es el simulador Riverbed Modeler[5], anteriormente conocido como OPNet Modeller., propiedad de Riverbed Technologies Inc. Este simulador estaba escrito inicialmente en C y portado posteriormente a C++. Presenta gran cantidad de modelos de protocolos y permite realizar modelos jerárquicos de gran profundidad. Dispone de un editor gráfico de los modelos, los cuales se convierten a un formato propietario, al igual que una interfaz gráfica para estudiar los resultados, análoga a la ofrecida por OMNet++. Al igual que ocurría con el ns-3 y OMNet++, este simulador también permite paralelizar las simulaciones a ejecutar.

[5] `http://www.riverbed.com/es/products/steelcentral/steelcentral-riverbed-modeler.html`

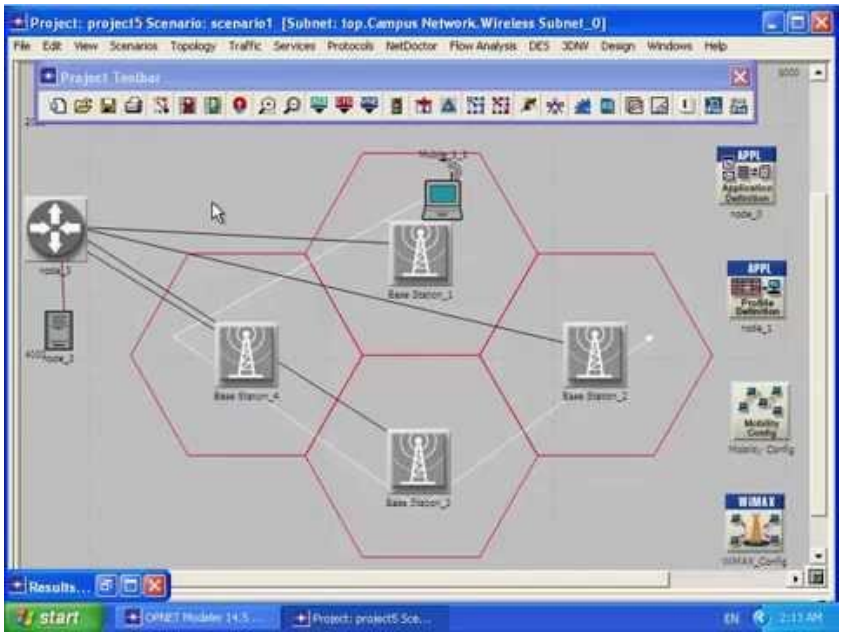

Figura 2.4*: Interfaz gráfica del simulador Riverbed Modeller*.

2.1.3. Propuesta de despliegue en el DNSE3

Para el desarrollo de la aplicación DNSE3 se ha optado por utilizar el simulador ns-3 en lugar del simulador ns-2 como se hizo en el DNSE por diversos motivos [Can12]. El primero de ellos, compartido por ambos simuladores, es que es *open-source* y de licencia gratuita. Dado que esta aplicación está planteada para ser utilizada en la mayoría de entornos de nube computacional, es de gran relevancia para permitir su libre distribución. Además, al ser *open-source*, se podrían realizar los cambios que se estimaran oportunos para mejorar sus capacidades. Otro de los motivos es la representación más fidedigna que hace el ns-3 de los nodos *software* empleados en el modelo en comparación al ns-2. Gracias a esto se consigue la ejecución de simulaciones más fieles a su implementación real. El último motivo planteado es el formato empleado en los resultados. El ns-3 permite almacenarlos en ficheros de traza que utilizan el formato empleado por otras aplicaciones del ámbito de las simulaciones de redes telemáticas, por lo que se simplifica la interacción del DNSE3 con otras herramientas.

2.2. DNSE

Tal y como se acaba de mostrar en la sección 2.1.2, algunos de los simuladores ofrecen alguna herramienta que permita la ejecución en paralelo de las simulaciones. Sin embargo, no ofrece la posibilidad de utilizarla en un entorno distribuido, que permitiría incrementar el número de simulaciones ejecutadas en paralelo sin necesidad de mejorar las capacidades de la máquina. Para solventar este problema, Bote et al. desarrollaron la aplicación DNSE [BLAPGS+12], que hacía uso del simulador ns-2 para ofrecer un sistema distribuido de simulación de redes telemáticas.

La aplicación DNSE se basaba en el uso del grid computacional, el cual consiste en una infraestructura que permita la compartición de *software* y *hardware* entre diferentes instituciones a través de la red [FK04]. El sistema presentaba una arquitectura

de servicios, cada uno orientado a una función que tenía que desempeñar. Así, las simulaciones se distribuían entre las máquinas que tuvieran el servicio de simulación instalado y otro servicio se encargaba de coordinar la ejecución entre ellas. Estos servicios se diseñaron siguiendo el marco WSRF (*Web Service Resource Framework*) [OAS]. Este *framework* permite modelar y acceder a los recursos de los servicios, definiendo, entre otras cosas, las operaciones soportadas por las interfaces y el formato de los mensajes empleados en la comunicación, siguiendo el diseño SOAP (*Simple Object Transfer Protocol*) [OAS].

El DNSE permitía la ejecución de simulaciones tanto individuales, como de barrido de parámetros, en las cuales se recorren los posibles valores de alguno de los parámetros de la simulación dentro de un rango de valores. Para su correcta ejecución se empleaba, además de los *scripts* y ficheros que conforman el modelo de simulación, un fichero XML que describía la simulación, los parámetros empleados, así como los ficheros que lo componen. La combinación de todos ellos formaba un paquete de simulación, sobre el cual se realizaban peticiones de simulación, ya sí individuales o de barrido de parámetros. En la Figura 2.5 se muestra la interfaz de usuario de la aplicación empleada para acceder al DNSE.

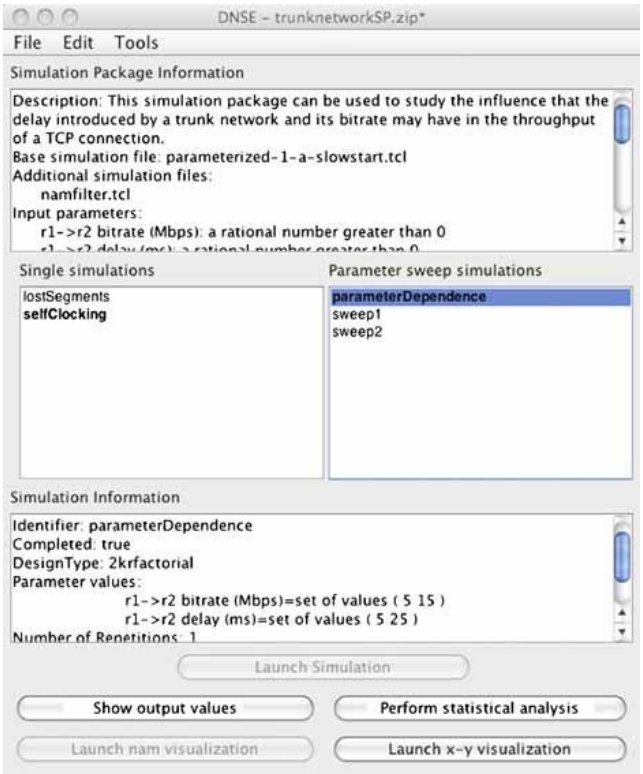

Figura 2.5: *Captura de pantalla de la interfaz del usuario del DNSE.*

Otra característica adicional que ofrecía el DNSE era la capacidad de mostrar animaciones de los modelos, procedente de los ficheros de animación generados tras la ejecución de la simulación, empleando para ello la herramienta nam ofrecida por el simulador ns-2. Tal y como se muestra en la Figura 2.6, estas animaciones permitían ver de forma gráfica el funcionamiento del modelo simulado, lo cual era de gran ayuda para los alumnos. Por una parte, les permitía ver el funcionamiento del modelo, pudiéndolo contrastar con su funcionamiento básico y su comportamiento antes posibles anomalías y

fallos. Por otra parte, al ver su funcionamiento, es bastante fácil detectar los posibles fallos que hayan surgido en la simulación cuando se obtienen resultados anómalos.

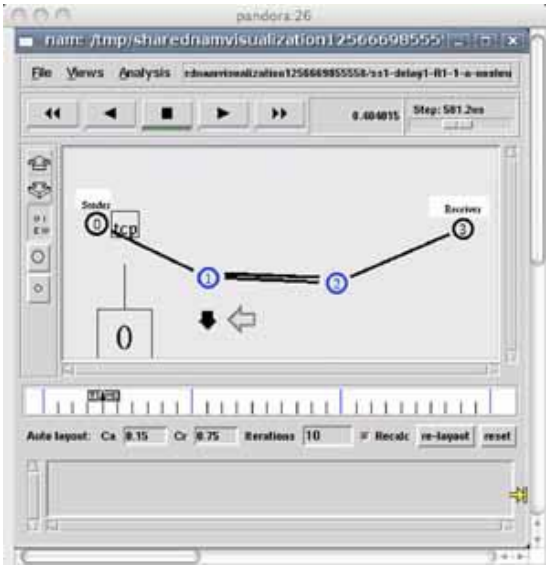

Figura 2.6: *Animación nam generada con el DNSE.*

La aplicación DNSE tuvo gran acogida dentro de las aulas, al reducir el tiempo necesario para la ejecución de las simulaciones de las sesiones prácticas, además de ofrecer un sistema gráfico de visionado de las simulaciones [BLAPGS+12].

2.2.1. Críticas al DNSE

El DNSE no estaba exento de problemas, la mayor parte de ellos vinculados a su diseño. En primer lugar, el sistema de grid computacional, a pesar de ser flexible y escalable utilizando las máquinas de diferentes administraciones, no es capaz de realizar dicho escalado de forma automática, sino que se requería de un administrador capaz de configurar adecuadamente las máquinas para que sean capaz de trabajar de forma coordinada con el resto. Este hecho hace que la adaptación del grid a la demanda de trabajos, a pesar de ser posible, es bastante costosa y poco eficiente. Por otro lado, los servicios WSRF, por su propia composición, no proporcionan un mecanismo sencillo para monitorizar el estado de cada uno de ellos. La monitorización de estos servicios permite conocer en detalle la demanda que se está efectuando de la propia aplicación, lo que facilitaría el adecuado escalado de la propia infraestructura. Además, han caído en desuso a favor de alternativas más sencillas de utilizar e implementar, como son los servicios REST.

Otro problema que presenta el grid computacional es el poco éxito que obtuvo. A pesar de ser un paradigma prometedor y de realizarse trabajos importantes con él [Fos01], se trataba de una tecnología bastante compleja, hasta el punto que la documentación existente era insuficiente. En resumen, se trata de una tecnología no amigable con los nuevos desarrolladores que quieran utilizarla. Estos motivos propiciaron que, cuando se efectuó la actualización del simulador en las asignaturas del ns-2 a su siguiente versión, el ns-3, se dejase de utilizar.

Sin embargo, la aplicación no se abandonó por completo. Se buscaron diferentes formas de mejorar sus prestaciones, hasta llegar a la nueva versión, el DNSE3. Esta nueva versión cambiaba de paradigma de computación distribuida, pasando a emplear la nube computacional. Por otra parte, el diseño de la arquitectura de servicios se cambió para adaptarse al nuevo entorno en el que funcionaría la aplicación, además de la API empleada por ellos, pasando a ser una API REST. Antes de hablar de esta nueva versión, se introducirán las herramientas empleadas.

Una posible alternativa que entró en juego fue el uso de la nube computacional en lugar del grid, debido a las ventajas que presenta. En primer lugar, la escalabilidad de los recursos de la nube es mucho más sencilla de realizar, además de permitir su automatización. Para realizar este escalado automático se pueden recoger datos provenientes de los diferentes servicios que componen la nube para estimar la demanda de tráfico solicitada. Finalmente, a diferencia de lo que pasó con el grid computacional, la nube está siendo empleada por multitud de usuarios y dispone de numerosa documentación con la cual aprender a manejarla. Es por todo esto que parece más que razonable intentar desplegar una nueva aplicación para el manejo de un entorno de simulación distribuido basado en nube computacional, y de hecho se empezó a desarrollar, esta vez bajo el nombre de DNSE3 [Can12]. Antes de proceder con su presentación, se introducirá el paradigma de la nube computacional y el diseño de servicios RESTful empleado en la aplicación.

2.3. Nube computacional

En los últimos años se ha popularizado con un nuevo paradigma de computación distribuida: la nube computacional. Según el *National Institute of Standards and Technology* (NIST), *la nube computacional es un modelo que permite el acceso remoto a un conjunto de recursos computacionales configurables y compartidos de forma cómoda, desde cualquier lugar y bajo demanda* [MG11]. Dentro de estos recursos podemos encontrar servidores, redes, aplicaciones, espacio de almacenamiento y servicios, entre otras cosas. La nube presenta una serie de características esenciales, entre las cuales se encuentran el aprovisionamiento de recursos bajo demanda, el acceso remoto desde cualquier clase de dispositivo que pueda conectarse a la red, la compartición de los recursos de forma transparente a los consumidores finales, el escalado rápido de los recursos ofrecidos y la monitorización de los servicios y recursos empleados [MG11].

El reciente éxito que tiene la nube se debe principalmente a tres motivos, según nos muestran Armbrust et al. [AFG+10]. El primero de ellos es la capacidad de ofrecer recursos aparentemente infinitos. Cuando se despliega un nuevo servicio o aplicación se tiene que estudiar la posible demanda que va a tener, para evitar la falta o malgasto de recursos utilizados. Aun así, se pueden seguir dando casos en los que los recursos no estén bien ajustados y se siga produciendo un sobredimensionamiento o carencia de ellos. Al trabajar en un entorno de nube, no es necesario realizar este ajuste de recursos de primeras, sino que van ajustando en función de la demanda actual del servicio desplegado.

El segundo y tercer motivo se pueden juntar al estar íntimamente relacionados. El primero de ellos es la eliminación del pago inicial en caso de contratar los servicios de la nube computacional a terceros. El otro motivo permite pagar por uso en lugar de una

cuantía fija que delimitase los recursos accesibles. Estos factores han facilitado la creación de nuevos negocios con capital limitado, que no pueden permitirse el desembolso inicial en equipos, como son las *startups*. Por otra parte, se puede utilizar la nube para ofrecer el aprovisionamiento extra no previsto en el despliegue de los servicios y pagar solamente por el uso que se haga de ellos

2.3.1. Clasificación de la nube computacional

La nube computacional se suele dividir en diferentes clases en función de ciertos parámetros. Los dos principales son el propietario de la nube y el nivel de abstracción [MG11].

Dentro de la primera distinción nos podemos encontrar con la nube pública, la nube privada, la nube comunitaria y la nube híbrida [MG11]. La nube pública abarca aquellos servicios de nube computacional que se ofrecen a los usuarios desde la infraestructura del proveedor, con un coste asociado al uso que se haga de ella, ofreciendo las ventajas económicas previamente nombradas. La nube privada, por otro lado, hace referencia a los centros computacionales privados de las empresas, centros de investigación, etc., empleados para uso interno, pero siempre perteneciendo a una única organización. Si se comparte entre varias organizaciones se trataría de una nube comunitaria. Aunque este modelo presenta ciertas desventajas frente a la nube pública, como puede ser los recursos limitados por las capacidades de las máquinas disponibles, generalmente inferiores a los ofrecidos en las nubes públicas, presenta unas ventajas como son la no dependencia de ellas, que pueden dejar de ofrecer el servicio por causas puntuales; la privacidad de los datos, al quedarse dentro de las propiedades de la institución, y la consolidación de los servidores empleados en la nube. Esta última ventaja hace referencia a la virtualización de los recursos de manera que se optimiza el aprovechamiento del propio *hardware* ya instalado. Una solución intermedia a estas dos es la nube híbrida: consiste principalmente en el uso de una nube privada y solicitar los servicios de la nube pública cuando sea necesario, pagando únicamente por este uso.

En lo que se refiere a la clasificación según el nivel de abstracción, nos podemos encontrar con tres modelos: *Software*-como-Servicio (SaaS, *Software-as-a-Service*), Plataforma-como-Servicio (PaaS, *Platform-as-a-Service*) e Infraestructura-como-servicio (IaaS, *Infrastructure-as-a-Service*). Estas distinciones representan el control que tenemos sobre los recursos ofrecidos. En el caso de SaaS, se nos ofrece el acceso a diferentes aplicaciones que funcionan bajo la propia nube. En este modelo no tenemos ningún tipo de control sobre la propia nube ni sobre la propia aplicación, más allá de su configuración a nivel de usuario y los datos que le hayamos ofrecido [MG11]. Algunos ejemplos de estos servicios son el almacenamiento en nube, con Dropbox[6] y Google Drive[7] como ejemplos más destacables, las aplicaciones ofimáticas en la nube, como son

[6] `https://www.dropbox.com/`
[7] `https://drive.google.com/`

Google Docs[8] y Microsoft Office365[9], y el correo electrónico, pudiendo indicar Gmail[10] y Microsoft Outlook[11].

En el modelo PaaS, se ofrece al consumidor de la nube un entorno en el cual pueda desplegar sus propias aplicaciones. Ahora tenemos control sobre el diseño de la aplicación y de su configuración, pero no tenemos ningún tipo de control ni información sobre los recursos subyacentes empleados. Para facilitarnos el despliegue de dichas aplicaciones, los proveedores de la nube nos ofrecen una serie de lenguajes, librerías y herramientas soportados [MG11]. Dentro de esta categoría podemos encontrar algunas alternativas comerciales como son Microsoft Azure[12] y Google App Engine[13], entre otras.

Finalmente, el modelo IaaS es el nivel de abstracción más bajo de los tres. Permite a sus consumidores administrar los recursos virtuales empleados, tales como las máquinas virtuales, discos duros y redes, entre otros, pero en ningún momento se tiene control sobre la infraestructura subyacente en la que está montada la nube [MG11]. Algunos ejemplos de este modelo pueden ser Amazon Web Services[14] (AWS) y Google Compute Engine[15], además de algunos *middlewares* de gestión de nubes privadas como OpenStack[16] y Eucalyptus[17].

Estos niveles de abstracción se pueden combinar entre sí, de forma que dependiendo de quién sea el consumidor final del servicio en nube se trabajará con un modelo u otro. Por ejemplo, en este TFG se quiere desplegar una aplicación en la nube que vaya a ser utilizada por los alumnos. Mientras que los alumnos sólo trabajar con la aplicación, siendo para ellos un modelo SaaS, en nuestro caso tendremos que gestionar los diferentes recursos de la nube empleados, por lo que trabajaría en un modelo IaaS.

2.3.2. Ejemplos de nubes computacionales

Si queremos utilizar la nube para nuestras aplicaciones, tenemos múltiples alternativas, ya sea utilizando la nube pública o por medio de una nube privada. En esta sección se pretende presentar algunas de ellas, aunque nos centraremos en las soluciones IaaS y PaaS, ya que permiten implementar nuestras propias aplicaciones.

En lo que se refiere a la nube pública, son varias de las grandes potencias del sector las que ofrecen estos servicios. Entre ellas se encuentra Google Inc., quien los ofrece bajo el nombre de Google Cloud Platform. Dentro de esta plataforma ofrece diferentes servicios, cada uno orientado a un propósito distinto como puede ser la propia computación, el almacenamiento en nube o Big Data, entre otros. En lo referente a la

[8] https://docs.google.com/
[9] https://microsoft.office.com/
[10] https://mail.google.com/
[11] https://outlook.live.com/
[12] https://azure.microsoft.com
[13] https://appengine.google.com/
[14] https://aws.amazon.com/
[15] https://cloud.google.com/compute/
[16] https://www.openstack.org/
[17] http://www8.hp.com/us/en/cloud/helion-eucalyptus.html

computación en nube, su servicio más conocido es Google App Engine, una propuesta PaaS de nube orientada al despliegue de aplicaciones web, aunque también dispone de un modelo IaaS bajo el nombre de Google Compute Engine. App Engine se encarga de gestionar los recursos empleados por la aplicación de forma automática, como es el escalado de estos, el registro de acciones de la aplicación y el balance de carga. Las aplicaciones pueden estar escritas en Go, Java, .NET, Node.js, PHP, Python, y Ruby. Empresas como Best Buy Co. Inc.[18], Rovio Entertainment Ltd[19] o Ubisoft Entertainment[20] utilizan este servicio de nube.

Otra gran empresa del sector como es Microsoft también ofrece servicios de nube, bajo el nombre de Microsoft Azure. Aunque ofrece multitud de servicios como bases de datos y máquinas virtuales, el que más destaca es el despliegue de aplicaciones web, ofreciendo un modelo PaaS. Al igual que ocurría con App Engine, en este servicio nos centramos en el desarrollo de la aplicación, que puede estar escrita en java, .NET, Node.js, PHP, Python y Ruby. Está integrado dentro del IDE de Visual Studio y permite testear las aplicaciones antes de desplegarlas, utilizando Azure Emulator. Este servicio es utilizado por empresas como AccuWeather[21], Aviva[22] o Dell[23].

Sin embargo, el servicio más popular dentro de las nubes públicas es el ofrecido por Amazon, con el nombre de Amazon Web Services. Esta plataforma ofrece un modelo IaaS con multitud de servicios orientados a su gestión. Dentro de estos servicios destacan Simple Storage Service (S3), servicio de almacenamiento remoto de objetos para almacenar ficheros de forma independiente, y Elastic Compute Cloud (EC2), encargado del despliegue de los diferentes recursos empleados en la nube, como son las máquinas virtuales o instancias empleadas. Algunas empresas como Netflix[24], Shazam[25], Spotify[26] y Twitter[27] hacen uso de este servicio.

Debido a la gran popularidad del entorno de nube de Amazon, son múltiples los *middlewares* de gestión de nubes privadas e híbridas los que han implementado unos servicios análogos a los ofrecidos por AWS, de forma que se encuentra un ecosistema compatible con dicho entorno y más amigable a la hora de cambiar el gestor empleado. Dentro de los gestores de nubes privadas destacan dos entre todos los disponibles: Eucalyptus y OpenStack.

Si nos centramos ahora en las nubes privadas, la mayoría de los *middlewares* de gestión ofrecen un modelo IaaS y, dada la popularidad de AWS, son compatibles con ese servicio para gestionar nubes híbridas. Entre los diferentes *middlewares* disponibles destacan en particular Eucalyptus y OpenStack.

---

[18] http://www.bestbuy.com/
[19] http://www.rovio.com
[20] https://www.ubisoft.com/
[21] http://www.accuweather.com/
[22] http://www.aviva.com/
[23] http://www.dell.es/
[24] https://www.netflix.com/
[25] http://www.shazam.com/
[26] https://www.spotify.com/
[27] https://twitter.com/

Eucalyptus[28] (*Elastic Utility Computing Architecture for Linking Your Programs to Useful Systems*) es un marco de nube computacional de código abierto, compatible con las APIs de los diferentes servicios ofrecidos por AWS, siendo el primer gestor de nube privada en ofrecer esta compatibilidad en torno a 2008. Los servicios que ofrece este gestor consisten en una implementación de código abierto de los ofrecidos por el entorno de Amazon, utilizando los mismos formatos de peticiones y configuración. Permite ofrecer entornos de nube computacional en cualquiera de los tres niveles de abstracción: Iaas, PaaS y SaaS.

OpenStack[29] por su parte, es otro framework de nube computacional de código abierto también compatible con las APIs de AWS, siendo además el entorno *open-source* que más ha evolucionado en los últimos años. A diferencia de Eucalyptus, en lugar de tener una implementación de los servicios de Amazon, presenta su propia implementación de cada uno de ellos, además de utilizar un modelo de configuración propio. Aun así, mantiene la compatibilidad con los servicios de AWS, hasta el punto que se puede administrar o bien empleando las herramientas que ofrecen o bien utilizando las ofrecidas por Amazon. Su nivel de evolución y prestaciones llego a tal punto que, desde la versión 11.10 de Ubuntu[30], reemplazó a Eucalyptus en el gestor ofrecido en su distribución orientada a nube.

2.3.3. Propuesta de despliegue en el DNSE3

En el desarrollo de este TFG se ha optado por emplear un entorno de nube privada en lugar de un entorno de nube pública. Aunque la nube pública presenta las ventajas de gasto por uso y la mayor capacidad de escalado, la Escuela de Técnica Superior de Ingenieros de Telecomunicación de la Universidad de Valladolid dispone de diversos recursos computacionales que pueden ser aprovechados para la virtualización de recursos en un entorno de nube privada. Gracias su uso se consigue aprovechar los recursos ya existentes, además de que se tiene control sobre la infraestructura subyacente. Por otra parte, las capacidades que ofrece este entorno ofrece, en teoría, los recursos suficientes como para permitir el despliegue de la aplicación para su uso en las asignaturas de "Ingeniería de Teletráfico en Redes Telemáticas" y "Teletráfico" de los grados en Tecnologías y Tecnologías Específicas de Telecomunicación de la Universidad de Valladolid. Sin embargo, para permitir el uso de la aplicación en la mayor cantidad de entornos posibles, se diseñará de forma que se pueda emplear también en entornos de nube pública o híbrida.

En cuanto al nivel de abstracción y *middleware* empleado, se ha optado por el modelo IaaS dado que se requiere de la configuración de las instancias en las que funcionará la aplicación. Dentro de esta configuración se incluye tanto la instalación del simulador empleado como las redes a las que tendrá acceso las diferentes instancias. Por otro lado, de nuevo haciendo referencia a la portabilidad de la aplicación, se hará uso de los servicios de nube ofertados por AWS. Aunque se trate de una nube pública, se ha convertido en los servicios de referencia en esta clase de entornos, por lo que la gran

[28] http://www8.hp.com/es/es/cloud/helion-eucalyptus-overview.html
[29] https://www.openstack.org/
[30] http://www.ubuntu.com/

mayoría de *middlewares* incorporan cierto grado de compatibilidad con los servicios ofertados por AWS. Tal es el caso de Eucalyptus y OpenStack, tal y como se comentó en la sección 2.3.2, que además son los *middlewares* de nube privadas más utilizados. De entre estos dos se ha optado por utilizar OpenStack dado que Eucalyptus presentaba una serie de problemas en sus diferentes servicios y generaba cierta inestabilidad en su uso. Esto fue uno de los motivos por los que Ubuntu cambió de *middleware* en su distribución orientada a nube.

2.4. REST

Tanto el DNSE como el DNSE3 presentan una arquitectura orientada a servicios (SOA, *Service Oriented Architecture*) que consiste en la repartición de las funcionalidades a desempeñar por la aplicación. Para efectuar dichas funcionalidades será necesario que los servicios se comuniquen entre ellos para realizar un trabajo coordinado. La ventaja que presenta esta clase de arquitectura es la facilidad que presenta para construirla, mantenerla y extenderla, al ser cada uno de los servicios independiente del resto en lo que se refiere a desarrollo y preocuparse cada uno de ellos en la forma de acceder a sus funcionalidades más que en la forma en la que trata y procesa los datos [Vin07]. Para acceder a estas funciones se necesita establecer la interfaz que ofrecerá cada uno de estos servicios, es decir, los métodos y los datos necesario que emplean cada uno de ellos. Por otra parte, se debe separar la interfaz de su implementación, permitiendo a los consumidores del recurso ignorar cómo realiza cada una de sus funciones y centrar en las llamadas soportadas.

Para diseñar estos servicios, se seguía hasta hace relativamente poco las pautas establecidas por SOAP, en el que se utilizaban unos métodos propios del servicio que se enviaban encapsulados dentro de un documento XML en el cuerpo de las peticiones HTTP [CDK+02]. Sin embargo, en los últimos años se está abandonando a favor de un nuevo estilo de diseño de servicios web: REST. La idea básica detrás de este estilo es utilizar la propia capa de aplicación HTTP, con sus métodos, cabeceras y códigos de respuestas, para ofrecer la interfaz del servicio. En esta sección nos centraremos en este último.

2.4.1. Descripción

REST es un estilo de diseño de servicios en el que se propone exponer su funcionalidad a través de una interfaz única y universal, expuestas a través de unos recursos que tienen un identificador propio [RR07]. Estos recursos hacen referencia a las entidades más importantes del servicio. Para hacer uso de la interfaz ofrecida por el servicio se produce un intercambio de representaciones de los recursos, que muestran el estado actual del recurso. En esta clase de servicios se hace un gran uso de las representaciones, pudiendo estar en multitud de formatos distintos, pero totalmente compatibles, de forma que se pueda trabajar con el que resulte más adecuado. En el caso de los servicios web, se propone por su parte la especificación RESTful, donde se utiliza la capa de aplicación HTTP para mostrar la interfaz de los recursos, que utilizan el espacio de direcciones URI (*Universal Resource* Identifier) para establecer sus identificadores únicos de acceso[RR07].

Las características más relevantes que ofrece este estilo de diseño de servicios web se pueden agrupar en el listado siguiente [RR07]:

- El servicio es direccionable (*addresable*), es decir, los datos más relevantes del servicio son accesibles a través de sus recursos. Para conseguir esto, se requiere establecer una relación unívoca entre los recursos y sus URIs, evitando el acceso a diferentes recursos por medio de la misma URI. Gracias a esto, se consigue que los diferentes clientes puedan acceder a nuestro servicio utilizando el mismo tipo de acceso y pueda ser compartido y explotado de la forma que se estime oportuna.
- El servicio carece de estado, es decir, no mantiene ningún tipo de información relacionada con peticiones previas que se hayan realizado, por lo que son cada una de ellas independientes entre sí. Los clientes de dicho servicio deberán de mantener la información necesaria para realizar las futuras peticiones e incluirlas en la petición. Gracias a esta característica se pueden escalar esta clase de servicios, ya que no se necesita coordinar las peticiones entre los diferentes servidores.

  No hay que confundir el estado del cliente, que define el punto de operación en el cual se encuentra tras una serie de peticiones a su interfaz y que es mantenido por la aplicación del cliente, con el estado del servicio, que definen su capa de persistencia, es decir, los datos permanentes que utilizará en su ejecución.
- El servicio está completamente conectado, de forma que se puede ofrecer una ruta que conecte a los diferentes recursos por medio de enlaces enviados en sus representaciones. Aunque se puedan establecer pautas acerca de la generación de las URIs de los diferentes recursos, estas dejan de ser necesarias si se ofrece una forma de acceder a ellos, mostrando además la relación que se establece entre los diferentes recursos.
- El servicio presenta una interfaz uniforme a lo largo del servicio y entre otros servicios que presenten una API REST. Al emplear la capa de aplicación HTTP para establecer los métodos soportados por la API, con solo exponer los recursos accesibles se podría llegar a manejar el servicio sin tener ningún tipo de información adicional, a excepción de los datos enviados en las peticiones, donde es necesario conocer el formato y datos empleados. Dado que los métodos HTTP son limitados en número y tienen un significado particular, se conocen de antemano las posibles operaciones que soportan sin necesidad de tener que buscar la especificación de cada uno de ellos.

2.4.2. REST vs SOAP

Anteriormente se comentó que otro de los diseños de servicios web más empleados anteriormente era SOAP, que hacía uso del intercambio de mensajes XML donde se describían tanto los métodos invocados del servicio como sus parámetros de entrada. En esta sección vamos a comparar las características de estos paradigmas de diseños para mostrar los puntos fuertes y flaquezas de cada uno. Para ello, nos centraremos en el diseño de los servicios, la interfaz y el intercambio de mensajes [WT12].

Dado que SOAP surgió mucho antes que REST, se ha podido trabajar más tiempo con él y hay multitud de herramientas de desarrollo que ayudan en su diseño. Sin embargo, como REST utiliza la propia interfaz de HTTP, la complejidad de su diseño reside en la estructuración de los recursos y sus representaciones, que es mucho menor que en la estructuración de los mensajes empleados en SOAP [WT12]. Por otro lado, en caso de realizar modificaciones en el servicio, en el caso de los servicios REST el impacto es menor al no requerir grandes cambios en el lado del cliente, mientras que con SOAP se debe de ajustar a la nueva estructura empleada.

Tanto SOAP como REST utilizan HTTP para entablar las conversaciones, pero, mientras REST lo utiliza para establecer su interfaz, SOAP lo utiliza como capa de transporte en la que colocar el mensaje en sí, a pesar de que es una capa de aplicación [WT12]. Dado que la interfaz real empleada reside en el mensaje, los clientes deben tener acceso al fichero WSDL en el cual se documenta la propia API y adaptarse a su uso. Por su parte, REST utiliza las funciones permitidas por HTTP, siendo los métodos GET, PUT, POST y DELETE los más típicos, al igual que el resto de sus elementos, como son las cabeceras y los códigos de respuesta. Como todos ellos tienen un significado y propósito bien definido y establecido [FGM+99], se podría utilizar cualquier cliente HTTP para consumir el servicio. Por otra parte, como los servicios REST ofrecen su funcionalidad desde los distintos recursos, se necesita una gran cantidad de URIs (*Uniform Resource Identifier*) para su acceso, mientras que en SOAP se pueden tener unas pocas URIs.

2.4.3. Propuesta de exposición de los servicios en el DNSE3

En el desarrollo del DNSE3 se ha optado por emplear un diseño REST en los servicios que implementa. Los motivos de esta decisión son varios. El primero de ellos es que el uso que hace de las representaciones es más eficiente en comparación con SOAP. Permite emplear cualquier tipo de formato y, por lo general, son de menor tamaño que las empleadas por SOAP. Si tenemos en cuenta que el DNSE3 debe estar orientado para su suo en diferentes entornos de nube computacional, en el caso de las nubes públicas, donde se puede cobrar por el tráfico interno entre las máquinas empleadas puede ser importante para minimizar su coste. Otro de los motivos de su uso es que los gestores de nube computacional, como es el caso de AWS, también ofrece una API REST, por lo que se consigue mantener un ecosistema de servicios homogéneo. Una de las ventajas que ofrece es en el intercambio de peticiones entre los servicios, donde se reduce el número de transformaciones efectuadas en las representaciones intercambiadas.

Los otros motivos de su elección están relacionados íntimamente con su implementación y despliegue. Por una parte, los servicios carecen de estado. Tal y como se vio anteriormente, hace que las diferentes peticiones sean independientes entre sí y, en caso de escalar el servicio utilizando nuevos servidores, no será necesaria la sincronización de las operaciones efectuadas. Por otra parte, estos servicios presentan compatibilidad total con los diferentes clientes HTTP disponibles actualmente, como son los navegadores web. Para realizar el mantenimiento de estos servicios no será necesario desarrollar un cliente específico, sino que podemos hacer uso de ellos.

Para el desarrollo de los servicios, se hará uso de un *framework* que permita el desarrollo de servicios REST. Las restricciones que se establecen en su elección se

mantienen igual que para el resto de herramientas: que no requieran de licencia para su uso para permitir su libre distribución. De las diferentes alternativas consideradas en el estudio de la aplicación[Can12], entre las que se encontraban ASP.NET[31] Web API para ASP.NET y JAX-RS[32] para Java, se ha optado por el uso de Restlet Web API Framework[33] para el lenguaje de programación Java. Este *framework* permite desarrollar tanto los propios servicios como los clientes de estos, de tal forma que se pueden implementar de forma conjunta en los servicios de la aplicación, además de ser compatible con las diferentes ediciones de Java, entre las que se encuentra Java SE/EE, GWT, GAE, Android y OSGi.

2.5. **DNSE3**

Tras haber presentado las tecnologías utilizadas por esta nueva versión, ya podemos hablar de ella. Esta aplicación, a diferencia de su predecesor, hace uso del entorno de nube computacional para adaptar los recursos utilizados por ella a la demanda que tenga que atender. Antes de realizar su implementación, se realizó un estudio sobre su viabilidad en esta clase de entornos [Can12]. En dicho estudio planteó una arquitectura basada en servicio, capaz de cubrir todas las funciones necesarias para el correcto funcionamiento de la aplicación. Entre dichas funciones se encontraba la capacidad de procesar en paralelo las simulaciones solicitadas por los usuarios, siendo el propósito principal de la aplicación, poder escalar las capacidades de la aplicación en función de la demanda de simulaciones y poder monitorizar dicha demanda, además de permitir una gestión de las simulaciones a procesar. Estos servicios presentaban un diseño RESTful en su arquitectura e interfaz.

En la Figura 2.7 se presenta la arquitectura propuesta. Se dividía en 8 servicios, los cuales se podían clasificar en función del acoplamiento que tenían con la aplicación final. Dicha clasificación se divide en:

- Servicios propios de la infraestructura: estos servicios presentan un muy bajo acoplamiento con la aplicación DNSE3, pero muy alto acoplamiento con la infraestructura de la nube computación, encargándose de su gestión. Estos servicios, además, son más propicios a ser empleados por otra clase de aplicaciones basadas en nube computacional. Dentro de esta categoría se encuentran los siguientes servicios:
    - Servicio de almacenamiento: Este servicio contiene un repositorio de ficheros genérico y autoescalable que permite la creación, modificación, visionado y eliminación de ficheros y directorios. En este servicio se almacenarán tanto los scripts de los modelos de las simulaciones de los usuarios como los resultados obtenidos tras su ejecución y los informes generados con estos resultados.
    - Servicio de monitorización: Este servicio permite la gestión de métricas relacionadas con el uso y funcionamiento del resto de servicios, de forma que se puedan crear, consultar y eliminar tanto las métricas como los

[31] http://www.asp.net/web-api

[32] http://java.net/projects/jax-rs-spec

[33]

valores que toman dichas métricas a lo largo del tiempo. Entre las métricas que se comentan en su diseño se encuentran el número de simulaciones solicitadas y endientes de ejecutar y el número de instancias encargadas de procesar las simulaciones a lo largo del tiempo.

  - Servicio de escalado: Este servicio se encarga de realizar el escalado de los recursos ofrecidos por la nube computacional, ya sea hacia arriba, creando nuevos recursos, o hacia abajo, eliminandolos. Este escalado se realiza utilizando unas métricas, basadas en los valores obtenidos de las métricas del servicio de monitorización, especialmente en el número de simulaciones pendientes.

- Servicios generales de la aplicación: A diferencia de los servicios propios de la infraestructura, estos servicios tienen cierto acoplamiento con la aplicación DNSE3, aunque realmente bajo, pero no tienen ningún tipo de relación con la nube computacional. Dentro de esta categoría podemos encontrar los siguientes servicios:
  - Servicio de colas: Este servicio se encarga de gestionar un sistema de colas de carácter general que permite el procesamiento y ejecución de forma asíncrona de las tareas registradas en el sistema. En este sistema de colas se almacenan las diferentes simulaciones que han pedido los usuarios, procedentes ya sea de simulaciones individuales como de la división de las simulaciones de barrido de parámetros.
  - Servicio de simulación: Este servicio se encarga de ejecutar los modelos de simulación utilizando para ello el simulador ns-3. Estos modelos los recogerá del servicio de colas y se le podrá notificar del progreso de la simulación, indicando su correcta ejecución o los fallos producidos. Una vez terminada esta ejecución se debe de encargar de almacenar los resultados obtenidos de la forma adecuada en el servicio de almacenamiento.
  - Servicio de estadística: Este servicio permite la realización de una serie de métodos estadísticos, previamente implementados, para la obtención de estadísticas relacionadas con aquellos datos que recibe a la entrada de las peticiones.
- Servicios propios de la aplicación: Estos servicios son aquellos que tienen el mayor nivel de acoplamiento con la aplicación DNSE3, al ser sus funciones muy concretas y formar parte del núcleo de la aplicación. En esta categoría podemos encontrar los siguientes servicios:
  - Servicio de orquestación: Este servicio es el encargado de coordinar las diferentes funcionalidades a desempeñar por la aplicación DNSE3. Entre sus diversas tareas se encuentra la gestión de las cuentas de usuario y de las sesiones de estos, administrar las simulaciones recibidas por los usuarios, realizando además la división de las simulaciones de barrido de parámetros y el filtrado y formateado de los resultados obtenidos.
  - Servicio de informe: Este servicio se encargará de generar un documento en el que se presenten los resultados obtenidos tras la ejecución de las simulaciones, pudiendo contener diversos cálculos realizados sobre estos.

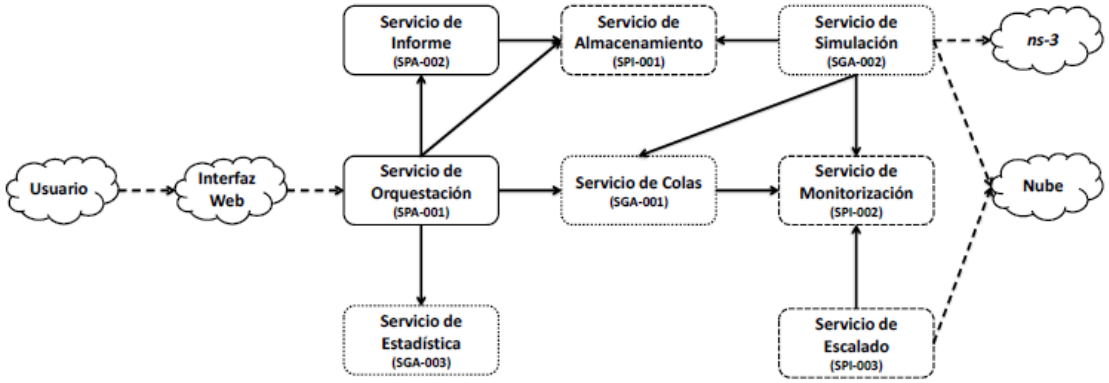

Figura 2.7: *Diagrama de arquitectura inicial de servicios de la aplicación DNSE3 [Can12].*

2.5.1. Críticas a la arquitectura

El diseño que se hizo de esta aplicación fue realizado cuando el nivel de desarrollo de los sistemas de gestión de la nube computacional no era tal alto como el que podemos encontrar actualmente. Muestra de ello son los servicios propios de la infraestructura, que son a día de hoy de uso habitual cuando se trabaja con la nube y ya presentan una solución integrada. En la gran mayoría de *middlewares* de gestión de nube computacional se encuentra una solución para dichas funcionalidades adaptada a dicho entorno de nube, por lo que sería más que adecuado intentar adaptar su uso para nuestra aplicación, a pesar de la restricción al *middleware* empleado.

En el caso del servicio de almacenamiento, de forma general se presentan 2 alternativas que ofrecen un almacenamiento persistente y escalable: el almacenamiento de objetos y el almacenamiento de bloques. En el primer caso, el almacenamiento de objetos permite el almacenamiento de objetos persistente a través de la red en unos contenedores de objetos. Por el contrario, el almacenamiento de bloques permite el manejo de unidades de almacenamiento virtuales que se pueden utilizar a modo de discos duros. Estas unidades de almacenamiento deben están montadas en una de las instancias de la nube y solo puede ser utilizada por una de ellas a la vez. Las ventajas que presenta el almacenamiento de bloques frente al almacenamiento de objetos son el uso de un sistema de ficheros, facilitando la navegación por la estructura de ficheros, y la posibilidad de portar el dispositivo de bloques a otro entorno de nube computacional de forma prácticamente inmediata. Ejemplos de estos servicios son el servicio Swift de Openstack, Object Storage de Eucalyptus y Amazon Simple Storage Service en AWS, en lo referente al almacenamiento de objetos, y el servicio Cinder de Openstack, Eucalyptus Block Storage de Eucalyptus y Elastic Block Storage de AWS.

En lo referente a la monitorización del uso de los servicios en la nube, se planteó inicialmente para controlar el uso que se hacía de la infraestructura y cobrar por ello. Sin embargo, posteriormente se extendió su uso para poder recoger las métricas que fuesen necesarias y controlar las aplicaciones y recursos empleados en la nube. Dentro de esta clase de servicios se incluye tanto el registro de las métricas y trazas de actividades, como el uso de alarmas que avisen del cumplimiento de una serie de reglas relacionadas con las

métricas obtenidas. Ejemplos de este servicio es el servicio Ceilometer de Openstack y CloudWatch de Eucalyptus y AWS.

Finalmente, en los entornos de nube computacional, a pesar de poder gestionar los diferentes recursos empleados de forma manual, es interesante su gestión automática. Nos permite la creación de una serie de recursos de forma conjunta al trabajar todos ellos de forma coordinada, o incluso establecer grupos de recursos que compartan las mismas características, pero con un número de recursos variables. Al igual que pasaba con los servicios anteriores, se incluyen implementaciones de este servicio, como es Heat de OpenStack y CloudFormation de Eucalyptus y Amazon Web Services.

El hecho de aprovechar estos servicios ya integrados dentro de la propia nube nos presenta una serie de ventajas. En primer lugar, son servicios bien documentados, con un soporte por detrás tanto de sus propios desarrolladores como de los usuarios que les utilizan, por lo que es más fácil solventar cualquier problema que surja en estos casos que si tenemos que desarrollar estos servicios, además de que reciben actualizaciones sucesivas que mejoran y amplían su funcionalidad. Por otra parte, se reduce su tiempo previsto paa el desarrollo e implementación, lo que nos ofrece más tiempo de desarrollo y testeo del resto del sistema.

Si ahora nos fijamos en el resto de servicios, la gran mayoría de hechos tienen una funcionalidad muy específica y centrada en un aspecto de la aplicación. Sin embargo, en el servicio de orquestación tenemos una función principal como es la coordinación del resto de funcionalidades de la aplicación, pero también tenemos la gestión de las cuentas de usuario. Aunque es más que razonable que sea el servicio de orquestación el que se encargue de dicha funcionalidad, dado que es el punto de acceso desde la red externa al entorno de la aplicación, supone una sobrecarga de trabajo. Aunque se hablará de este tema más adelante en el rediseño de la aplicación, se intentará separar esta funcionalidad en un servicio independiente.

Otro caso que merece la pena destacar es el servicio de estadística. Se diseñó esencialmente para poder aplicar una serie de métodos estadísticos a unos datos que recibiese como entrada de las peticiones. Su uso estaba orientado para el servicio de Monitorización y Escalado, para realizar los cálculos necesarios sobre las métricas y ajustar los recursos de la nube de la forma adecuada, y para el servicio de Informe, para poder trabajar con los resultados obtenidos de las simulaciones y ofrecer dichos cálculos a los usuarios para su análisis. Dado que los servicios de Monitorización y Escalado ya presentan una solución integrada dentro de la propia nube, dejarían de ser clientes del servicio de Estadística y sería lógica su integración dentro del servicio de Informe, al ser su único cliente aparente.

2.6. **Conclusiones**

A lo largo de este capítulo se han ido presentando las diferentes tecnologías que se utilizarán en el desarrollo del DNSE3. En todos los casos, se ha intentado utilizar herramientas que ofrezcan su uso sin licencia, ya que el fin último de la aplicación es que se pueda utilizar en el mayor número de escenarios que puedan hacer uso de las simulaciones de redes telemáticas. En el caso del gestor de nube, además, se ha seguido

la implementación ofrecida por AWS, dada su relevancia en entornos de nube computacional.

En lo que se refiere a la aplicación en sí, se ofrece una arquitectura basada en servicios que permite, por un lado, un mejor aprovechamiento de los recursos de la nube al especializarse en las tareas que deben llevar a cabo. Por otra parte, al utilizar este tipo de arquitectura, se puede reutilizar cada uno de los servicios por separado en futuros proyectos, aportando un factor de reusabilidad y portabilidad. Sin embargo, se debe de realizar modificaciones a su diseño para integrar los servicios ofrecidos por los entornos de nube, no existentes en su diseño.

En lo que resta del documento se presentará el trabajo realizado a partir de esta arquitectura, empezando por el rediseño que se ha estado anticipando en los párrafos anteriores y que se complementará con un entorno de pruebas de la propia aplicación para testear su núcleo, es decir, el procesamiento adecuado de las simulaciones solicitadas y la recuperación de los resultados obtenidos tras su ejecución.

Capítulo 3. Arquitectura del DNSE3

En el capítulo anterior se ha mostrado las ventajas que ofrecen las simulaciones dentro de los entornos académicos, así como sus ventajas frente a otros modelos de enseñanza. Sin embargo, cuando se quieren utilizar los barridos de parámetros, el tiempo necesario para su correcta ejecución puede ser excesivo para la duración típica de las sesiones de laboratorio.

Para paliar este problema, se han diseñado diversas aplicaciones que permiten su ejecución en paralelo, ya sea empleando la computación en paralelo o la computación distribuida. Este es el caso de la aplicación DNSE, que, tal y como se indicó en la sección 2.2, permitía la ejecución de las simulaciones en un entorno de grid computacional. Sin embargo, dicha aplicación presentaba una serie de problemas, como su baja escalabilidad automática y uso de la misma infraestructura, que propiciaron la búsqueda de alternativas en su diseño.

Una de las nuevas propuestas fue el DNSE3, que hace uso de la nube computacional para ofrecer un entorno de simulación de redes telemáticas fácilmente escalable a la demanda de la aplicación. Esta aplicación presenta una arquitectura SOA en la que se proponen una serie de servicios que implementan alguna de la funcionalidad del sistema y trabajan de forma conjunta. Cada uno de estos servicios se ha diseñado según lo establece el diseño de servicios RESTful, ofreciendo una API REST accesible desde sus diferentes servicios.

Esta arquitectura se diseñó integrando una serie de funciones encargadas de la gestión de la propia infraestructura de la nube que a día de hoy se incluyen en la mayoría de *middlewares* de gestión de nube computacional. En este capítulo, partiendo de la arquitectura original del DNSE3, se realizará un rediseño de esta arquitectura, integrando los servicios ofrecidos por la nube para cubrir aquellas funciones necesarias en la aplicación. Por otra parte, se hará un rediseño parcial del resto de servicios para distribuir la carga de trabajo de cada uno de ellos y permitir la introducción de herramientas de terceros que integren algunas de dichas funciones.

En primer lugar, se mostrará la arquitectura final de los diferentes servicios que componen la aplicación, para después mostrar el diseño realizado en ellos, Se hará especial hincapié en la capa de recursos ofrecida y los métodos HTTP soportados por cada uno de estos.

Finalmente, se mostrarán los diferentes servicios que ofrece la propia nube para realizar las diferentes tareas de gestión. De forma adicional, se detallará la configuración realizada en ellos orientada a su uso en el DNSE3.

3.1. Diseño de la arquitectura

La arquitectura de la aplicación DNSE3, siguiendo el diseño SOA, se compondrá de una serie de servicios que trabajarán de forma coordinada para llevar a cabo las diferentes tareas. A continuación, se enumerarán los servicios que componen esta arquitectura, clasificándoles en función del nivel de acoplamiento que tengan con el DNSE3. En la Figura 3.1 se muestran cada uno de ellos, además de las comunicaciones producidas entre ellos.

- Servicios genéricos de la aplicación (SGA): Estos servicios presentan un bajo acoplamiento con el DNSE3, al realizar las tareas de propósito general, pero no tiene ningún tipo de relación con la nube computacional. Dentro de esta categoría nos podemos encontrar los siguientes servicios:
    - Servicio de colas (SGA-01): Este servicio gestiona y mantiene un sistema de colas de carácter general que permite el procesamiento y ejecución de forma asíncrona de las tareas registradas en él. En el caso del DNSE3, se almacenan las diferentes simulaciones solicitadas por los usuarios, ya sean individuales o procedentes de la división de una simulación de barrido de parámetros.
    - Servicio de simulación (SGA-02): Este servicio se encarga de ejecutar los modelos de simulación, utilizando para ello el simulador ns-3. Los modelos se recogerán del servicio de colas, al que se le notificará sobre el éxito o no en su procesamiento. Deberá de almacenar correctamente los resultados obtenidos tras su ejecución en el servicio de almacenamiento.
    - Servicio de estadística (SGA-03): Este servicio permite la realización de una serie de métodos estadísticos, previamente implementados, para la obtención de estadísticas relacionadas con los datos recibidos como entrada de las peticiones.
    - Servicio de gestión de sesiones (SGA-04): Este servicio se encarga de mantener el listado de los usuarios registrados en la aplicación, junto con sus datos personales y permisos dentro de ella, además de gestionar sus sesiones de trabajo, donde se validarán las operaciones solicitadas.
- Servicios propios de la aplicación (SPA): Estos servicios presentan un nivel de acoplamiento elevado con el DNSE3, al ser sus funciones muy concretas y formar parte del núcleo de la aplicación. En esta categoría podemos encontrar:
    - Servicio de orquestación (SPA-01): Este servicio se encarga de coordinar las diferentes tareas a desempeñar por la aplicación. Para ello, se encarga de la gestión de las peticiones de los usuarios, ya sean de simulación o de recuperación de datos, y de distribuirlas entre el resto de servicios para su correcta ejecución.
    - Servicio de informe (SPA-02): Este servicio se encarga de recoger los datos obtenidos tras la ejecución de las simulaciones y prepararlos para su consumo por parte de los usuarios. Entre sus tareas se incluye el formateo

de dichos resultados y la generación de un documento de tipo variable (PDF, HTML, DOC) en el que se agrupen junto con posibles cálculos que se realicen con ellos.

- Servicios propios de la infraestructura (SPI): Estos servicios presentan un alto nivel de acoplamiento con el entorno de la nube computacional. Sus funciones se centran en la gestión y administración de los recursos ofrecidos por la infraestructura. A día de hoy, la mayoría de *middlewares* de gestión de nube implementan un servicio que desempeña dichas funciones, por lo que se intentará aprovechar en la medida de lo posible. Los servicios empleados por el DNSE3 que entran dentro de esta categoría son:
  - Servicio de almacenamiento (SPI-01): Este servicio deberá de ofrecer un repositorio de ficheros de propósito general de acceso remoto y autoescalable. Deberá permitir el almacenamiento, modificación, visionado y eliminación de los modelos de simulación enviados por los usuarios, los resultados obtenidos tras la ejecución de las diferentes simulaciones, los ficheros generados tras el procesamiento de los resultados y los ficheros de configuración empleados por los diferentes servicios.
  - Servicio de monitorización (SPI-02): Este servicio se encargará de gestionar las diferentes métricas relacionadas con el uso que se hace de los diferentes servicios del DNSE3 para gestionar los recursos empleados por la aplicación. Dentro de las posibles métricas a utilizar se pueden encontrar el número de simulaciones solicitadas y pendientes de procesar, al igual que el número de instancias encargadas de dicho procesamiento.
  - Servicio de escalado (SPI-03): Este servicio se encargará de adecuar los recursos empleados por la propia aplicación en función de la carga de trabajo que deba procesar. Para realizar este escalado se emplearán una serie de reglas que harán uso de las métricas obtenidas del servicio de monitorización.

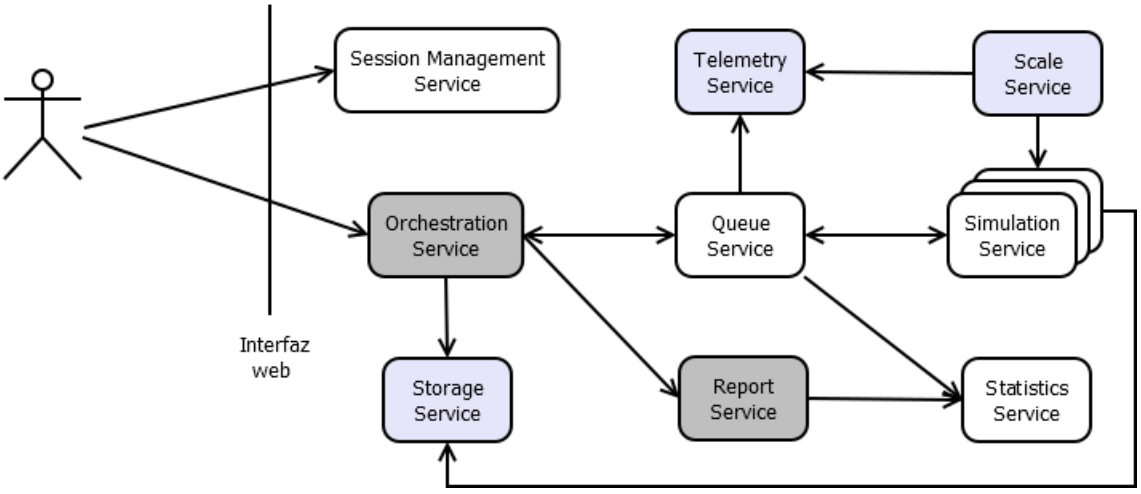

Figura 3.1: *Arquitectura de servicios de la aplicación DNSE3.*

El diseño de esta arquitectura mantiene prácticamente en su totalidad la propuesta en el diseño original [Can12]. Los cambios realizados en esta hacen referencia a la separación de la gestión de las sesiones de los usuarios en un servicio

aparte y ligeros cambios en el diseño interno de algunos de los servicios, sin olvidarnos de la integración que se realizará de los servicios de la propia nube. En las siguientes secciones se profundizará en dichos cambios.

3.2. Servicios de la aplicación

Cada uno de los servicios de la aplicación (tanto genéricos como propios) deben presentar una API REST para la exposición de sus funciones, además de seguir la filosofía RESTful. Para su diseño se seguirá la arquitectura ROA (*Resource Oriented Architecture*), resultando en una arquitectura basada en diferentes recursos a los cuales se tendrá acceso. Se ha optado por este tipo de diseño por diferentes razones, mencionadas en la sección 2.4.3. De forma resumida, este diseño permite el desarrollo de un sistema homogéneo de nube computacional, en el que se realizará una transformación mínima en el intercambio de representaciones de los servicios de la aplicación. Además, estos servicios se podrán escalar sin necesidad de sincronizar el estado de cada una de las instancias del servicio. Por último, se podrá hacer uso de navegadores web y otros clientes HTTP para testear su funcionamiento.

Otro aspecto importante de su diseño es que se ha realizado, en la medida de lo posible, totalmente independiente de la tecnología empleada. La única restricción que se le ha puesto es la inclusión de una API REST. Por lo demás se puede utilizar con cualquier lenguaje de programación o *framework* que permita presentar esta interfaz. Además, también es independiente del entorno de nube computacional en el que será lanzada la aplicación o el simulador que utilizará. Este tipo de diseño permite tener una implementación totalmente libre y que permita su adaptación a las condiciones en las que deba funcionar.

En esta sección se presentarán las funciones ofrecidas por cada uno de los servicios y se discutirá el diseño de estos, centrándonos especialmente en los recursos y métodos expuestos.

3.2.1. Servicio de colas

Este servicio se encarga de la gestión de una cola de propósito general que permita la transmisión asíncrona de tareas entre diferentes servicios en un sistema basado en servicios. Siguiendo una planificación de “robo de trabajos” (*work stealing*) las tareas podrán ser tomadas por un “trabajador” (*worker*). Para evitar la inanición, el trabajador deberá refrescar periódicamente su tarea indicando que sigue procesándola.

El diseño de este servicio deberá de permitir su inclusión sencilla en otros sistemas que requieran de un sistema de colas para compartir las tareas a ejecutar. Para ello, el diseño de estas tareas se realizará por medio de una clase abstracta que defina una serie de atributos y métodos mínimos para su manejo, con la opción de incluir nuevos atributos a mayores. En el caso del DNSE3, las tareas transmitidas son las simulaciones procedentes del servicio de orquestación para su posterior ejecución por parte del servicio de simulación.

Las funciones que debe desempeñar este servicio y que ofrecerá a través de su API REST son las siguientes:

- Consultar una o todas las tareas que se encuentran dentro de la cola.
- Añadir nuevas tareas para su posterior procesamiento.
- Mostrar la siguiente tarea que esté pendiente de procesar.
- Actualizar el estado de cada una de las tareas.
- Eliminar las tareas de la cola que hayan sido procesadas o presenten algún error en su ejecución.

Todas estas funciones se agrupan en el diagrama de casos de uso mostrado en la Figura 3.2.

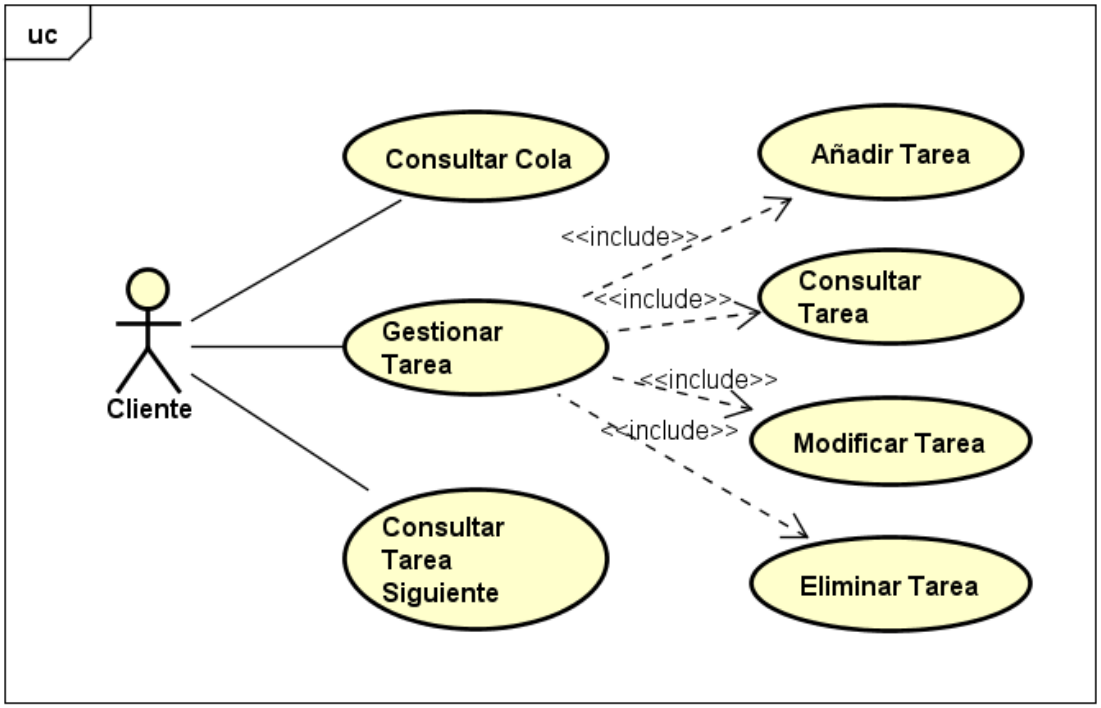

Figura 3.2: *Diagrama de casos de uso del servicio de colas.*

En lo referente a la actualización del estado de las tareas, se permitirá modificar cualquiera de sus atributos, aunque entre los más relevantes se encuentra su estado de ejecución, pudiendo ser uno entre los 4 siguientes: *WAITING*, cuando la tarea este esperando a ser ejecutada; *PROCESSING*¸ cuando se esté ejecutando; *FINISHED*¸ cuando se haya ejecutado correctamente, y *MALFORMED*, cuando se haya producido algún error en su ejecución.

Otro de los atributos más importantes que se debe actualizar es la fecha de vencimiento de la reserva de ejecución. Anteriormente se comentó que se utiliza *work stealing* para planificar la asignación de las diferentes tareas de ejecución, al igual que se deberá refrescar dicha asignación de forma periódica para evitar el abandono de su ejecución. La fecha de vencimiento define el instante a partir del cual la tarea pasará a estar disponible para el resto de *workers*. La extensión temporal de dicha reserva se ha establecido igual al tiempo necesario para realizar 2 renovaciones, de forma que no se tengan en cuenta posibles fluctuaciones en el tiempo empleado en la reserva.

Para el correcto desempeño de las funciones anteriores, este servicio se comunicará con los siguientes servicios:

- El servicio de orquestación (SPA-01) solicitará la inserción de nuevas simulaciones a procesar, facilitando los datos necesarios para la recuperación de los *scripts* de los modelos empleados y la correcta recogida de los resultados obtenidos. También pedirá la eliminación de aquellas simulaciones que se hayan completado correctamente o hayan presentado algún error en su ejecución. En este último caso, se solicitará también la eliminación del resto de simulaciones relacionadas para evitar el consumo de recursos innecesario. Estas eliminaciones se producirán tras recibir las actualizaciones de estado de las simulaciones.
- El servicio de simulación (SGA-02) pedirá el acceso a la siguiente tarea pendiente de procesar. Una vez que comprueba la integridad de esta, la reservará para su ejecución. Mientras la esté ejecutando, enviará actualizaciones periódicas para notificar de que aún continúa trabajando con ella.
- El servicio de monitorización recogerá las métricas de la evolución de la cola, incluyendo el número de tareas pendientes de ser procesadas.

Para acceder a estas funciones, el servicio de colas presenta una arquitectura ROA compuesta por los siguientes recursos:

- *Queue*: este recurso representa a la cola de tareas gestionada por el servicio. Su único atributo es el listado de las distintas tareas que recibe, que mantiene su orden de llegada.

  Los métodos HTTP soportados por este recurso son GET, que devuelve el listado de las tareas almacenadas, y POST, que permite la inserción de nuevas tareas a la cola. Tras la llamada al método POST se devuelve la URI de acceso a la tarea recién creada.
- *Task*: este recurso representa cada una de las tareas contenidas dentro del servicio. Entre sus atributos se encuentran la URI de acceso a los ficheros de entrada de la tarea, los parámetros de entrada necesarios para su ejecución y una serie de datos que permiten gestionar la forma en la que se recogerán los resultados obtenidos tras su ejecución. También incluye los tiempos de renovación, que establecen los periodos en los que se tienen que renovar las tareas y dar evidencias de su procesamiento activo, y la fecha de expiración de ejecución, equivalente a 2 tiempos de renovación y marcan el instante a partir del cual se considera que ha surgido un imprevisto con el encargado de su procesado y vuelve a estar disponible para su recogida.

  Los métodos HTTP soportados por este recurso son GET, que devuelve el estado actual de la tarea, PATCH, que permite actualizar su estado, y DELETE, que elimina la tarea del servicio. Se ha optado por el método PATCH en lugar de PUT para la actualización del estado de la tarea dado que PATCH, según está diseñado, permite la modificación selectiva de los atributos del recurso [DS10], mientras que con PUT se tiene que actualizar todos los atributos, aunque sea para mantenerlos idénticos. Esto además supone que los documentos enviados en las peticiones resultan más livianos en el caso de utilizar PATCH.

Debido a las continuas actualizaciones que van a recibir las tareas, este recurso hará uso de los *ETags* [FR14] para evitar el solapamiento de modificaciones del recurso.

- *NextTask*: este recurso representa a la siguiente tarea pendiente de ser procesada. Permite el acceso al recurso *Task* al cual está haciendo referencia, por lo que es un recurso autogenerado.

  Solo soporta el método HTTP GET, que permite acceder al recurso *Task* que contiene la tarea a procesar.

En la Figura 3.3se muestra el diagrama de recursos del servicio, mostrando la dependencia de cada uno de ellos.

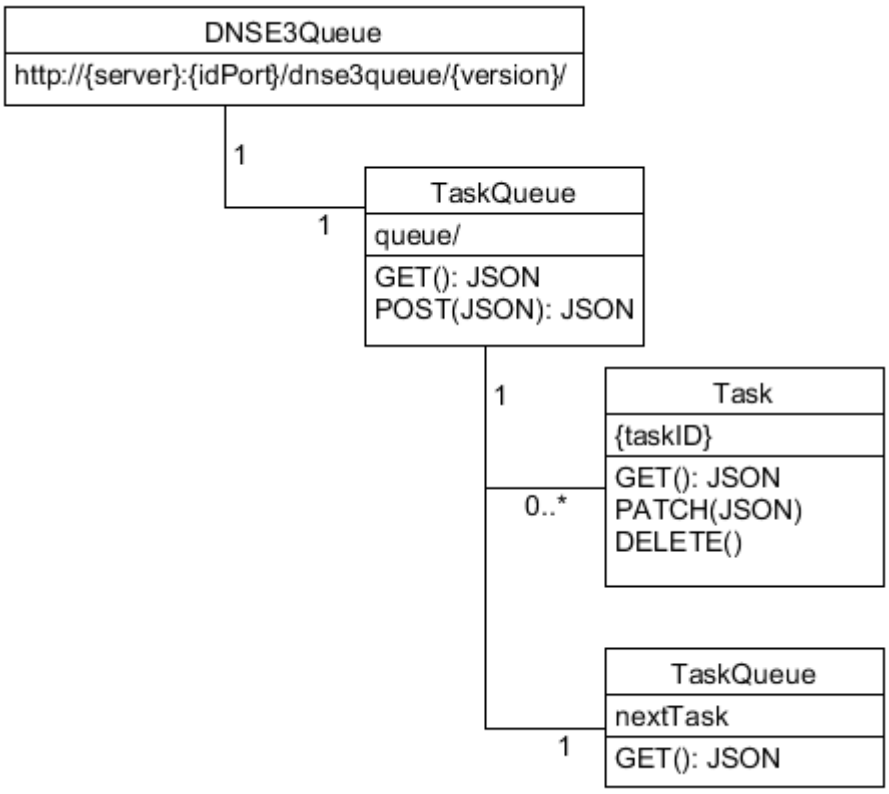

Figura 3.3*: Diagrama de recursos del servicio de colas.*

Para conocer más acerca de este servicio, como su implementación y las representaciones y códigos de respuesta empleados en las diferentes peticiones, se recomienda la lectura de su correspondiente informe de diseño e implementación [Ser16a].

3.2.2. Servicio de estadística

Este servicio permite la aplicación de diferentes métodos estadísticos para la obtención de estadísticas de los datos adjuntos a las peticiones enviadas. Para realizar toda esta serie de cálculos de la forma más rápida y eficiente posible, se ha estimado oportuno que este servicio haga uso de algún programa de cálculo matemático, en lugar de las propias herramientas que ofrece Java para este fin. La principal ventaja que

presentan estos programas es una mayor velocidad de cálculo utilizando vectores y matrices en comparación con los procesos iterativos presentes en los lenguajes de programación cotidianos.

De los diferentes programas de cálculo matemático que hay disponibles actualmente, el más conocido es MATLAB[34], que ofrece un entorno de desarrollo para programas matemáticos escritos en lenguaje MATLAB. El principal problema que presenta es que necesita de una licencia comercial para su uso y disfrute, lo que dificultaría la distribución de este servicio dentro del DNSE3. Es por ello que se ha optado finalmente por utilizar GNU Octave[35], programa de cálculo matemático *open-source* de licencia gratuita que, a pesar de tener un rendimiento y características inferiores a las ofrecidas por MATLAB, presenta una gran compatibilidad con él, hasta el punto de utilizar el mismo lenguaje en sus *scripts*. Aun así, el uso de estas herramientas no se ha fijado por completo y se sigue estando abierto a nuevas propuestas de integración.

Estas capacidades de cálculo se ofrecen a los diferentes servicios del DNSE3, aunque harán un posible uso de este los siguientes servicios, que actuarán como clientes:

- El servicio de orquestación (SPA-01) consultará el conjunto de métodos soportados por el servicio, además de sus parámetros de entrada. Estos métodos se mostrarán a los usuarios de la aplicación para que elijan las diferentes estadísticas que quieren obtener de los resultados obtenidos de las simulaciones.
- El servicio de informe (SPA-02) solicitará la realización de dichos métodos sobre los resultados de las simulaciones para la correcta generación del informe solicitado por los usuarios de la aplicación.
- El servicio de colas (SGA-01) podría llegar a hacer uso de este servicio para la generación de métricas más avanzadas que se publicarían en el servicio de monitorización.

Este servicio ofrece a través de su API REST las siguientes funcionalidades:

- Listar los métodos estadísticos soportados por el servicio.
- Ver cada uno de los métodos soportados el servicio, además de los datos necesarios para su ejecución.
- Aplicar un método estadístico a los datos enviados en la petición.

Todas estas funciones se muestran en el diagrama de casos de uso mostrado en la figura Figura 3.4.

[34] `http://www.mathworks.com/products/matlab/`

[35] `https://www.gnu.org/software/octave/`

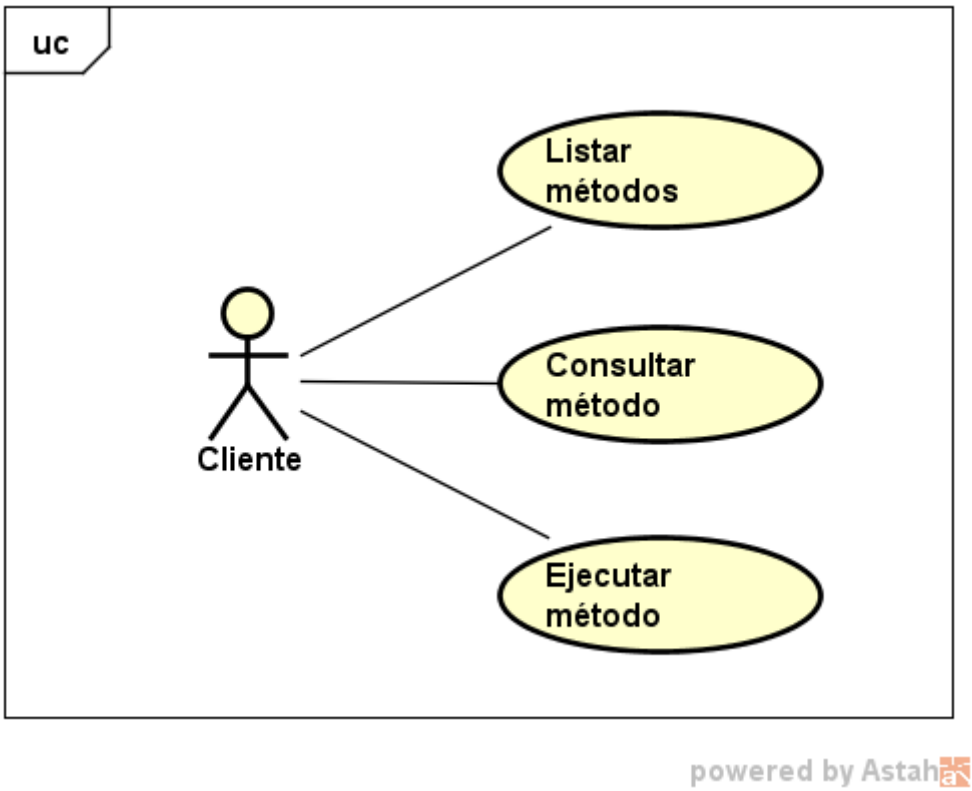

Figura 3.4*: Diagrama de casos de uso del servicio de estadística*.

Para permitir el acceso a dichas funciones, el servicio presenta una arquitectura ROA compuesta por los siguientes recursos:

- *Statistics*: Este recurso contiene el conjunto de todos los métodos estadísticos soportados por el servicio. De todos los métodos HTTP posibles, solo hace uso de GET para mostrar el listado con dichos métodos matemáticos.
- *Statistic*: Este recurso representa cada uno de los métodos integrados en el servicio. Entre sus atributos se encuentran los diferentes parámetros y datos necesarios para su correcta ejecución. Como el objetivo principal de este servicio es la ejecución de dichos métodos, este recurso ofrece los métodos HTTP GET, para consultar los datos necesarios para su ejecución, y POST, que permite el procesado de los datos adjuntos en la petición. Estos datos deben mantener el formato indicado en la representación del recurso tras la llamada al método GET.

En la Figura 3.5 se muestra el diagrama de recursos del servicio, mostrando la dependencia de cada uno de ellos.

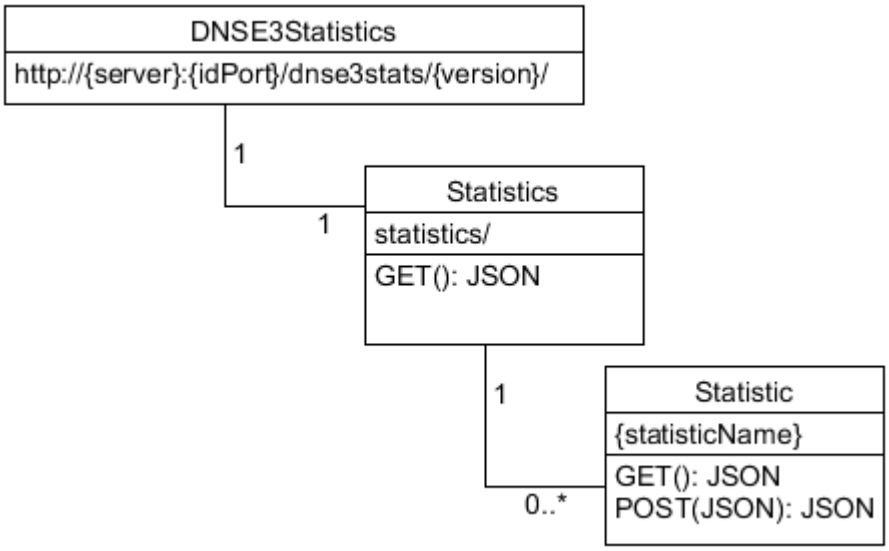

Figura 3.5*: Diagrama de recursos del servicio de estadística*.

Para conocer más acerca de este servicio, como su implementación y las representaciones y códigos de respuesta empleados en las diferentes peticiones, se recomienda la lectura de su correspondiente informe de diseño e implementación [Ser16b].

3.2.3. Servicio y cliente de simulación

Antes de entrar en el diseño de este servicio, es necesario realizar unas aclaraciones en relación a su funcionalidad. El propósito principal de este servicio es la correcta ejecución de las simulaciones solicitadas por los usuarios y el respaldo de los resultados obtenidos. De forma adicional, se ofrecerá una forma de gestionar dicha ejecución de las simulaciones, particularmente la consulta del estado de estas simulaciones y la cancelación de su ejecución, pero no constituyen el núcleo del servicio. Es por esto que en este caso se realizará una división en el estudio del servicio: por un lado, el cliente de simulación, encargado de la ejecución de las simulaciones, y por otro el servicio de simulación, que permite la gestión de la ejecución.

Este cliente se encarga de la correcta ejecución de las simulaciones recogidas del servicio de colas utilizando el simulador ns-3 y del correcto respaldo de los resultados obtenidos. Se ejecutarán en paralelo varios despliegues de este cliente, cada una en una instancia distinta de la nube computacional, por lo que se requiere de cierta coordinación entre ellos. Se ha optado por utilizar el modelo *work-stealing* para la coordinación y asignación de las ejecuciones de simulación. Con este modelo, cada uno de los clientes de simulación activos deberá recoger del servicio de colas la siguiente simulación a procesar, con lo que se evita tener un registro de los servicios activos. Para evitar la apropiación concurrente de la misma simulación, se utiliza el método de reserva de tareas comentado anteriormente en el servicio de colas.

El ciclo de trabajo efectuado por el cliente se organiza en la siguiente secuencia de tareas:

1. El servicio de simulación consulta al servicio de colas si hay alguna simulación pendiente de ejecutar. Si la respuesta es negativa, se espera 30 segundos antes de volver a realizar la consulta. Se ha optado por este tiempo ya que presenta un compromiso entre el tiempo de reacción del cliente ante la aparición de nuevas simulaciones, donde interesa que sea lo más rápido posible, y la sobrecarga producida en la red de la aplicación en el caso de que fuesen demasiado bajos.
2. Si la respuesta es positiva, se comprobará que el modelo empleado de la simulación es accesible. Para ello se realiza una petición HEAD a la URI en la que se encuentra el modelo. Esto se hace para evitar posibles errores de entrada de la tarea recogida.
3. Si no se recibe respuesta, se notifica al servicio de colas del error encontrado en la simulación y volverá al paso 1.
4. Si, por el contrario, recibe una respuesta de la petición, se procede con la reserva de la simulación, enviando en la petición PATCH la URI a la cual se puede acceder al servicio de simulación para consultar la ejecución de la simulación.
5. A partir de este momento, se tendrán dos flujos de trabajos corriendo en paralelo: la ejecución de la simulación y la renovación de esta en el servicio de colas. En caso de terminar uno antes que el otro, le notificará de su finalización y se actuará de la forma acorde.
6. En lo que se refiere al hilo de ejecución de la simulación, se seguirán los siguientes pasos:
   6.1. Se realizará la descarga del *script* de simulación, consistente de uno o varios ficheros escritos en C++. En caso de encontrarse comprimidos, se procederá con su descompresión.
   6.2. Tras comprobar que los modelos presentan el formato de fichero adecuado, se procede con su compilación, utilizando las librerías ofrecidas por el simulador. Dado que no se conocen de antemano los diferentes ficheros que componen el modelo, se hará uso de un fichero *makefile* genérico para su compilación. Este fichero se muestra en el apéndice A. Si se produce algún fallo en la compilación, se indica al hilo de renovación que notifique del error en la simulación y termine con su ejecución.
   6.3. Si la compilación ha resultado exitosa, se ejecutará la simulación, empleando para ello tanto los parámetros de entrada como las variables de entorno indicadas en la simulación procedente del servicio de colas.

        Dado que no conocemos la procedencia ni el contenido del *script* del modelo, se deberá ejecutar en un entorno protegido que impida modificar los ficheros almacenados en la propia instancia en la cual está corriendo el servicio o realizar envíos de mensajes a través de la red del sistema. Con estas medidas de seguridad se minimiza la entrada de posibles amenazas al sistema.

        Si la ejecución de la simulación resulta fallida, se notificará al hilo de renovación de forma análoga a como se realizó en el paso 6.2.
   6.4. Una vez completada la ejecución exitosa de la simulación, se procederá a guardar los resultados obtenidos en el servicio de almacenamiento. Si la simulación ejecutada procedía de una única simulación, se guardará tanto la salida estándar y de error como los diferentes ficheros generados tras su ejecución. Si, por el contrario, la simulación procede de la repetición de una misma simulación o de la división de una de barrido de parámetros, solo se guardarán las salidas estándar y

de error, por lo que se reduce la cantidad de espacio empleada por el conjunto de simulaciones.

Dado que cada cliente de simulación trabaja de forma independiente al resto, para evitar posibles conflictos en el volcado de los resultados de las simulaciones, en el caso de las simulaciones de barrido de parámetros los resultados obtenidos se guardarán en ficheros independientes. El nombre de estos ficheros contendrá los parámetros empleados en su ejecución, según se especifica en la simulación recogida del servicio de colas.

Al igual que sucedía en pasos anteriores, si se produce algún fallo durante este paso, se notificará al hilo de renovación para que notifique del error al servicio de colas.

6.5. Tras guardar correctamente los resultados, se notificará al hilo de renovación del éxito en la ejecución, de forma que el hilo de renovación notifique al servicio de colas del éxito de la simulación, finalizando con su ejecución.

7. En lo referente al hilo de renovación, se procederá con el siguiente flujo de trabajo:
   7.1. Una vez iniciado el hilo, se enviará al servicio de colas una petición PATCH a la tarea que se está ejecutando en la que se pide que cambie su estado para indicar su procesado, además de la renovación de la fecha de expiración de la reserva de la tarea y la inclusión de la URI del servicio de simulación para la consulta del estado de la ejecución.
   7.2. Tras enviar la solicitud, se esperará un tiempo igual al tiempo de vida indicado en la simulación recibida del servicio de colas.
   7.3. Una vez se termina de esperar, se realiza una nueva petición PATCH a la tarea, en la que se pide solamente que se renueve la fecha de expiración.
   7.4. Se realizarán los pasos 7.2 y 7.3 de forma cíclica hasta que termine la ejecución de la simulación. En caso de producirse algún error en los pasos 7.1 al 7.3, se notificará al hilo de procesado para que termine de efectuar el trabajo que esté realizando y aborte su ejecución.
8. Tras terminar ambas tareas, se procede con la eliminación de todos los ficheros empleados en la ejecución. Esto incluye los ficheros descargados y, en su caso, descomprimidos del modelo, el fichero ejecutable que contiene la simulación y los resultados obtenidos tras su ejecución.
9. Una vez se tiene el entorno de trabajo del servicio restaurado, se volverá a retomar el paso 1.

Este flujo de trabajo muestra el proceso a seguir si solo se va a trabajar con una única simulación por servicio. En caso de querer trabajar con más de una simulación a la vez, cada vez que se produzca la reserva de una simulación se iniciará un nuevo flujo de trabajo encargado de pedir una nueva simulación. Esta generación de nuevos flujos se producirá hasta alcanzar el tope máximo de trabajos que puede realizar el servicio de forma paralela. Cada vez que termine un trabajo de ejecución, se retomará la petición de una nueva simulación únicamente cuando no haya otro flujo de trabajo pidiéndola actualmente.

Por lo que se ha estado mostrando anteriormente, este cliente se comunicará con los siguientes servicios:

- El servicio de colas (SGA-01). Se le consultarán las siguientes tareas pendientes de procesar para ejecutarlas, al igual que se reservarán para evitar que sean utilizadas por otros clientes de simulación. Este servicio, a su vez, notificará al servicio de simulación de la cancelación de la ejecución si se encuentra actualmente activa, al ser el único servicio que conoce la URI para acceder al trabajo.
- El servicio de almacenamiento (SPI-01). Se accederá a este servicio para consultar la disponibilidad de los modelos de simulación a utilizar y proceder con su posterior recogida. Por otra parte, se le enviarán los resultados obtenidos tras la ejecución de las simulaciones, encontrándose cada uno de ellos en un fichero independiente y en cuyo nombre aparecerán los valores de los parámetros empleados en la simulación.

En lo que respecta al servicio de simulación, deberá realizar las funciones de gestión de las simulaciones procesadas por el cliente de simulación, entre las cuales se incluyen:

- Listar los trabajos de simulación actualmente activos en el servicio.
- Consultar el estado y la evolución de los trabajos de simulación.
- Cancelar la ejecución de una simulación.

Estas últimas funciones se muestran en el diagrama de casos de uso mostrado en la figura Figura 3.6.

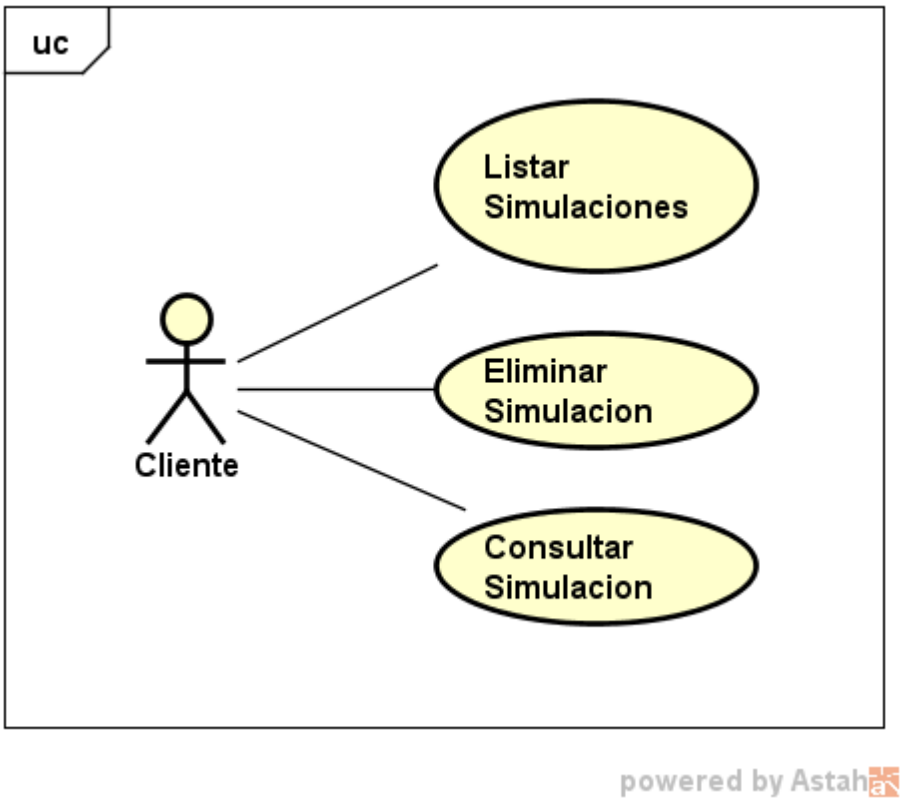

Figura 3.6*: Diagrama de casos de uso del servicio de simulación.*

Para permitir el acceso a dichas funciones, el servicio presenta una arquitectura ROA compuesta por los siguientes recursos:

- *Simulations*: Este recurso representa el conjunto de procesos de simulación actualmente activos dentro del propio servicio. Su único atributo es el listado de estos procesos. De los diferentes métodos HTTP únicamente soporta el método GET, con el que se puede consultar dicho listado.
- *Simulation*: Este recurso representa cada uno de los procesos de simulación actualmente activos. Los atributos de este recurso son los datos obtenidos desde el servicio de colas para la correcta ejecución de la simulación, además del estado en el que se encuentra la simulación. Permite el uso de los métodos HTTP GET, para consultar el estado de ejecución de la simulación, y DELETE, para cancelar dicha ejecución.

En la Figura 3.7se muestra el diagrama de recursos del servicio, mostrando la dependencia de cada uno de ellos.

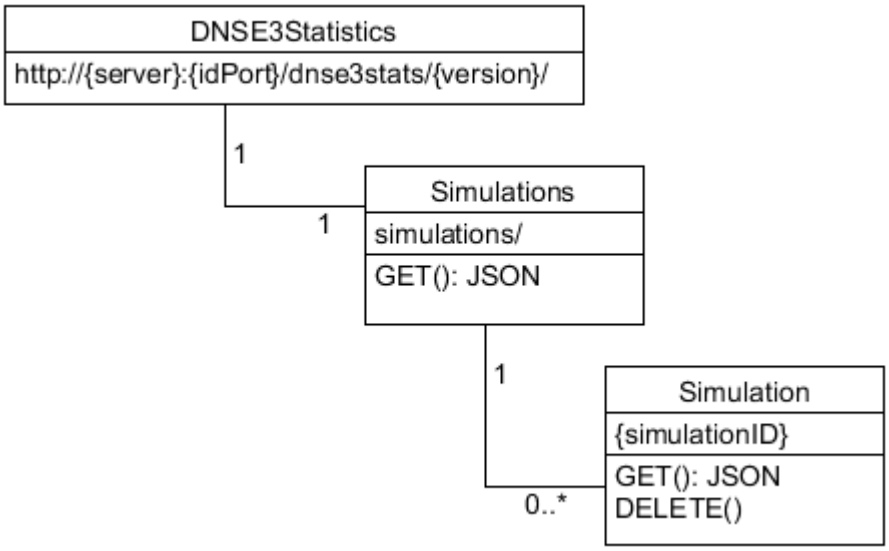

Figura 3.7*: Diagrama de recursos del servicio de simulación.*

Para conocer más acerca de este servicio, como su implementación y las representaciones y códigos de respuesta empleados en las diferentes peticiones, se recomienda la lectura de su correspondiente informe de diseño e implementación [Ser16e].

3.2.4. Servicio de gestión de sesiones

Este servicio se encarga de la gestión tanto de las cuentas de usuario de los usuarios finales de la aplicación como de los propios servicios que componen el sistema. En lo que se refiere a las cuentas de usuario, este servicio mantendrá los datos de cada una de estas cuentas, al igual que permitirá el acceso a la aplicación a los usuarios. Por otro lado, en lo que se refiere a los servicios del sistema, permitirá comprobar la identidad de las peticiones realizadas a cada uno de los servicios. Se añade este nivel adicional de

seguridad de las comunicaciones establecidas para evitar que se envíen peticiones falsas que puedan comprometer el correcto funcionamiento del sistema.

Entre las distintas funciones que debe desempeñar este servicio se encuentran:

- Listar el conjunto de usuarios registrados en el sistema y crear un nuevo usuario,
- Consultar los datos de cada uno de los usuarios registrados y darles de baja.
- Permitir el acceso a los usuarios a la aplicación.
- Verificar la identidad del usuario que realizó una petición al sistema.
- Listar el conjunto de servicios activos en el sistema.
- Consultar los permisos que tiene un servicio en el sistema.
- Verificar la identidad del servicio que realizó una petición en el sistema.

Para integrar estas funcionalidades, se necesita de un protocolo de autenticación con un nivel de protección suficiente para su uso en la aplicación. El problema presente en el uso de este servicio es que, debido a la carencia de estado de los servicios REST, será necesario enviar la información de autenticación de los usuarios en cada una de las peticiones, por lo que se deberá de intentar, en la medida de lo posible, que no contenga información crítica de los usuarios.

De entre los diferentes protocolos de autenticación, hay alguno que permite solucionar este problema, como sucede con el protocolo OAuth 2.0, que se basa en el uso de unos códigos denominados *tokens* que permiten utilizar las aplicaciones en representación del usuario vinculado durante un periodo de tiempo limitado [Har12]. Estos *tokens* se adjuntarían en las peticiones realizadas en lugar de los datos personales del usuario. Si el *token* caduca mientras se utiliza la aplicación, se deberá de generar un nuevo *token* para continuar con su uso habitual.

Si queremos utilizar este protocolo dentro de la aplicación para realizar la gestión de sesiones, podríamos realizar una implementación propia del protocolo o, en su lugar, buscar una implementación ya implementada y testeada por un proveedor de *tokens* OAuth 2.0. Aunque la primera alternativa permitiría una mayor personalización presenta el problema de que debe implementar correctamente el protocolo de forma íntegra. Por su parte, el uso de un proveedor ya implementado nos ofrecería una solución robusta y testeada del protocolo, además de que se actualice el software a medida que se producen cambios en el protocolo.

Una alternativa adicional para la gestión de las cuentas de usuario sería utilizar el proveedor de *tokens* de servicios públicos de terceros, como Facebook, Google o Twitter, para permitir la vinculación de las cuentas de usuario en dichos servicios a los usuarios propios de nuestra aplicación. De estos servicios públicos, solo se pediría acceso a los datos personales básicos y relevantes para la aplicación, como puede ser el nombre del usuario, y en ningún momento se le pedirá permiso para acceder a alguna funcionalidad adicional.

3.2.5. Servicio de orquestación

El servicio de orquestación se encargará de orquestar los diferentes servicios de la nube, es decir, coordinar su funcionamiento para realizar las peticiones recibidas por parte

de los usuarios. Estas peticiones pueden ser tanto de ejecución de las simulaciones como de recuperación de datos y generación de informes. Hay que tener en cuenta que el principal cliente esperado de este servicio será un módulo software que implemente la interfaz de usuario final.

Para coordinar todas estas tareas, el servicio de orquestación se comunicará con los siguientes servicios:

- El servicio de colas (SGA-01) recibirá cada una de las simulaciones a procesar, estando estas ya preparadas para contener cada una de ellas las diferentes combinaciones de parámetros enviados por el usuario. Este servicio, a su vez, notificará al servicio de orquestación de la evolución de las simulaciones, utilizando para ello un modelo de notificaciones *push*.
- El servicio de estadística (SGA-03) enviará el conjunto de métodos estadísticos a disposición de los usuarios para la realización de los informes, así como los parámetros necesarios para aplicar dichos métodos.
- El servicio de informe (SPA-02) recibirá las peticiones de los usuarios de la generación de informes, con los diferentes métodos estadísticos a realizar, así como de las peticiones de formateo de los resultados.
- El servicio de almacenamiento (SPI-01) recibirá los modelos de simulación proporcionados por los usuarios para su posterior uso en el procesado de las simulaciones. Adicionalmente, se le consultará los diferentes ficheros aptos para la descarga por parte del usuario que contendrán los resultados obtenidos de las simulaciones.

Antes de continuar con la explicación de este servicio, se definirán unos términos que se utilizarán a lo largo de esta sección. El primero de ellos es el término “paquete de simulación”, el cual hace referencia a los modelos de simulación utilizados por los usuarios. Estos paquetes de simulación consistirán en los modelos de simulación (es decir, la topología de la red, las características de los enlances y nodos…), pudiendo ser tanto *scripts* individuales como una agrupación de ellos presentados en un archivo comprimido. Sobre este paquete se realizarán las diferentes peticiones de ejecución de simulación. El otro término, “descripción de simulación”, hace referencia al conjunto de parámetros que definen las diferentes simulaciones a ejecutar. Dentro de dichos parámetros se incluye, además de los propios de la simulación, el tipo de simulación que se desea efectuar, que puede ser individual o de barrido de parámetros, y el número de veces que se desea ejecutar cada simulación. Estas descripciones se vincularán a cada uno de los paquetes de simulación, de tal forma que caracterizan la simulación a ejecutar de forma conjunta. Sobre un mismo paquete de simulación (topología, nodos, enlaces, etc) se pueden proponer muchas descripciones de simulación (valores de los parámetros, etc).

Una vez aclarados dichos términos, se procede con la presentación de las funciones desempeñadas por el servicio de orquestación, entre las que se encuentran:

- Consultar los paquetes de simulación solicitados por cada uno de los usuarios y añadir nuevos paquetes de simulación a la aplicación.
- Recuperar y eliminar los paquetes de simulación de cada uno de los usuarios.
- Listar las diferentes descripciones de simulación, tanto individuales como de barrido de parámetros, vinculadas a cada uno de los paquetes de simulación.

- Añadir nuevas descripciones de simulación a cada uno de los paquetes de simulación.
- Listar los parámetros de las diferentes descripciones de simulación
- Modificar y eliminar los parámetros de las descripciones de simulación
- Procesar cada una de las descripciones de simulación de los diferentes paquetes de simulación.
- Consultar la evolución del procesado de las descripciones de simulación en ejecución.
- Eliminar las descripciones de simulación solicitadas de cada uno de los paquetes de simulación.
- Consultar los resultados obtenidos tras la ejecución de las descripciones de simulación.
- Recuperar los resultados obtenidos tras la ejecución de las descripciones de simulación.
- Consultar el listado de informes asociados a las diferentes descripciones de simulacióny generar nuevos informes de estas.
- Recuperar y eliminar los informes asociados a las diferentes descripciones de simulación.

Todas estas funciones se muestran en los siguientes diagramas de casos de uso, mostrado en las figuras Figura 3.8, Figura 3.9, Figura 3.10 y Figura 3.11. Se ha dividido la organización de estos casos de uso en función del elemento central al que hacían referencia, como son los paquetes de simulación, las descripciones de simulación, los resultados de las simulaciones y los informes de estas.

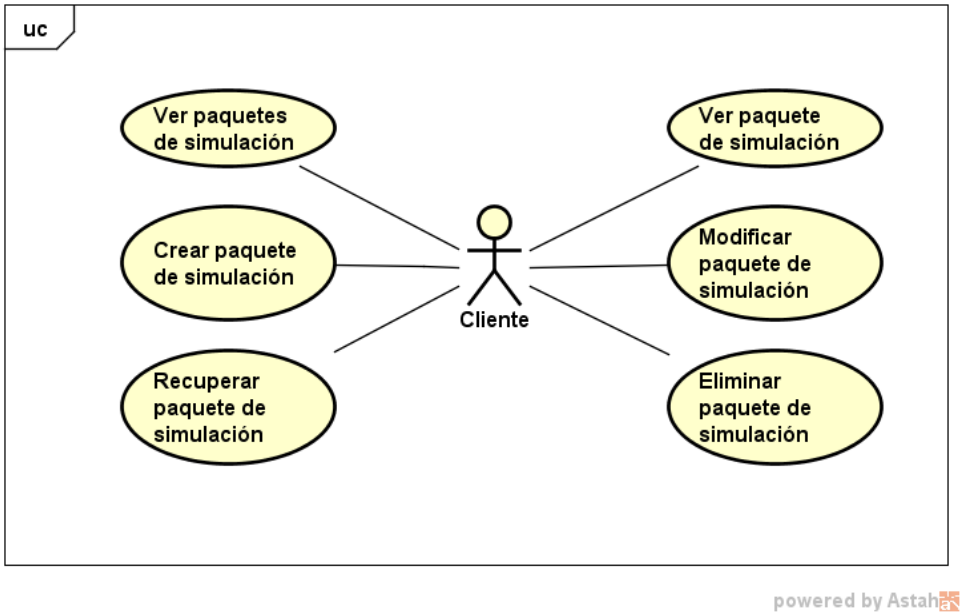

Figura 3.8: *Diagrama de casos de uso del servicio de orquestación vinculados a los paquetes de simulación.*

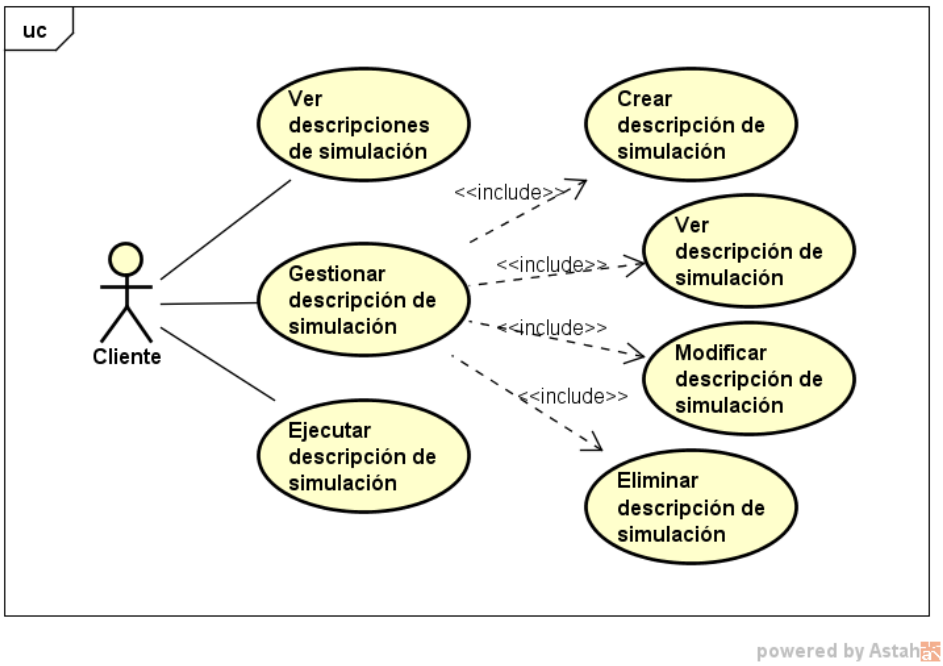

Figura 3.9: *Diagrama de casos de uso del servicio de orquestación vinculados a las descripciones de simulación.*

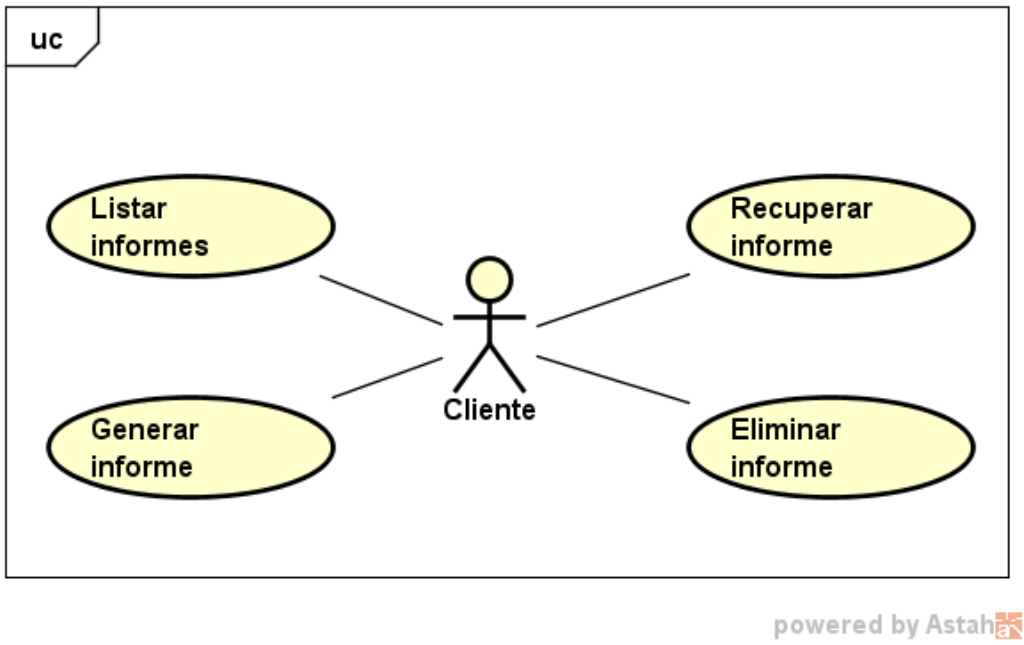

Figura 3.10: *Diagrama de casos de uso del servicio de orquestación vinculados a los informes de las simulaciones.*

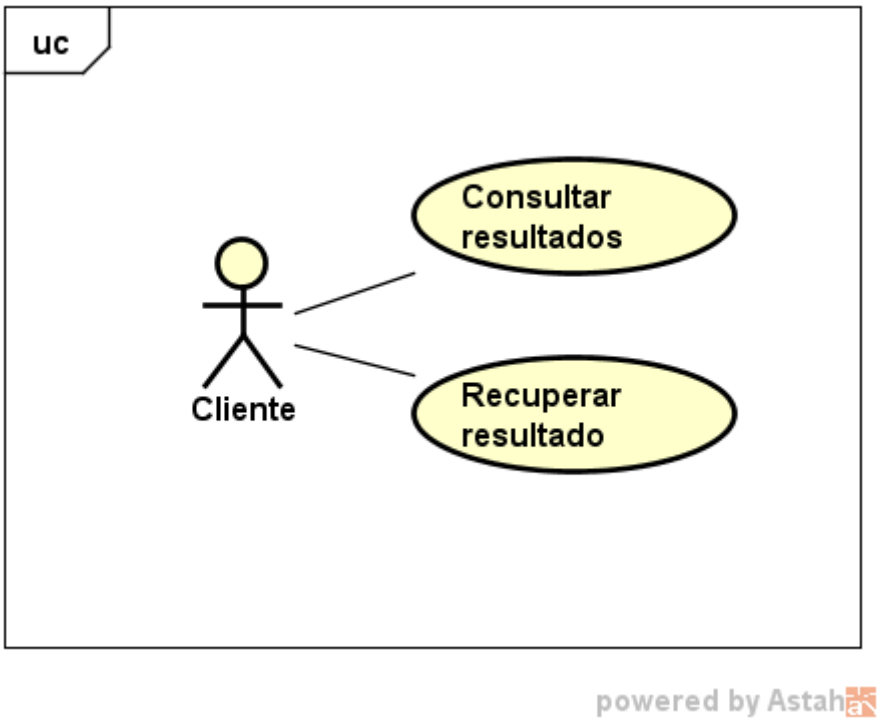

Figura 3.11: *Diagrama de casos de uso del servicio de orquestación vinculados a los resultados de las simulaciones.*

Para ejecutar las descripciones de simulación, el servicio de orquestación deberá realizar la correcta división de las simulaciones de barrido de parámetros, generando todas las posibles combinaciones de parámetros que se hayan definido. Tanto si se tratan de simulaciones individuales como si son de barrido de parámetros, deberá ofrecer un nuevo recurso que permita recoger las notificaciones *push* de la evolución de las simulaciones enviadas.

Para permitir el uso de este servicio por parte de los usuarios, se presenta una arquitectura ROA compuesta por los siguientes recursos:

- *SimulationPackages*: este recurso hace referencia al listado de los paquetes de simulación de cada uno de los usuarios del sistema. Su único atributo es el listado de los paquetes. De todos los métodos HTTP disponibles, este recurso soporta los métodos GET, para recuperar el listado de los paquetes, y POST, para agregar nuevos paquetes al sistema.
- *SimulationPackage*: este recurso hace referencia a cada uno de los paquetes de simulación disponibles en el sistema. Los atributos de este servicio incluyen el nombre del paquete, su descripción, el listado de descripciones de simulación vinculadas a dicho paquete y el modelo que contiene el paquete. Este recurso, a su vez, expone los métodos HTTP GET, para recuperar el estado del recurso, así como el acceso a sus recursos subyacentes, y DELETE, para eliminarlo del sistema.
- *SimulationDescriptions*: este recurso hace referencia al listado de todas las descripciones de simulación vinculadas a cada paquete de simulación. Su único atributo es el listado de las descripciones. De los métodos HTTP disponibles, este recurso soporta los métodos GET, para recuperar el listado de las descripciones

de las simulaciones, y POST, para agregar nuevas descripciones de simulación a cada uno de los paquetes.

- *SimulationDescription*: este recurso hace referencia a la descripción de una de las simulaciones vinculadas a alguno de los paquetes. Los atributos de este recurso son el nombre de la descripción, los parámetros que caracterizan la simulación a procesar, el tipo de simulación deseada (individual o de barrido de parámetros), su estado de ejecución y el número de veces que se debe procesar cada simulación. Los métodos HTTP soportados por el este recurso son GET, para ver el estado del recurso y poder acceder a los recursos subyacentes, PUT, para modificar alguno de los atributos del recurso, POST, para solicitar la ejecución de la simulación.
- *Parameters*: este recurso hace referencia al conjunto de los diferentes parámetros que se utilizan en las descripciones de simulación. Su único atributo es el listado de estos. Solo permite utilizar el método HTTP GET para la recuperación de dicho listado.
- *Parameter*: este recurso hace referencia a cada uno de los parámetros empleados en las descripciones de simulación. Los atributos de este recurso incluyen tanto su nombre como el tipo de parámetro que es. El resto de atributos dependen del tipo de parámetro que es, pudiendo ser uno de los siguientes 5 tipos:
    - Variables de entorno: permiten ajustar las variables de entorno empleadas durante la ejecución de la simulación. Su único atributo adicional es el valor de la variable.
    - Cadena de caracteres: permiten el uso de parámetros alfanuméricos en la ejecución de las simulaciones. El atributo adicional que presenta este tipo de parámetro depende del tipo de simulación a la que está vinculado. Si es una simulación individual, solo tendrá el valor que toma el parámetro. Si por el contrario es una simulación de barrido de parámetros, este atributo tiene todos los posibles valores que pueda tomar.
    - Numérico: permite la introducción de números, ya sean enteros o decimales, como entrada de las simulaciones. Al igual que sucedía con los parámetros de cadenas de caracteres, si la simulación es individual solo puede tener el valor que tomará. Si, por el contrario, la simulación es de barrido de parámetros, los atributos que puede tener este recurso son o bien sus posibles valores, o bien el rango de valores que puede tomar, indicando el valor mínimo, máximo y distancia entre los diferentes valores tomados.
    - Numérico aleatorio: permite la introducción de números generados aleatoriamente como entrada de las simulaciones. Este tipo de parámetro no tiene actualmente ningún atributo adicional.

  Este recurso presenta de entre todos los métodos HTTP disponibles, los métodos GET, para consultar el estado del propio parámetro, PUT, para tanto crear un nuevo parámetro como modificarlo, y DELETE, para eliminarlo de la descripción de la simulación.
- *Reports*: Este recurso representa el listado de los diferentes informes generados a partir de los resultados obtenidos tras la ejecución de las simulaciones. Su único atributo es el listado de los informes. Permite el uso de los métodos HTTP GET, para consultar dicho listado, y POST, para solicitar la generación de un nuevo informe a partir de los resultados.

- *Report*: Este recurso representa cada uno de los informes generados a partir de las simulaciones. Su único atributo es el documento, de formato variable, generado con dichos datos. Permite el uso de los métodos GET, para recuperar el informe, y DELETE, para eliminarlo del sistema.
- *Outputs*: Este recurso representa el listado de los diferentes resultados obtenidos tras la ejecución de las simulaciones. Su único atributo es el listado de los resultados. Permite el uso de los métodos HTTP GET, para consultar dicho listado, y POST, para solicitar la generación de un nuevo informe a partir de los resultados.
- *Output*: Este recurso representa cada uno de los resultados obtenidos a partir de las simulaciones. Su único atributo es el documento que contiene dichos datos. Permite el uso de los métodos GET, para recuperar el documento con los resultados, y DELETE, para eliminarlo del sistema.

En la Figura 3.12 se muestra el diagrama de recursos del servicio, mostrando la dependencia de cada uno de ellos.

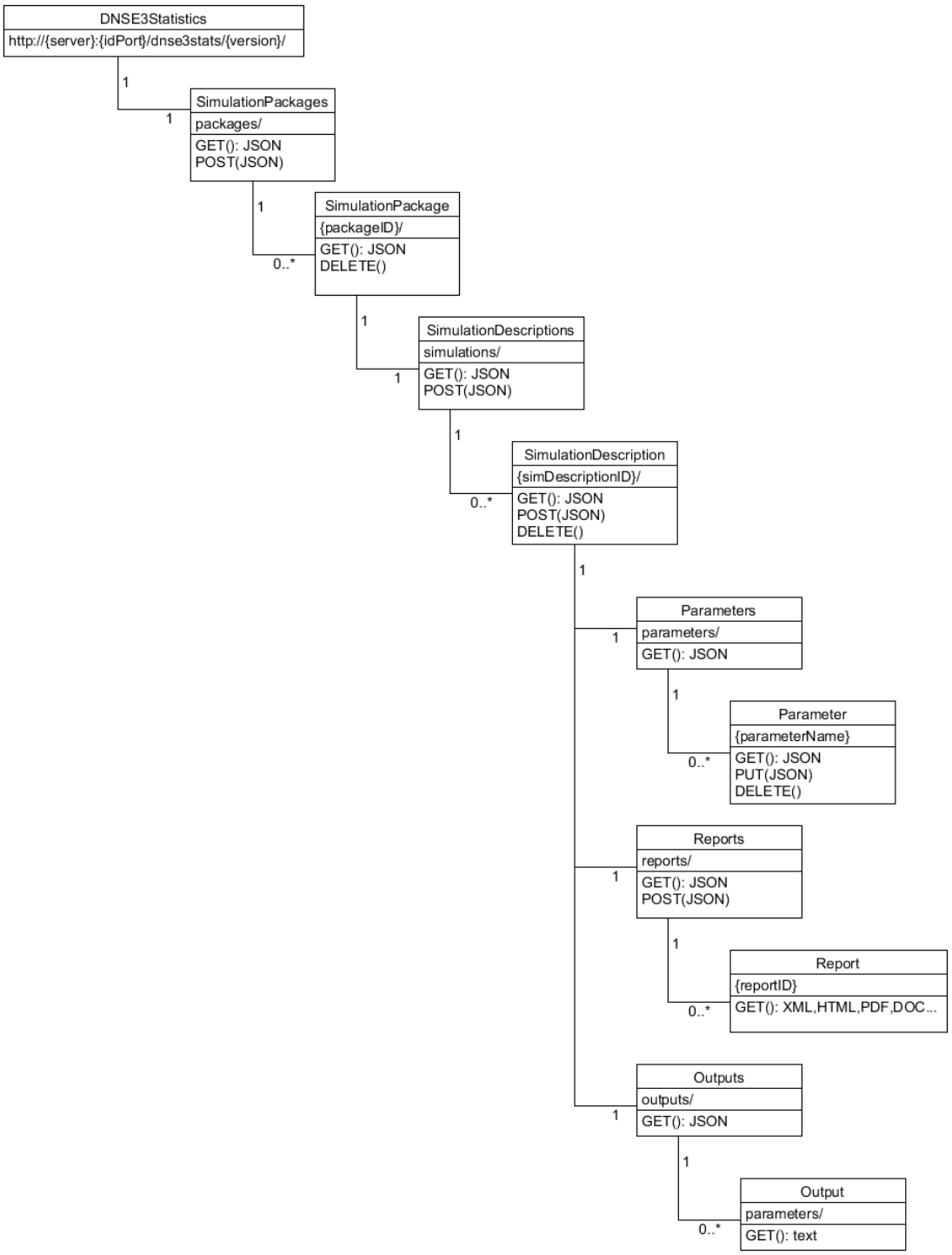

Figura 3.12*: Diagrama de recursos del servicio de orquestación.*

Para conocer más acerca de este servicio, como su implementación y las representaciones y códigos de respuesta empleados en las diferentes peticiones, se recomienda la lectura de su correspondiente informe de diseño e implementación [Ser16d]

3.2.6. Servicio de informe

Este servicio se encarga de procesar los datos obtenidos tras la ejecución de las simulaciones. Debido a que, según se comentó en la sección 3.2.3, los resultados de las simulaciones se encuentran fraccionados en multitud de ficheros, este servicio debe agrupar dichos resultados en un fichero que sigue el formato especificado por el usuario. A su vez, deberá generar los informes solicitados por los usuarios, en los que se pueden incluir diferentes estadísticas de los resultados obtenidos.

Para poder cumplir con su propósito, el servicio de informes se comunicará con los siguientes servicios:

- El servicio de estadística (SGA-03) aplicará los métodos estadísticos a los datos enviados por el servicio de informe.
- El servicio de orquestación (SPA-01) enviará las solicitudes de generación de informe. En estas solicitudes se puede indicar el fichero a partir del cual obtener los datos o la forma en la que se agruparán para su posterior uso.
- El servicio de almacenamiento (SPI-01) permitirá el acceso a los ficheros que contienen los resultados de las simulaciones.

Dado que, para cada informe, puede ser necesario realizar un gran número de peticiones HTTP para la recuperación de los datos, y que no son instantáneas, se ha propuesto ofrecer un acceso al servicio similar al presente en el servicio de colas, permitiendo la publicación de estas peticiones. Sin embargo, este servicio incluirá a su vez un *worker* encargado de recorres la cola del servicio e ir preparando y generando los diferentes informes pedidos. Aunque el trabajo de este *worker* se podría haber integrado en un servicio aparte que permitiese su escalado, de forma similar a lo sucedido con el servicio de simulación, no se ha estimado oportuno ya que no habrá un volumen de peticiones de informe comparable al de las simulaciones, las cuales son más numerosas. Aclarando un poco las ideas expuestas, este servicio ofrece el acceso a una cola FIFO en la cual se pueden publicar peticiones de generación de informes, que serán procesadas por un *worker* integrado en el servicio,

Las funciones expuestas por el servicio de informes son las siguientes:

- Listar las peticiones de generación de informe de los resultados de las simulaciones realizadas.
- Añadir nuevas peticiones de generación de informe.
- Consultar cada una de las peticiones de generación de informe.

Estas funciones se muestran en el diagrama de casos de uso en la Figura 3.13. Dentro de la generación de informe se incluye también el agrupamiento y formateo de los resultados de las simulaciones.

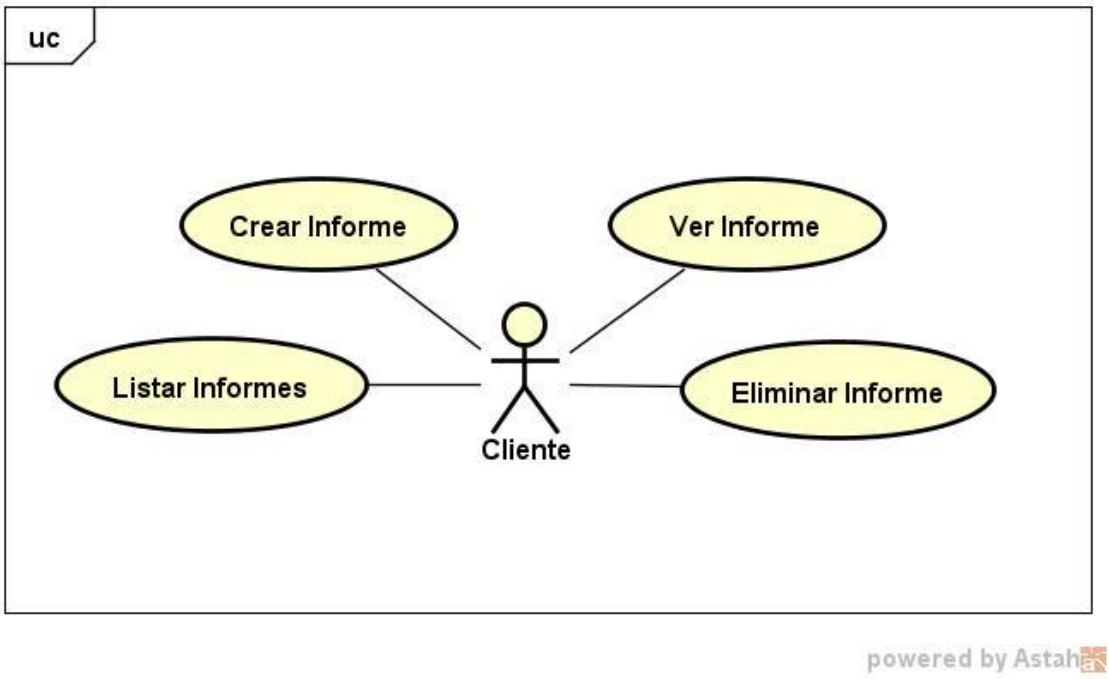

Figura 3.13*: Diagrama de casos de uso del servicio de informe.*

Para permitir el acceso a dichas funciones, el servicio presenta una arquitectura ROA compuesta por los siguientes recursos:

- *Tasks*: este recurso representa el listado de peticiones de generación de informe registradas en el servicio. Este recurso, a su vez, proviene de la implementación del servicio de colas, por lo que mantiene tanto su nombre como atributos y funciones. Cabe recordar que las funciones soportadas eran los métodos GET, para la recuperación del listado, y POST, para añadir las nuevas peticiones al listado.
- *Task*: este recurso representa cada una de las peticiones de generación de informe registradas en el servicio. Este recurso, a su vez, proviene de la implementación del servicio de colas, pero solo mantendrá los atributos genéricos de las tareas. El resto de atributos que incorpora hacen referencia al resto de parámetros necesarios para la correcta generación del informe. Los métodos HTTP proporcionados por este recurso son GET, para la consulta de los parámetros del informe, y DELETE, para la cancelación del informe, siempre y cuando no se esté preparando en dicho momento.

En la Figura 3.14 se muestra el diagrama de recursos del servicio, mostrando la dependencia de cada uno de ellos.

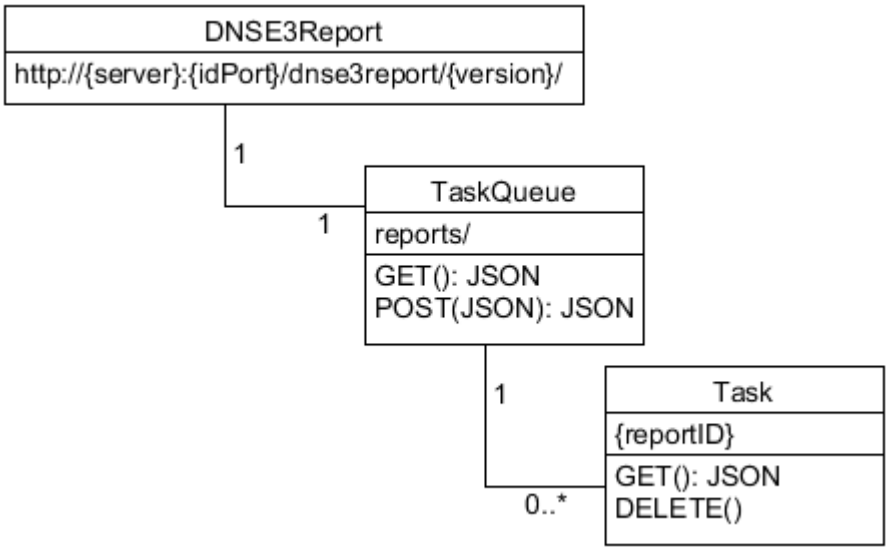

Figura 3.14*: Diagrama de recursos del servicio de informe.*

Para conocer más acerca de este servicio, como su implementación y las representaciones y códigos de respuesta empleados en las diferentes peticiones, se recomienda la lectura de su correspondiente informe de diseño e implementación [Ser16c].

3.2.7. Interacción de los servicios

El propósito de una arquitectura orientada a servicios es la repartición de las diferentes tareas que debe desempeñar la aplicación. Ahora bien, para desempeñar las diferentes funciones de esta, se necesita que dichos servicios se comuniquen entre sí para efectuar dichas funciones. En esta sección se pretende presentar la comunicación existente entre los servicios de la aplicación para realizar los casos de uso más importantes de la aplicación, los cuales son la creación de los paquetes de simulación y de las descripciones de simulación, la ejecución de las simulaciones de los usuarios, tanto individuales como de barrido de parámetros, y la creación de informes de los resultados de las simulaciones.

Para cada uno de estos casos de uso se describirá los intercambios de mensajes realizados y se adjuntará un diagrama de colaboración de los muestre de forma gráfica.

3.2.7.1. Creación de los paquetes de simulación

Cuando un usuario desea emplear un modelo de simulación dentro del DNSE3, deberá de solicitar la creación de un nuevo paquete de simulación vinculado a su cuenta al servicio de orquestación. Para ello se le enviará un formulario en el que se incluyan el nombre del paquete de simulación, una breve descripción de su uso y, finalmente, el modelo empleado. Este modelo deberá ser o bien un *script* de código fuente o bien un archivo comprimido que los contenga. Una vez comprobado el correcto formato del modelo, se enviará al servicio de almacenamiento para permitir su posterior uso y, finalmente, se enviará al usuario la URI que identifica al nuevo paquete de simulación.

Este intercambio de peticiones se muestra en el diagrama de colaboración de la Figura 3.15.

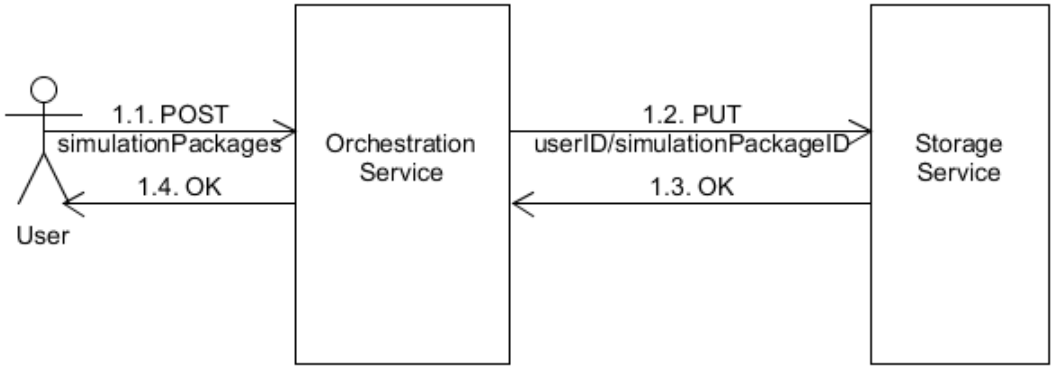

Figura 3.15: *Diagrama de colaboración de la comunicación entre los servicios del DNSE3 para la carga de paquetes de simulación.*

3.2.7.2. Creación de las descripciones de simulación

Cuando los usuarios quieran crear una nueva simulación vinculada a uno de los modelos previamente cargados en el sistema, se realizará una petición de creación al recurso *SimulationDescriptions* vinculado al paquete que contiene el modelo deseado. En la petición de creación se podrán adjuntar los diferentes parámetros que se utilizarán en la simulación. En caso de querer modificarlos posteriormente, se deberán de realizar las peticiones al recurso *Parameter* correspondiente, siempre y cuando aún no se haya ejecutado la simulación.

En el diagrama de colaboración de la Figura 3.16 se muestran estas peticiones.

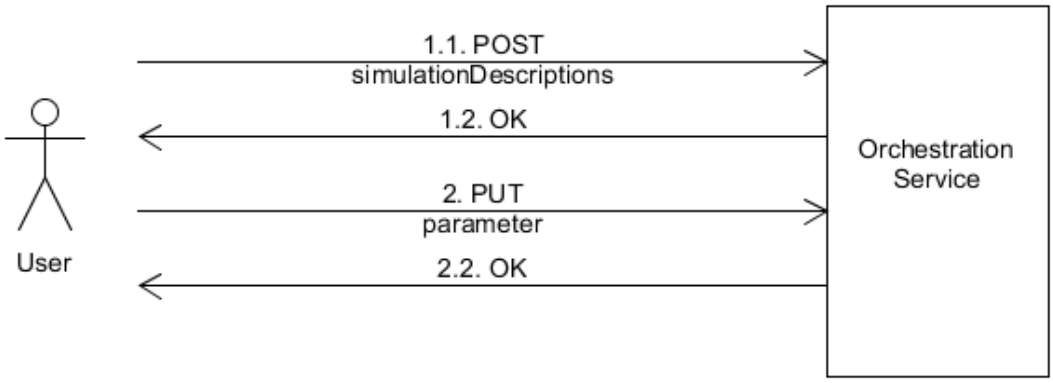

Figura 3.16: *Diagrama de colaboración de las peticiones de creación de descripciones de simulación y modificación de sus parámetros.*

3.2.7.3. Ejecución de las simulaciones

Cuando los usuarios realizan la petición de ejecución de las descripciones de simulación al servicio de orquestación, este, tras realizar la correcta preparación de las diferentes simulaciones necesarias, publicará en el servicio de colas dicho listado de

simulaciones. Una vez haya sido publicado, se le notificará al usuario del éxito de la petición. A partir de este momento, el usuario podrá consultar el progreso de la ejecución.

El servicio de simulación, por su parte, de forma periódica realizará peticiones al servicio de colas para consultar las nuevas simulaciones pendientes de procesar. Cuando el servicio de colas le devuelve una respuesta vacía, empezará con su ejecución realizando el ciclo de reservas y renovaciones de la simulación.

Una vez se complete la ejecución de la simulación, el servicio de simulación avisará al servicio de colas del cambio de estado, quien a su vez notificará al servicio de orquestación, siguiendo el modelo de notificaciones *push*. Cuando el servicio de orquestación las recibe, enviará una petición al servicio de colas para que elimine la simulación de la cola.

Este intercambio de peticiones realizado se muestra en el diagrama de colaboración de la Figura 3.17.

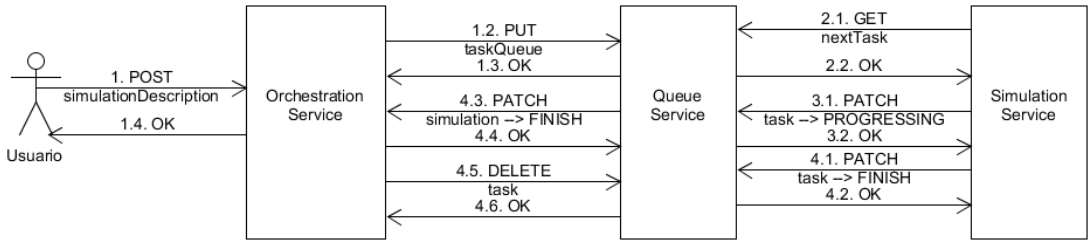

Figura 3.17: *Diagrama de colaboración de la comunicación entre los servicios del DNSE3 para ejecutar las simulaciones.*

3.2.7.4. Generación de informes

Cuando los usuarios desean generar un informe a partir de los resultados de una simulación, deberán realizar dicha petición al servicio de orquestación, indicando los distintos datos que se emplearán y los diferentes cálculos que se desean realizar sobre ellos. El servicio de orquestación retransmitirá dicha petición al servicio de informe, que la almacenará en su cola de informes. Tras completar la publicación, se notificará al usuario del éxito en la operación. A partir de este momento, el usuario podrá consultar cuándo ha finalizado la ejecución del informe.

Por su parte, el servicio de informe irá recogiendo las diferentes tareas presentes en su cola y recogerá los resultados del servicio de almacenamiento. Tras procesarlos, realizará una secuencia de peticiones al servicio de estadística para obtener las estadísticas solicitadas por los usuarios. Tras esta recolección, el servicio de informe terminará de generar el informe y lo publicará en el servicio de almacenamiento en diferentes formatos.

Una vez completada la generación del informe, se notificará al servicio de orquestación del cambio de estado del informe. El servicio de orquestación, por su parte, solicitará al servicio de informes que lo elimine de la cola. A partir de este momento, cuando el usuario consulte el estado del informe, se le notificará de su generación y se le permitirá su descarga, actuando el servicio de orquestación como intermediario entre el usuario y el servicio de almacenamiento.

Este intercambio de peticiones realizado se muestra en el diagrama de colaboración de la Figura 3.18.

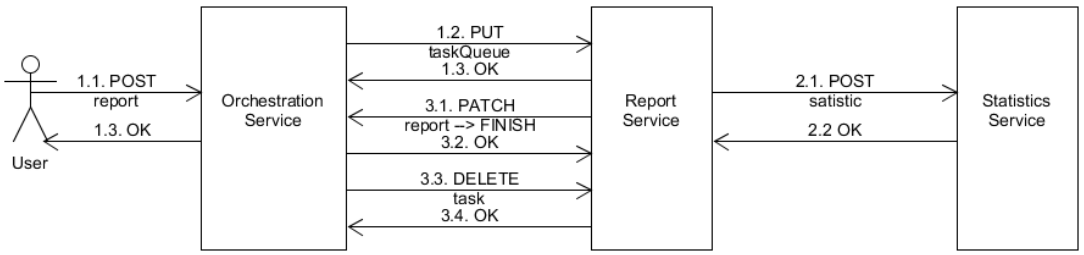

Figura 3.18: *Diagrama de colaboración que muestra la comunicación entre los servicios del DNSE3 para la elaboración de los informes.*

3.3. Servicios de la nube

Según se comentó en la sección 2.5, el diseño original que presentaba el DNSE3 incorporaba una serie de servicios que, a día de hoy, los gestores de nubes computacionales que ofrezcan un nivel de abstracción IaaS traen de forma nativa. En esta sección se pretende presentar las diferentes opciones que podemos aprovechar para incorporar los servicios propios de la infraestructura y la configuración realizada en cada caso.

Dado que se dispone de una gran variedad de *middlewares* de gestión y como nuestro propósito es ofrecer un entorno de simulación distribuido que pueda utilizarse en el mayor número de escenarios posibles, se tomarán los servicios ofrecidos por AWS, al ser la nube de referencia en el nivel de abstracción IaaS. En la Tabla 3.1se listan dichos servicios, con su función principal y el nombre de los servicios análogos en OpenStack y Eucalyptus, al ser los *middlewares* de gestión de nube privada más utilizados.

Servicio	Amazon Web Services	OpenStack	Eucalyptus
Identificación y control de acceso	Identity and Access Management	Keystone	Identity and Acces Management
Gestión de imágenes	Amazon Machine Images	Glance	Eucalyptus Machine Images
Almacenamiento de bloques	Elastic Block Storage	Cinder	Storage Controller
Almacenamiento de objetos	Simple Storage Service	Swift	Walrus Storage
Redes	Virtual Private Cloud	Neutron	Virtual Private Cloud
Monitorización	CloudWatch	Ceilometer	CloudWatch
Orquestación	CloudFormation	Heat	CloudFormation
Computación	Elastic Cloud Computing	Nova	Cloud Controller
Panel de comandos	AWS Console	Horizon	Eucalyptus Management Console

Balanceo de carga	Elastic Load Balancing	Load Balance-as-a-Service (LBaaS)	Elastic Load Balancing

Tabla 3.1: *Comparación de servicios entre los gestores de nubes computacionales basados en AWS.*

A continuación, se mostrarán las diferentes funciones que deben desempeñar estos servicios y se elegirá el o los servicios que mejor las integren y efectúen.

3.3.1. Servicio de almacenamiento

El servicio de almacenamiento debe permitir el almacenamiento persistente, remoto y autoescalable de los diferentes ficheros empleados en la aplicación, al igual que su recuperación. Estos ficheros incluyen tanto los modelos de simulación y los resultados e informes de las simulaciones como los ficheros de configuración de los servicios que permiten la recuperación de su estado tras diferentes sesiones de trabajo.

3.3.1.1. Servicios disponibles

Para solventar esta necesidad se ofrecen dos tipos de servicios de almacenamiento. La primera aproximación consiste en el almacenamiento de objetos, en el que los diferentes ficheros se almacenan como entidades independientes en una colección de objetos denominada contenedor. La otra propuesta es el almacenamiento de bloques, que permite el uso de discos duros virtuales por parte de las instancias empleadas en la nube.

El almacenamiento de objetos permite el almacenamiento de los ficheros como entidades individuales. Estos ficheros ser almacenan en unos contenedores que permiten agrupar los ficheros que tengan algún tipo de relación. De esta forma se puede tener un contenedor por cada tipo de proyecto que se esté realizando en la nube. Este tipo de almacenamiento presenta unos problemas, como son la imposibilidad de utilizar directorios en los contenedores y la no disponibilidad continua de los contenedores. El primero de ellos no es crítico, al poder establecer un pseudo-árbol de directorios por medio del nombre de los ficheros. El segundo, sin embargo, nos impide utilizar el servicio durante ciertos intervalos de tiempo en los cuales el servicio no es accesible. AWS ofrece este servicio bajo el nombre de Simple Storage Service (S3).

El almacenamiento por bloques, de forma similar a otros servicios de la nube como es el servicio de computación, encargado de la gestión de las instancias de la nube; permite el uso de discos duros virtuales para su consumo por parte de la infraestructura. Presentan la ventaja en comparación al almacenamiento de objetos de estar disponible de forma continua el acceso a estos discos, además de que se pueden migrar a otros entornos de nube computacional. Sin embargo, su mayor hándicap es la limitación de su acceso a una única instancia a la vez, requiriendo su montaje y desmontaje. Para utilizar estos recursos se utiliza el servicio Elastic Block Storage (EBS).

Dado que el acceso al servicio de almacenamiento es prácticamente simultáneo por los diferentes servicios del DNSE3, se ha estimado oportuno el uso del almacenamiento de objetos, el cual permite un acceso múltiple a los diferentes ficheros. Es posible que no se puedan acceder al servicio puntualmente, pero sólo supondría que la aplicación dejaría de funcionar durante unos breves instantes.

3.3.1.2. Árbol de ficheros

Para utilizar este servicio, se establecen dos contenedores, el primero de ellos destinado al almacenamiento de los ficheros de configuración de los servicios que presenten un estado permanente entre sesiones de trabajo, como son el servicio de almacenamiento y el servicio de gestión de sesiones. El otro contenedor almacenará los ficheros de los usuarios, siendo estos los modelos de simulación y los informes y resultados de cada una de las simulaciones.

Para ofrecer una jerarquía en los ficheros almacenados, se mantendrá en su nombreun árbol de directorios, separando cada uno de los directorios con la barra invertida "/". La jerarquía empleada en dichos contenedores será la siguiente:

- En el contenedor de configuración de los servicios, se tendrá un directorio por cada uno de los servicios que lo necesiten, teniendo como nombre el del servicio propietario. Dentro de cada uno de estos directorios, la distribución de los ficheros será acorde a las necesidades del servicio.
- En el contenedor de los usuarios se tendrá un directorio para cada uno de los usuarios registrados en el sistema, teniendo como nombre el identificador único de usuario. Dentro de este directorio nos encontraremos:
    - Un directorio por cada uno de los paquetes de simulación de cada usuario. El nombre de este directorio será el identificador del paquete de simulación. Dentro de este directorio estarán:
        - Un directorio llamado *scripts*, el cual contendrá los modelos de simulación utilizados por el paquete de simulación.
        - Un directorio llamado *simulations*, en el cual nos encontraremos:
            - Un directorio con cada una de las descripciones de simulación vinculadas a cada paquete de simulación. Su nombre será el identificador de la descripción. Dentro de este directorio habrá:
                - Un directorio llamado *outputs*, el cual tendrá todos los resultados de la simulación aptos para su descarga directa, al estar ya correctamente preparados.
                - Un directorio llamado *results*, el cual tendrá los resultados de las simulaciones procedentes de la ejecución de simulaciones de barrido de parámetros o sobre las que se ha repetido su ejecución numerosas veces. Estos ficheros no son aptos para su uso por parte de los usuarios de la aplicación, siendo necesaria su agrupación por parte del servicio de informe.
                - Un directorio llamado *reports*, el cual contendrá todos los informes generados por los usuarios.

En la Figura 3.19 se muestra la jerarquía del contenedor de usuarios.

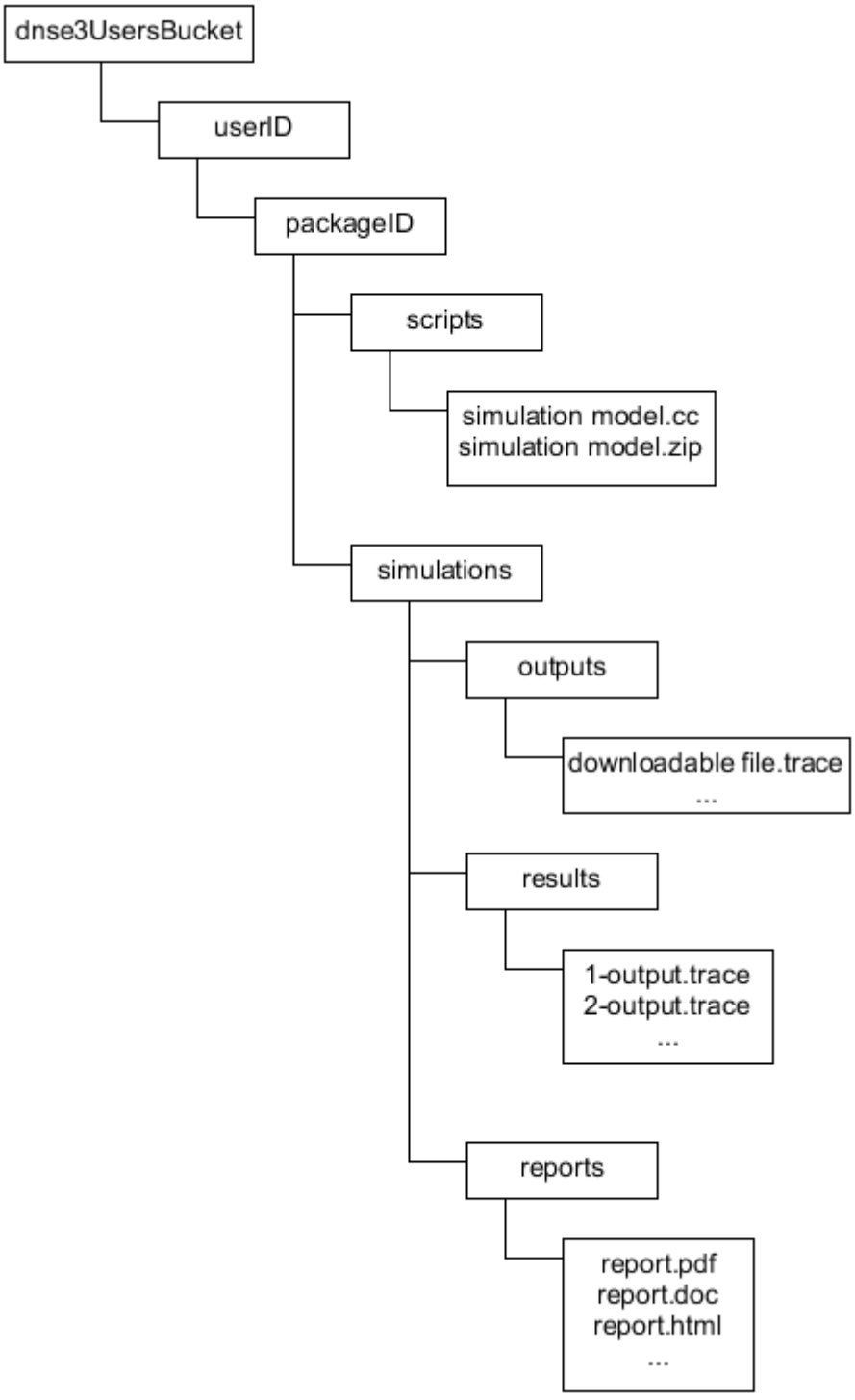

Figura 3.19: *Árbol de pseudo-directorios del contenedor de los ficheros de los usuarios.*

3.3.2. Servicio de monitorización

Este servicio se encargará de gestionar las diferentes métricas relacionadas con el uso que se hace de los diferentes servicios del DNSE3 para gestionar los recursos utilizadosDentro de las posibles métricas a utilizar se pueden encontrar el número de

simulaciones solicitadas y pendientes de procesar, al igual que el número de instancias encargadas de dicho procesamiento.

3.3.2.1. Servicios disponibles

Entre los diversos servicios ofrecidos por los gestores de nube computacional, se suele encontrar un servicio de monitorización, que recoge el uso que se hace de los diferentes recursos de la nube. Estas métricas estaban orientadas a la cuantificación del uso que se hace de la nube para cobrar por él. Sin embargo, actualmente su uso se ha extendido de forma que se permite recoger cualquier tipo de métrica que sea de utilidad y se puedan utilizar con otros fines, como puede ser la automatización del escalado de recursos. El servicio que ofrece AWS para cubrir estas necesidades es el servicio CloudWatch.

El uso que haremos de este servicio será la monitorización de la evolución de la cola de simulaciones. Inicialmente se consideró recoger también el número de instancias de simulación activas. Sin embargo, esta última métrica se recoge de forma automática, al referirse al uso que se hace de los recursos.

3.3.2.2. Métricas empleadas

Dentro del ámbito del DNSE3 se hará uso por el momento de una única métrica, llamada `dnse3.queue.size`, que consistirá en el número de tareas, tanto pendientes de procesar como en ejecución. El propósito de esta métrica será, a través de su histórico, medir la demanda efectuada de la aplicación y comprobar si los recursos empleados en la ejecución de las simulaciones son suficientes o no. Esta métrica la tendrá que publicar el servicio de colas, al ser el único servicio que conoce dicha información. En la sección de escalado se detallará acerca del uso que se hace de esta métrica.

3.3.3. Servicio de escalado

El servicio de escalado es el responsable de realizar un escalado horizontal[36] de los recursos empleados por la aplicación para la ejecución de las peticiones de los usuarios. En particular, realizará el escalado horizontal de las instancias que ejecutan el servicio de simulación, encargadas de procesar las simulaciones de los usuarios.

3.3.3.1. Servicios disponibles

En los gestores de nube computacional se nos ofrece un servicio de orquestación[37], que permite la gestión automática de los recursos empleados. Entre los usos que se pueden hacer de este servicio, se pueden destacar el levantamiento conjunto de recursos vinculados y el escalado automático de estos. El primero de los dos se refiere a la

[36] El escalado horizontal consiste en variar el número de recursos empleados, a diferencia del escalado vertical, en el cual se modifican las características de estos, como puede ser el número de CPUs de la instancia o su memoria RAM. Se emplea el escalado horizontal ya que es más rápido en comparación al escalado vertical, donde generalmente es necesario parar la máquina para realizar estas modificaciones.

[37] Aunque compartan el mismo nombre, no se debe confundir con el servicio de orquestación del DNSE3.

capacidad que nos ofrece el servicio, utilizando un fichero de configuración del servicio o *plantilla*, activar todos los recursos que vayamos a necesitar, siendo estos recursos tanto recursos computacionales como recursos ofrecidos por otros servicios de la nube, como pueden ser contenedores del servicio de almacenamiento de objetos.

El segundo uso que se hace del servicio es el que más nos puede interesar. Por medio de una plantilla podemos definir un grupo de recursos autoescalable. Este grupo de recursos permite definir un conjunto de recursos que presentan la misma configuración y características, pero el número de los que son accesibles puede variar a lo largo del tiempo. Con estos grupos se pueden preparar las instancias en las que funcionarán las aplicaciones que se desean escalar, al compartir todas ellas la misma configuración. Para variar el número de ellos se hace uso de unas políticas de escalado, en las cuales se puede indicar la variación que se debe producir, bien de forma absoluta, bien de forma proporcional al tamaño actual del grupo.

Si el objetivo que queremos alcanzar con este servicio es la automatización del escalado de las instancias de simulación en función de las métricas del tamaño de la cola, se necesita vincular las políticas de escalado a estas métricas. Es en este punto donde entra en juego el sistema de alarmas del servicio de monitorización. Estas alarmas permiten detectar cuando el valor de una métrica o una de sus estadísticas, por lo general bastante sencillas, supera un determinado umbral en un intervalo de tiempo. A estas alarmas se les puede asignar una acción tanto cuando se produce la alarma como cuando no.

En definitiva, los entornos de nube computacional no presentan un servicio que pueda realizar el escalado basándose en las métricas recogidas por él mismo, como se planteaba en este servicio de escalado, sino que se reparten este trabajo en los servicios de orquestación y monitorización, por lo que emplearemos ambos en nuestra aplicación. En el caso de AWS, estos servicios se denominan CloudFormation y CloudWatch respectivamente.

3.3.3.2. Configuración de los servicios

Se utilizarán las plantillas del servicio de orquestación para la preparación del entorno empleado por el DNSE3. En esta plantilla se configurarán los siguientes recursos:

- Una instancia configurada para ejecutar tanto el servicio de orquestación del DNSE3 como el de gestión de usuarios de la aplicación. Esta instancia tendrá acceso a la red externa y permitirá la recogida de las peticiones HTTP de los usuarios.
- Una instancia configurada para ejecutar los servicios de colas, estadística e informe.
- Un grupo de instancias escalable configurado para ejecutar el servicio de simulación. El sistema operativo utilizado por estas instancias presenta una versión del simulador ns-3 previamente instalado y configurado.
- Una política de escalado, tanto ascendente como descendente, que permita la modificación del número de instancias del grupo de simulación, al cual estará vinculado.
- Un sistema de alarmas que tendrá asociadas las políticas de escalado de las instancias de simulación. Se hablará sobre ellas a continuación.

- Un grupo de usuarios, perteneciendo cada uno de ellos a los servicios de orquestación, informe y simulación. Se hablará de estos usuarios en la sección 3.3.4Servicio de identificación.

El sistema de alarmas empleado en el DNSE3 define un conjunto de reglas que permite adaptar el número de instancias empleadas a la demanda de la aplicación. Dado que las simulaciones presentan tiempos de compilación y ejecución distintos y no se conoce el patrón de peticiones de simulaciones, la métrica empleada en estas reglas será el número de tareas presentes en la cola. Esta métrica se utilizará de las siguientes formas:

- Se establecerá un número de simulaciones que deberá de poder procesar una única instancia, para definir unos umbrales que serán utilizados por las reglas de escalado.
- Se consultará la evolución de la cola a lo largo de un periodo de tiempo, empleándose en este caso la variación en lugar del tamaño absoluto. Si esta variación es positiva, indica que se están incluyendo nuevas simulaciones a procesar y podría ser necesario levantar una nueva máquina. Si por el contrario es negativa, solamente se planteará reducir el número de instancias cuando este decremento es muy grande o, a pesar de tener un decremento pequeño, el número de simulaciones restantes es muy bajo en comparación con las instancias empleadas.

Estas reglas deberán presentar cierta histéresis para evitar que se malgasten recursos por su temprana creación o eliminación. De forma adicional, se establecerá en las políticas de escalado un periodo de espera en el cual no se podrán generar ni destruir instancias. Queda como trabajo pendiente definir estas reglas en unas alarmas que puedan utilizarse en el servicio de monitorización.

3.3.4. Servicio de identificación

Al igual que los consumidores de los servicios de la nube computacional necesitan tener un usuario registrado para poder utilizar los servicios ofrecidos por ella, necesitaremos también unos usuarios para los servicios de nuestra aplicación que hagan uso de estos servicios. El servicio de identificación y acceso de la nube permite generar dichos usuarios y vincularles a un entorno de trabajo, para impedir que afecten al trabajo de otros usuarios. Estos usuarios tendrán una serie de permisos que les permitirán acceder a algunos de los servicios ofrecidos por la nube. En el caso de AWS, este servicio se denomina *Identity and Access Management* (IAM).

En el DNSE3, se creará un usuario para el servicio de orquestación, que tendrá acceso al servicio de almacenamiento; el servicio de colas, que tendrá acceso al servicio de monitorización; el servicio de informe, que tendrá acceso al servicio de almacenamiento, y el servicio de simulación, que tendrá acceso al servicio de almacenamiento. En el caso del servicio de simulación, el usuario se compartirá entre las distintas instancias de simulación.

En un primer momento, dado que se tenían que utilizar estos usuarios, se planteó la posibilidad de extender su uso para, además de poder acceder a los servicios de la infraestructura, autenticar las peticiones entre los servicios del DNSE3. Se habría

establecido un usuario por servicio y se le asignaría un rol por cada uno de los servicios con los que se podría comunicar. El motivo por el que se descartó esta idea fue debido a que, en ciertos entornos de nube computacional, como sucede con OpenStack, para poder comprobar la identidad de los usuarios, así como sus roles se necesita que el usuario que realiza la consulta tenga permisos de administrador, lo que presentaría problemas de seguridad en caso de capturarse las peticiones.

3.4. Conclusiones

Con este rediseño se ha conseguido paliar algunos de los problemas que presentaba la arquitectura anterior. Por un lado, se ha intentado aprovechar al máximo los diferentes servicios ofrecidos por la nube. Aunque estos servicios pueden no estar en todos los entornos de nube, sí que están presentes en aquellos entornos basados en AWS, al cual está orientado el diseño de la aplicación.

Por otro lado, se ha conseguido equilibrar la carga de trabajo del resto de servicios. Con la separación del servicio de gestión de sesiones hemos logrado, a su vez, añadir un nivel de seguridad extra en la aplicación al necesitar verificar la identidad del servicio solicitante. Algunos de los rediseños efectuados en el resto de servicios, como el producido en el servicio de informe, se deben a algunas consideraciones que se han realizado durante el rediseño del resto de servicios.

Aunque este diseño es correcto (en el sentido de que es implementable y permite lograr la funcionalidad deseada), aún presenta margen de mejora en los que se refiere a su arquitectura, repartición de trabajos y flujos de trabajo efectuados en las diferentes funciones a desempeñar. Se seguirá de momento la implementación propuesta de los diferentes servicios, aunque no se descarta la realización de un nuevo rediseño de la arquitectura y de los servicios en caso de ser necesario.

En lo que resta de documento se presentará la implementación realizada del DNSE3 hasta el momento de redacción de esta memoria y se procederá a evaluar el funcionamiento de un entorno de pruebas orientado al núcleo de la aplicación.

Capítulo 4. Implementación y pruebas de la aplicación

En la planificación de trabajo de este TFG se planteó la realización de las etapas de diseño, implementación y prueba de la aplicación DNSE3. A lo largo del capítulo Capítulo 3 se ha presentado el diseño que se va a utilizar a lo largo del resto de etapas. En este diseño se reestructuró la arquitectura de servicios de la aplicación, partiendo de la arquitectura diseñada en trabajos previos [Can12], centrándose especialmente en el aprovechamiento de los servicios ofrecidos por los *middlewares* de gestión de nube computacional, que implementan las funciones de gestión de la nube necesarias para el correcto funcionamiento del DNSE3.

Junto a este diseño, por cada uno de los servicios se planteó una posible implementación, en la que se especifica tanto las clases software que los compondrá como la API REST ofrecida. Dentro de la especificación de la API, se enumeraban los diferentes métodos HTTP soportados por cada uno de los recursos junto con los diferentes códigos de respuesta y error empleados, al igual que se mostraban las representaciones consumidas en cada uno de estos métodos.

Sin embargo, aún queda por realizar la etapa de implementación. En ella se integrarán las diferentes herramientas planteadas para su uso y se generarán versiones funcionales de la aplicación. Al tratarse de un trabajo iterativo, se empezará realizando las funciones del núcleo de la aplicación, dado que son las tareas más relevantes y primordiales para su desarrollo. En cada una de las iteraciones se irán mejorando dichas funcionalidades y ampliando para completar las funciones previstas que debe cumplir. Tras su realización, se deberá de comprobar su correcto funcionamiento, aplicando los principales escenarios de éxito y error de la aplicación. Con estas pruebas se puede hacer una retroalimentación de los diferentes fallos detectados y que se deberán de solventar en futuras iteraciones.

Sin embargo, en la etapa de implementación también se deben de configurar los entornos en los que deberá de ejecutarse la aplicación. En nuestro caso, al ser una aplicación orientada a nube con un nivel de abstracción IaaS, se deberá de configurar las diferentes instancias empleadas, incluyendo tanto la aplicación como aquellas otras con las que pueda interactuar.

En este capítulo se presentará la implementación realizada hasta el momento de la aplicación. En este estudio se detallarán los diferentes servicios implementados, aplicaciones y *frameworks* de terceros empleados y la configuración empleada para su testeo. Por otro lado, se presentarán las pruebas realizadas a la aplicación, expresando el objetivo que se desea alcanzar con cada una de ellas, y se evaluarán los resultados obtenidos. Con estos resultados se podrá analizar si la implementación actual es viable para su uso o si se deberá de reformular.

4.1. Implementación actual de la aplicación

En las diferentes iteraciones de implementación de la aplicación se han ido añadiendo las diferentes funcionalidades que debe cumplir la aplicación. Tal y como se ha indicado en la introducción de este capítulo, se ha empezado con la implementación de las funciones más relevantes: la gestión de las simulaciones a procesar. Esta primera implementación consistía en el desarrollo de los servicios de colas y simulación, comprobando la correcta recogida de las diferentes simulaciones y su notificación de cambio de estado.

Además de las funciones centrales de la aplicación, se deben de cubrir ciertos aspectos que son esenciales para su uso. Este es el caso particular del escalado de las instancias de simulación. En estas iteraciones se debe de configurar el entorno, de forma que sea compatible con la propia aplicación y presente el comportamiento esperado. En nuestro caso, debe recoger las métricas del servicio de colas y utilizarlas para determinar si se debe realizar el escalado de las instancias.

En esta sección se presentarán en primer lugar los diferentes servicios que presenten cierto grado de desarrollo y los servicios de la nube que sean compatibles con la aplicación en el momento de redacción de este informe. Se detallarán los motivos por los que no se ha completado el resto de servicios. A continuación, se mostrarán las diferentes herramientas y frameworks empleados en su implementación y, finalmente, se mostrarán las diferentes representaciones utilizadas por estos servicios.

4.1.1. Servicios implementados

En el momento de redacción de este informe, los servicios del DNSE3 que presentan cierto nivel de desarrollo son los siguientes:

El servicio de colas (SGA-01) implementa correctamente las diferentes funciones de gestión de la cola, entre las que se encuentran la adición de nuevas tareas y su posterior consulta y eliminación. Adicionalmente, integra correctamente el sistema de notificaciones *push* que permite mantener actualizado al servicio de orquestación, en lo referente al progreso de las simulaciones.

El servicio de simulación (SGA-02) recoge las tareas pendientes de procesar del servicio de colas y realiza las funciones de renovación para evitar su acceso por parte de otras instancias de simulación. En lo referente a la ejecución de las simulaciones, permite la compilación y ejecución de una única simulación a la vez, es decir, cada instancia de simulación puede procesar una simulación de forma paralela e interna, al igual que realiza el respaldo de los resultados obtenidos.

El servicio de orquestación (SPA-01) permite realizar las principales tareas de gestión de los paquetes de simulación y las descripciones de estas. Entre estas tareas se incluye la generación y consulta de los paquetes de simulación, la generación, consulta y ejecución de las descripciones y la gestión de los parámetros vinculados a ellas. Por el momento no permite realizar las tareas de eliminación de los paquetes y las descripciones, al igual que no se puede acceder de momento a las funciones de almacenamiento ni de gestión de informes. Por falta de tiempo, no se ha podido implementar una interfaz web que permitiera el uso amigable del servicio por el usuario final, aunque el servicio puede ser probado con clientes en línea de comandos.

El servicio de informe (SPA-02) permite agrupar los resultados obtenidos de las simulaciones en el formato que se haya especificado en la petición y guardarlos en el servicio de almacenamiento. Por el momento el acceso al servicio se debe de realizar por medio de un cliente HTTP y no permite generar informes en los que se procesen los resultados de las simulaciones. Este servicio ha sido el último en implementarse por lo que no recibe las peticiones procesadas del servicio de orquestación.

El resto de servicios propios no presentan un grado de desarrollo suficiente como para testear su correcto funcionamiento. En el caso del servicio de gestión de sesiones SGA-04), se está realizando la búsqueda de un proveedor de *tokens* OAuth 2.0 que pueda ser integrado en la aplicación. Este hecho ha motivado que el servicio de orquestación de momento no agrupe los paquetes de simulación en función del usuario que realiza la petición.

En lo referente al servicio de estadística (SGA-03), aunque se han encontrado diferentes herramientas que permiten la utilización de programas de cálculo matemático externos, por falta de tiempo y por la complejidad subyacente en las diferentes peticiones, no se ha podido implementar, afectando a la generación de los informes.

4.1.2. Representaciones utilizadas

Para los servicios anteriormente mencionados, se han utilizado una serie de representaciones para el acceso a sus funciones. Todas estas representaciones se han diseñado en formato JSON (*JavaScript Object Notation*) [Bra14], dado que resulta más ligero en comparación a otro tipo de representaciones, como sucede con XML, además de que ofrece el acceso directo a los datos contenidos cuando se utiliza un programa escrito en JavaScript, el cual será utilizado en la interfaz web del DNSE3.

En el caso del servicio de colas, se hará uso de tres representaciones. La primera de ellas muestra el listado de las diferentes tareas registradas en la cola. En cada una de ellas se muestra el identificador de cada tarea, además de su URI de acceso y su estado de ejecución.

```
{“tasks” :[
{
“taskID”:”identificador de la tarea”,
“status”:”estado de la tarea”,
“uri”:”URI de acceso a la tarea”
},
{
...
```

```
}
]}
```

Otra representación empleada en este servicio muestra la información de cada una de las tareas registradas. Dentro de dicha información se encuentran el identificador de tarea, además de su identificador de paquete y de descripción vinculados. Estos identificadores son compartidos con el servicio de orquestación. También incluye la URI de acceso al modelo de simulación empleado y el prefijo que se deberá de incluir en cada uno de los ficheros generados tras la ejecución de la simulación, junto con los parámetros empelados en la simulación y si se deben de guardar estos ficheros. Otros daros relevantes son el estado de ejecución, el tiempo de renovación que se utilizará para mantener la ejecución activa y el instante de tiempo a partir del cual la simulación pasará a estar disponible.

```
{
      "taskID":"identificador de la tarea",
      "input":"URI de acceso al script de la tarea",
      "outputPattern":"prefijo utilizado en los ficheros de los
resultados",
      "parameters":[
            {
                  "name":"nombre del parámetro",
                  "value":"valor del parámetro"
            },
            {...}
      ],
      "simDescriptionID":"identificador de la descripción de
simulación vinculada",
      "packageID":"identificador del paquetes de simulación
vinculado",
      "renewalTime":"tiempo de renovación de la reserva de la tarea",
      "expirationDate":"fecha y hora de vencimiento de la reserva de
la tarea",
      "uriWorker":"URI del recurso de simulación del servicio de
simulación que está procesando la tarea",
      "status":"WAITING|PROCESSING|FINISHED|MALFORMED"
      "saveTraceFile":"true|false, indica si se deben de guardar los
ficheros generados"
}
```

Para renovar las diferentes tareas, se hará uso de un documento JSON-PATCH [BN13], en el que se indicará la solicitud de renovación de la tarea además de la URI de acceso a la simulación en el servicio de simulación y el estado de ejecución en el que se encuentra. Este formato de representación se utilizará para notificar al servicio de orquestación del progreso en la ejecución de las simulaciones, indicando únicamente el estado en el que se encuentra.

```
[
      {
            "op":"replace",
            "path":"/status",
            "value":"PROCESSING|FINISHED|MALFORMED"
      },
      {
            "op":"replace",
            "path":"/uriWorker",
```

```
            "value":"URI del recurso de simulación del servicio de
simulación que está procesando la tarea"
      },
      {
            "op":"replace",
            "path":"/expirationDate"
      }
]
```

El servicio de simulación reutilizará algunas de las representaciones anteriormente mostradas. Así, cuando se consulta la simulación activa, se mostrará la representación empleada en el servicio de colas para mostrar cada una de las tareas, compartiendo en ambos casos los mismos atributos. Por otra parte, utilizará la representación en formato JSON-PATCH para realizar la renovación de las tareas de ejecución.

El servicio de orquestación, por su parte, utilizará unas representaciones para consultar los paquetes de simulación y para generar y consultar las descripciones y parámetros. En el caso de los listados de paquetes de simulación, descripciones de simulación y parámetros, la representación empleada es análoga a la utilizada en el listado de las tareas del servicio de colas, por lo que no se considera necesario mostrarlas.

Los paquetes de simulación actualmente presentan en su representación el nombre y descripción del paquete, además de la URI del servicio de almacenamiento donde se encuentra el modelo. Este dato se muestra por motivos de prueba y será reemplazado en futuras iteraciones. Adicionalmente, mostrará la URI en la que se encuentran las descripciones de simulación que tiene vinculadas.

```
{
      "packageID":"identificador del paquete de simulación",
      "name":"nombre del paquete de simulación",
      "description":"descripción del paquete de simulación",
      "scriptURI":"URI de acceso al modelo de simulación",
      "simulationDescriptions":"URI de acceso a las descripciones de
simulación vinculadas"
}
```

Para generar las descripciones de simulación, la representación JSON utilizada mostrará el nombre que asociará el usuario a la descripción, además del tipo de simulación que será y del número de veces que se repetirá su ejecución. Esta representación puede incluir también los parámetros empleados en la ejecución, que tendrá la misma estructura que la representación utilizada para generarlos y consultarlos.

```
{
      "name":"nombre de la descripción de simulación",
      "type":"SINGLE_SIM|PARAMETER_SWEEP_SIM",
      "numSimulations":#numero de repeticiones de la simulación,
      "parameters":[
            {...}
      ]
}
```

La representación de los parámetros incluye el nombre del parámetro, el tipo de parámetro que es y los posibles valores que puede tomar. En este caso, puede mostrar el único valor que puede tomar o, si se utiliza con una simulación de barrido de parámetros, los diferentes valores o el rango que debe abarcar, indicando en este último caso el incremento a utilizar. Se permite también el uso de unidades en los parámetros.

```
{
      “name”:”nombre del parámetro”,
      “type”:”ENVIRONMENT_VARIABLE|STRING_VALUE|INTEGER_VALUE|FLOAT_VA
LUE|RANDOM_INTEGER”,
      “values”:[”valores del parámetro”,...],
      “minValue”:”valor mínimo del parámetro”,
      “maxValue”:”valor máximo del parámetro”,
      “step”:”separación entre valores del intervalo”,
“unit”:”unidades del parámetro”
}
```

Cuando se consulta una descripción de simulación, esta incluye tanto su identificador como el del paquete al que está vinculada, además de su nombre, listado de parámetros, numero de repeticiones, tipo de simulación y estado de ejecución. En función del estado en el que se encuentre, mostrará adicionalmente el progreso de la ejecución, si está actualmente siendo procesada, o el tiempo de inicio y fin de ejecución, una vez ha sido completada.

```
{
      “name”:”nombre de la descripción de simulación”,
      “simulationDescriptionID”:”identificador del modelo de
simulación”,
      “packageID”:”identificador del paquete de simulación al que está
vinculado”,
“type”:”SINGLE_SIM|PARAMETER_SWEEP_SIM”,
“status”:”WAITING|PROCESSING|FINISHED|MALFORMED”,
“numSimulations”:#número de repeticiones de la simulación,
“parameters”:[
            {
“name”:”Nombre del parámetro”,
                  “type”:”Tipo de parámetro”,
                  “uri”:”URI del parámetro”
            },
            ...
]
“progress”:”Progreso de ejecución de la simulación”,
“startTime”:”Fecha de inicio de la simulación”,
“finishTime”:”Fecha de finalizado de la simulación”
}
```

El servicio de informes, al utilizar una interfaz heredada del servicio de colas, presentará un formato similar en las representaciones de los informes. Si el usuario desea agrupar los resultados obtenidos tras la ejecución, de las simulaciones, se deberá de incluir en los parámetros del informe el parámetro `traceSet`, que permite indicar la forma en la que se agruparán los valores de los parámetros de la simulación y los resultados en el fichero generado, el parámetro `paramOrder`, que indica el orden en el que aparecen los parámetros en los ficheros de resultados, y `numSimulations`, que indica el número de repeticiones que se han producido de la simulación. Esta representación de momento no incluye la inclusión de los métodos empleados en la generación del informe.

```
{
      “taskID”:”identificador de la tarea”,
      “input”:”URI de acceso al script de la tarea”,
```

```
      "outputPattern":"prefijo utilizado en los ficheros de los
resultados",
      "parameters":[
            {
                  "name":"traceSet",
                  "value":"formato del fichero de salida"
            },
            {
                  "name":"paramOrder",
                  "value":"orden de los parámetros en el prefijo del
fichero"
            },
            {
                  "name":"numSimulations",
                  "value":"número de repeticiones de cada simulación"
            },
            {...}
      ],
      "simDescriptionID":"identificador de la descripción de
simulación vinculada",
      "packageID":"identificador del paquetes de simulación
vinculado",
      "uriWorker":"URI del recurso de simulación del servicio de
simulación que está procesando la tarea",
      "status":"WAITING|PROCESSING|FINISHED|MALFORMED"
      "saveTraceFile":"true"
}
```

4.1.3. Herramientas empleadas

Para la elaboración de los diferentes servicios se ha trabajado con el *framework* Restlet Web API Framework, tal y como se comentó en la sección 2.4.3. Con este *framework* se implementará la capa de recursos de los diferentes servicios y ofrecerá el acceso a la API REST. También se utilizará para el desarrollo de los clientes REST presentes en los diferentes servicios, de forma que realicen las interacciones entre ellos.

El lenguaje empleado para la implementación de los servicios ha sido Java en su edición estándar (*Java Standard Edition*, Java SE). Se ha decidido realizar la implementación actual con esta edición en lugar de la edición Enterprise (*Java Enterprise* Edition, Java EE) por diversos motivos. El primero de ellos está orientado con el rápido despliegue de la aplicación en las etapas de prueba de las iteraciones, ya que se está más familiarizado con su uso y no se han necesitado en general las ventajas que ofrece la edición EE. Por otra parte, el uso que se hace del *framework* Restlet es prácticamente idéntico, por lo que no resultará costosa su adaptación a esta otra edición.

Entre los diferentes gestores de nubes privadas se ha utilizado OpenStack en su versión *Liberty*, trabajando, por consiguiente, con los servicios *Swift*, *Ceilometer* y *Heat* para integrar las tareas de gestión de lainfraestrucutra de la nube en la aplicación. Aunque son compatibles con el formato utilizado en los servicios de AWS, debido a ciertos problemas detectados en el trabajo conjunto de *Ceilometer* y *Heat* en las primeras etapas de desarrollo, se ha optado por utilizar el formato propio de OpenStack. Como trabajo futuro se realizará su migración al formato establecido en AWS.

Ya que el diseño e implementación de la aplicación se ha realizado de forma agnóstica al entorno de nube empleado, las comunicaciones que haga la aplicación con los servicios de la infraestructura no se realizarán de forma directa. En su lugar, se hará uso de una clase interfaz que permita acceder a los servicios de la nube de forma transparente para la aplicación. Adicionalmente, permite el desarrollo de otras clases que integren la API utilizada por otra clase de entornos de nube computacional.

El uso que se ha hecho de los diferentes servicios de la nube computacional ha el siguiente:

- En el servicio de almacenamiento de objetos se utilizará un contenedor en el cual se almacenen tanto los modelos de simulación como los resultados obtenidos tras su ejecución y los ficheros que los contenga ya agrupados. La estructura de pseudo-directorios empleada es análoga a la utilizada en el diseño de la aplicación, con la salvedad de que no se presenta de momento los directorios propios de los usuarios.
- El servicio de monitorización permitirá la publicación de la métrica `dnse3.queue.size`, la cual muestra el tamaño absoluto actual de las tareas pendientes de procesar en el servicio de colas. A su vez, se crearán unas alarmas a través del servicio de orquestación de la nube que permitan realizar un escalado bastante agresivo, de forma que se genere una nueva instancia siempre y cuando haya tareas pendientes de procesar. Se ha utilizado este tipo de alarmas en lugar de las planteadas en la etapa de diseño, según aparece en la sección 3.3.3.2, con el fin de comprobar el correcto funcionamiento de las tareas de escalado.
- El servicio de orquestación de la nube se ha configurado de forma que permita la generación de las alarmas del servicio de monitorización, al igual que las políticas de escalado y el grupo de instancias autoescalable. Estas instancias tendrán un script de inicio que permita la ejecución automática del servicio de simulación. En el apéndice B se adjuntan las plantillas empleadas. Dado que Java no permite el uso de demonios y, mientras se aprende a utilizar herramientas más avanzadas como GlassFish[38], se ha optado por utilizar Java Service Wrapper[39], un gestor de demonios de Java que permite la creación de estos sin tener que acceder al código de la aplicación. En el futuro se abandonará esta solución y se utilizará GlassFish, al ofrecer, según los autores, mejores prestaciones.

4.1.4. Configuración de las instancias empleadas

Para implementar el estado actual de la aplicación, se dispone de 10 instancias para su despliegue. De entre estas instancias, dos de ellas se han empleado para realizar tareas remotas de gestión manual y pruebas rápidas de los cambios realizados en los diferentes servicios. Otra de las instancias disponibles se configurará de forma que implemente los servicios de orquestación, colas e informe. Se han agrupado todos ellos en la misma instancia para evitar que se produjeran conflictos con las instancias mencionadas anteriormente, al estar orientadas a pruebas rápidas, e intentar aprovechar

[38] `https://glassfish.java.net/`

[39] `http://wrapper.tanukisoftware.com/`

el mayor número de instancias restantes para realizar las tareas de simulación. Cada uno de los servicios que se ejecutan en esta instancia se ha levantado y configurado de forma manual.

El resto de instancias disponibles se integrarán en el grupo de instancias autoescalable gestionado por el servicio de orquestación de la nube. Cada una de estas instancias tendrá las mismas características que el resto y utilizarán una versión del sistema operativo Ubuntu, en el cual se ha instalado y configurado tanto el servicio de simulación con Java Service Wrapper, como el simulador ns-3, en su versión 3.18, para su uso directo por parte del simulador. Se ha utilizado esta versión al presentar ciertas mejoras en comparación con la versión 3.16 empleada en las asignaturas de Teletráfico, pero manteniendo la compatibilidad con los modelos utilizados en las prácticas de dichas asignaturas.

Con todo lo mencionado a lo largo de esta sección se puede realizar realizar los casos de uso fundamentales del DNSE3, como son publicar la publicación de paquetes y descripciones de simulaciones. Adicionalmente, se ha logrado con éxito que la aplicación se aproveche del escalado automático de la nube para mejorar sus capacidades. De estos aspectos se seguirá hablando a lo largo del capítulo.

4.2. Pruebas de la aplicación

En esta sección se pretende utilizar la implementación actual del DNSE3, mostrada en la sección anterior para la realización de una serie de pruebas. El objetivo de estas, además de permitir la detección de *bugs* y problemas de ejecución, es demostrar la utilidad de la aplicación en los escenarios previstos para su despliegue. En particular, se analizará su posible uso en la realización de las prácticas de las asignaturas de Teletráfico, comprobando que se consiguen resultados similares a los alcanzados en ellas.

Estas pruebas se pueden dividir en dos conjuntos. En el primero de ellos se hace uso de los modelos de simulación empleados en las sesiones prácticas de la asignatura “Ingeniería de Teletráfico en Redes Telemáticas” del Grado en Tecnologías de la Telecomunicación de la Universidad de Valladolid para comprobar el uso de la aplicación en dichas prácticas. De forma adicional, cada una de estas prácticas hace uso de una serie de características que son necesarias para la ejecución general de las simulaciones en el ns-3: el uso de variables de entorno, para registrar los diferentes eventos producidos en la simulación, y el uso de múltiples ficheros de código fuente, cuando los modelos se estructuran en múltiples ficheros que facilitan su gestión y reusabilidad.

El otro conjunto, por su parte, pretende comprobar el rendimiento de la aplicación ante la ejecución de numerosas simulaciones en paralelo, de forma que realmente se consigue reducir el tiempo necesario para su ejecución. El modelo empleado, de nuevo, será uno de los utilizados en las sesiones de laboratorio de la asignatura “Ingeniería de Teletráfico en Redes Telemáticas”.

4.2.1. Banco de pruebas

Antes de proceder con la exposición detallada y la realización de las pruebas anteriormente mencionadas, se presentará el banco de pruebas empleado. Este consistirá en las siguientes máquinas:

- Una máquina local, con un procesador de 4 núcleos a una frecuencia de 2GHz y 8 GB de memoria RAM. Esta máquina se empleará para obtener los resultados de la ejecución habitual de las prácticas en las sesiones de laboratorio
- Un entorno de nube computacional como el anteriormente comentado en la sección 4.1.4. Cada una de las instancias empleadas contará con una única CPU a 3,2GHz y 2 GB de memoria RAM. El número de instancias máximas que puede haber en ejecución en el grupo de simulación podrá alcanzar la capacidad disponible que nos ofrece la infraestructura, siendo en este caso de 7 máquinas.

Cada una de las pruebas realizadas se realizará en la máquina local y en la infraestructura de nube. Para estudiar cada una de ellas, se contrastará el mejor escenario ofrecido por la máquina local con el peor entorno de nube. En los dos entornos se partirá de un estado inicial de reposo. En el caso de la máquina local se intentará en la medida de lo posible que no se estén produciendo trabajo en paralelo para aprovechar al máximo sus capacidades. En el caso del entorno de nube se iniciarán las pruebas cuando solo haya disponible una única instancia de simulación, estando el resto disponibles para su escalado posterior, y se analizarán las métricas del tamaño de la cola una vez cada 2 segundos, para intentar seguir en la medida de lo posible el número de tareas pendientes. Por otra parte, dado que las instancias generadas requieren de cierto tiempo para que estén completamente activas y para evitar el gasto excesivo de recursos, tras cada generación y eliminación de instancias se deberá de esperar un minuto antes de volver a comprobar el estado de la cola.

4.2.2. Realización de las prácticas de las asignaturas de teletráfico

El objetivo principal de esta prueba es la comprobación de la utilidad que se le puede dar a esta aplicación en la realización de las prácticas de las asignaturas de Teletráfico. Sin embargo, cada una de las 3 prácticas establece un requisito que debe de cumplir la aplicación.

- La primera práctica hace uso del *script* propuesto en el tutorial de uso del ns-3. Este *script*, adjunto en el apéndice C, se utiliza para aprender la estructura de los modelos del simulador, al igual que muestra cómo se puede acceder al registro de los diferentes eventos producidos en la simulación. Este registro resulta accesible por medio de las variables de entorno empleadas en el momento de ejecución. En esta prueba se comprobará, por tanto, que el simulador puede hacer uso de las variables de entorno. De forma nativa, el *script* solicita el registro de los instantes de tiempo en los que se producen las llegadas y salidas de los mensajes enviados en la comunicación.
- La segunda práctica hace uso del script facilitado por el profesor J.I. Asensio durante el transcurso de la segunda parte de la práctica 2 [AP16a]. En dicho *script*,

adjunto en el apéndice C, se modela la red mostrada en la Figura 4.1 y se permite la libre configuración de diferentes parámetros que definen la red, como puede ser el retardo extremo a extremo en la comunicación, el tipo de mensajes enviados (de longitud fija o distribuida exponencialmente), la tasa de generación de estos mensajes y el tamaño del buffer de memoria del transmisor, entre otros. En dicha práctica se proponía a los alumnos estudiar el comportamiento de la red para los diferentes parámetros que se pueden configurar y la comparación de los resultados obtenidos con los modelos que ofrece la teoría de colas estudiados en las sesiones de teoría.

Para realizar esta práctica, era necesario realizar una serie de simulaciones de barrido de parámetros, obteniendo resultados para cada una de las muestras estudiadas. En este caso, las muestras hacen referencia a las diferentes combinaciones de parámetros empleadas. En nuestra aplicación, tras la ejecución de esta clase de simulaciones, se obtiene una serie de ficheros disgregados que contiene cada uno el resultado de una de las ejecuciones. El objetivo de esta prueba es que la aplicación sea capaz de agrupar dichos resultados en un único fichero, de fácil lectura y uso por parte del usuario final.

- La tercera práctica hace uso del script facilitado por el profesor J.I. Asensio durante el transcurso de la práctica 3 [AP16b], además de las librerías ofrecidas por Ramroop [Ram10] para realizar el modelado de nodos *DiffServ*. En dicha práctica se analizaba el comportamiento de la red de la Figura 4.2 cuando se aplicaban diferentes políticas y algoritmos que permiten ajustar la QoS ofrecida en dicha conexión. Entre los algoritmos empleados nos encontramos el uso de *token bucket* para conformar el tráfico emitido y el uso del algoritmo WRR (*Weighted Round Robin*) para repartir la capacidad del enlace entre los diferentes tipos de tráfico cursado.

  En esta práctica se hace uso de una serie de librerías externas no incluidas en la instalación estándar del simulador, por lo que se deben adjuntar al modelo para permitir su uso. El objetivo que se desea alcanzar con esta prueba es la correcta ejecución de las simulaciones que presentan más de un fichero de código fuente.

A continuación, se procederá con la ejecución de cada una de ellas y el estudio de sus resultados.

Figura 4.1: Modelo de red simulada en el script empleado en la práctica 2.1 de "Ingeniería de Redes y Servicios Telemáticos"

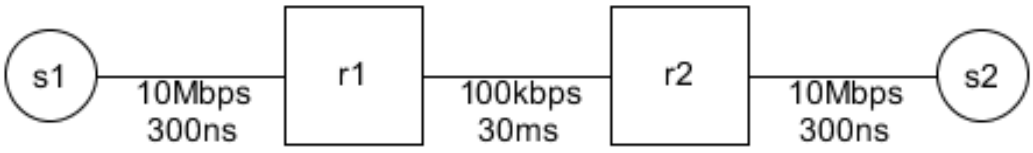

Figura 4.2: *Modelo de red simulada en el script empleado en la práctica 3 de "Ingeniería de Redes y Servicios Telemáticos".*

4.2.2.1. Uso de las variables de entorno

Tras realizar la carga del modelo empleado en esta prueba, se ha procedido con la creación de la simulación, enviando una petición HTTP POST al recurso *SimulationDescriptions* asociados al paquete empleado en esta prueba. El cuerpo de esta petición incluye el siguiente documento JSON

```
{
     “name”:”Prueba variables de entorno”,
     “type”:”SINGLE_SIM”,
     “numSimulations”:1,
     “parameters”:[
          {
               “name”:”NS_LOG”,
               “type”:”ENVIRONMENT_VARIABLE”,
          “value”:”UdpEchoClientApplication=level_all|
             prefix_func:UdpEchoServerApplication=leve
             l_all|prefix_func”
          }
     ]
}
```

En esta petición se solicita la ejecución de una única simulación que recibirá como parámetros la variable de entorno `NS_LOG`, que permitirá el registro de los eventos producidos por la aplicación `UdpEchoClient` y `UdpEchoServer`.

El resultado de esta simulación realizada en una máquina local es el siguiente:

```
     0s UdpEchoClientApplication:UdpEchoClient(0xf74770)
     0s      UdpEchoClientApplication:SetDataSize(0xf74770,
1024)
     1s UdpEchoServerApplication:StartApplication(0xf742f0)
     2s UdpEchoClientApplication:StartApplication(0xf74770)
     2s UdpEchoClientApplication:ScheduleTransmit(0xf74770,
+0.0ns)
     2s UdpEchoClientApplication:Send(0xf74770)
     2s At time 2s client sent 1024 bytes to 10.1.1.2 port
9
     2.00369s UdpEchoServerApplication:HandleRead(0xf742f0,
0xf74a30)
```

```
     2.00369s At time 2.00369s server received 1024 bytes
from 10.1.1.1 port 49153
     2.00369s Echoing packet
     2.00369s At time 2.00369s server sent 1024 bytes to
10.1.1.1 port 49153
     2.00737s UdpEchoClientApplication:HandleRead(0xf74770,
0xf79a70)
     2.00737s At time 2.00737s client received 1024 bytes
from 10.1.1.2 port 9
     10s UdpEchoClientApplication:StopApplication(0xf74770)
     10s UdpEchoServerApplication:StopApplication(0xf742f0)
     UdpEchoClientApplication:DoDispose(0xf74770)
     UdpEchoServerApplication:DoDispose(0xf742f0)
     UdpEchoClientApplication:~UdpEchoClient(0xf74770)
     UdpEchoServerApplication:~UdpEchoServer(0xf742f0)
```

Si procedemos con la ejecución de esta simulación en el DNSE3, obtendremos el siguiente resultado.

```
     At time 2s client sent 1024 bytes to 10.1.1.2 port 9
     At time 2.00369s server received 1024 bytes from
10.1.1.1 port 49153
     At time 2.00369s server sent 1024 bytes to 10.1.1.1
port 49153
     At time 2.00737s client received 1024 bytes from
10.1.1.2 port 9
```

Como se puede observar, el resultado obtenido no es el esperado al no registrar los eventos indicados en la variable de entorno `NS_LOG`. Esto puede ser debido a que el domino del servicio de simulación ejecutado gracias a Java Service Wrapper esté siendo ejecutado en un proceso que impida que se modifiquen las variables de entorno de los procesos hijo que genere. Esto es una suposición que se intentará corroborar al migrar la aplicación a Java EE y GlassFish, pero se seguirán buscando otras posibles causas y soluciones. La parte positiva de los resultados obtenidos es que se recoge adecuadamente la salida de error de las simulaciones, que es empleada por el ns-3 para registrar los eventos, tanto establecidos en las variables de entorno como dentro del *script*, como sucede en esta prueba.

4.2.2.2. Agrupación de los resultados de una simulación

Tras realizar la carga del modelo empleado en esta prueba, se ha procedido con la creación de la simulación, enviando una petición HTTP POST al recurso *SimulationDescriptions* vinculado al paquete de simulación empleado en la prueba. El cuerpo de la petición incluye el siguiente documento JSON:

```
{
     “name”:”Prueba variables de entorno”,
     “type”:”PARAMETER_SWEEP_SIM”,
     “numSimulations”:5,
```

```
        "parameters":[
            {
                "name":"tmax",
                "type":"INTEGER_VALUE",
            "value":10000
            },
            {
                "name":"meanPacketSize",
                "type":"INTEGER_VALUE",
                "minValue":200,
                "maxValue":1000,
                "step":200
            },
            {
                "name":"rep",
                "type":"RANDOM_INTEGER"
            }
        ]
    }
```

En esta petición se solicita la ejecución de una simulación de barrido de parámetros, que tiene como parámetros `tmax`, que define el tiempo de simulación, `rep`, que define la semilla empleada en las simulaciones y toma un valor aleatorio, y `meanPacketSize`, que define la longitud media en bytes de los paquetes transmitidos por la red. Este último parámetro es el que se desea barrer, recorriendo el intervalo de 200 a 1000 con incrementos de 200 para cada valor intermedio. El tamaño de las muestras obtenidas para cada simulación es igual a 5, según indica el atributo `numSimulations`. La ejecución de esta simulación deberá producir un total de 25 ficheros que contengan cada uno el resultado de cada simulación. A continuación, en la Tabla 4.1se presenta el listado de los diferentes ficheros generados, junto con el contenido de cada uno de ellos.

Nombre del fichero	Contenido del fichero
10000-0200-1-output.trace	0.0289362
10000-0200-2-output.trace	0.0291193
10000-0200-3-output.trace	0.0291191
10000-0200-4-output.trace	0.0301703
10000-0200-5-output.trace	0.0287881
10000-0400-1-output.trace	0.167254
10000-0400-2-output.trace	0.160818
10000-0400-3-output.trace	0.170081
10000-0400-4-output.trace	0.162687
10000-0400-5-output.trace	0.160147
10000-0600-1-output.trace	0.50145
10000-0600-2-output.trace	0.511437
10000-0600-3-output.trace	0.515209
10000-0600-4-output.trace	0.509462
10000-0600-5-output.trace	0.495164
10000-0800-1-output.trace	1.26276
10000-0800-2-output.trace	1.25634

10000-0800-3-output.trace	1.3146
10000-0800-4-output.trace	1.29375
10000-0800-5-output.trace	1.24359
10000-1000-1-output.trace	3.13113
10000-1000-2-output.trace	3.27672
10000-1000-3-output.trace	3.13535
10000-1000-4-output.trace	3.12818
10000-1000-5-output.trace	3.19733

Tabla 4.1: *Ficheros generados tras la ejecución de la simulación de la práctica 2.2 de la asignatura "Ingeniería de Tráfico en Redes Telemáticas".*

Al estar cada uno de los resultados separados en múltiples ficheros, se deberán agrupar en un fichero que permita su uso por parte de los usuarios. Para generarlo, utilizaremos el servicio de informe. Dado que en el momento de redacción de este informe el servicio de orquestación no presenta los recursos necesarios para realizar esta carga, se realizará una petición directa al servicio de informe. Esta petición consistirá en una llamada al método HTTP POST al recurso *Reports* del servicio de informe, en cuyo cuerpo se incluirá el siguiente documento JSON:

```
{
"input":"output",
"outputPattern":"mean packet size sweep.trace",
     "parameters”:[
     {
          “name”:”traceSet”,
          “value”:”meanPacketSize output”
     },
     {
          “name”:”paramOrder”,
          “value”:”tmax meanPacketSize”
     },
     {
          “name”:”numSimulations”,
          “value”:”5”
     }
],
"packageID":"s2t1in4",
"simulationID":"66rpad5",
"saveTraceFile":"true"
}
```

En dicha petición se solicita la generación de un fichero de traza a partir de los resultados de la simulación, como se indica en el atributo input. En este fichero generado cual la primera columna de cada fila se corresponderá con el parámetro `meanPacketSize` y el resto de la fila contendrá cada uno de los valores de la muestra vinculada a dicho parámetro, según se especifica en el parámetro `traceSet`. Para que el servicio de informes conozca el orden de los parámetros que aparece en el nombre de cada fichero, se le indica la asociación de cada uno de ellos por medio del parámetro `paramOrder`, al igual que se le indica el número de simulaciones, en el parámetro

`numSimulations`, que se han efectuado para que pueda recoger los ficheros suficientes. Para que los pueda recoger de forma correcta, los ficheros están correctamente ordenados alfabéticamente. Se adjunta además tanto el identificador de paquete como el identificador de la simulación a la que están vinculados los resultados. Tras enviar dicha petición, el servicio de informe procesará la petición y se podrá acceder al fichero generado a través del servicio de almacenamiento. A continuación, se muestra el contenido de dicho fichero.

Nombre del fichero: `generatedTrace-mean packet size sweep.trace`

```
0200 0.0289362 0.0291193 0.0291191 0.0301703 0.0287881
0400 0.167254 0.160818 0.170081 0.162687 0.160147
0600 0.50145 0.511437 0.515209 0.509462 0.495164
0800 1.26276 1.25634 1.3146 1.29375 1.24359
1000 3.13113 3.27672 3.13535 3.12818 3.19733
```

Como se puede observar, se han agrupado correctamente los valores de cada una de las muestras obtenidas. Este fichero podrá ser utilizado posteriormente en otros programas para el tratamiento de dichos datos.

4.2.2.3. Uso de múltiples ficheros de código fuente

Tras realizar la carga del modelo empleado en esta prueba, en esta ocasión siendo un archivo comprimido `.zip` que contiene todos los ficheros necesarios, se ha procedido con la creación de la simulación, enviando una petición HTTP POST al recurso *SimulationDescriptions* asociado al paquete empleado en esta prueba. El cuerpo de dicha petición contiene el siguiente documento JSON:

```
{
     “name”:”prueba de multiples ficheros”.
     “type”:”SINGLE_SIM”,
     “parameters”: [
          {
               “name”:”cir”,
               “type”:”INTEGER_VALUE”,
               “value”:37080
          }
     ]
}
```

En esta petición se está solicitando la creación de una única simulación que utilice los parámetros por defecto del modelo simulado a excepción del parámetro `cir`, que define el parámetro CIR (*Commited Information Rate*) del algoritmo *token bucket*. Este parámetro define la tasa de regenarión de los *tokens* empleados en el conformado del tráfico y tomará un valor de 37080 kbps. Tras ejecutar dicha simulación, se han almacenado en el servicio de almacenamiento los ficheros de traza generados por la simulación, los cuales se muestran a continuación:

- `37080-P3-DiffServ-s1-arrivals.trace`

- `37080-P3-DiffServ-s2-arrivals.trace`
- `37080-r1DropNonConformantAFBE.trace`

Estos ficheros contienen los instantes de tiempo en los cuales se generan paquetes en el nodo s1 y en los que llegan paquetes al nodo s2, respectivamente, de la red modelada. Una vez descargados, se pueden utilizar con el script de GNU Octave utilizado en dicha práctica, adjunto en el apéndice C. Con dicho script podemos generar gráficas como la mostrada en la Figura 4.3, que muestra la tasa de paquetes que se generan y reciben en los extremos de la comunicación, además de una tercera gráfica que muestra los paquetes descartados en el nodo r1 de la red modelada debido a la aplicación de *token bucket*.

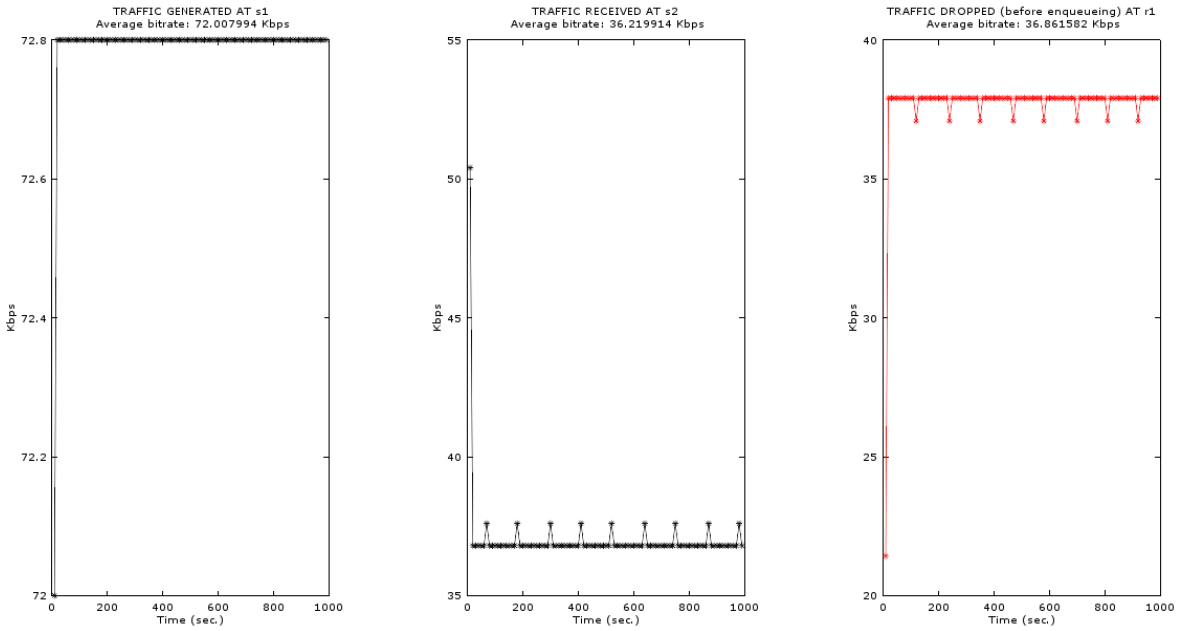

Figura 4.3: *Gráfica generada con los resultados obtenidos tras la ejecución de la simulación de la práctica 3.*

El resultado de esta tercera prueba resulta satisfactorio, aunque se ha detectado un problema con el fichero comprimido empleado. Solo permite el uso de ficheros comprimidos que presenten los ficheros de código fuente del modelo en el primer nivel de los directorios contenidos. Esto es debido al fichero `makefile` empleado en el servicio de simulación, que no contempla esta clase de casos y se deberá solucionar en futuras iteraciones, utilizando una lista que indique la ruta en la que se encuentran estos ficheros.

4.2.2.4. Conclusiones de la prueba

Tras la ejecución de las pruebas anteriores, a excepción del uso de las variables de entorno, se ha conseguido realizar las diferentes prácticas propuestas. A falta de implementar una interfaz usable por los alumnos que les permita realizar las peticiones anteriores, se ha comprobado su aplicación en la asignatura y se podría plantear realizar el despliegue una vez terminada la implementación de todas las funcionalidades.

4.2.3. Tiempo de ejecución de simulaciones en paralelo

En esta prueba se ha intentado comprobar que la aplicación consigue reducir el tiempo de ejecución necesario para la obtención de los resultados de las diferentes simulaciones. Para ello, se han preparado 4 bancos de pruebas, en los cuales se utilizaba el modelo utilizado en la primera parte de la práctica 2 de "Ingeniería de Teletráfico en Redes Telemáticas", el cual se muestra en el apéndice C, para realizar un bloque de 5, 10, 50, 100, 500 y 1000 simulaciones, todas ellas presentando los mismos parámetros de entrada. El hecho de que presenten los mismos parámetros de entrada hace que el tiempo de ejecución de todas y cada una de las simulaciones sea parecidos Si se presentase algún tiempo de ejecución diferente en alguna de las posibles ejecuciones a realizar, podría camuflar los resultados obtenidos al realizar la repartición de tareas entre las instancias.

En la ejecución de esta prueba se ha modificado el número de instancias que puede alcanzar como máximo el grupo de simulación, fijándose dicho límite en 1, 4 y 7 instancias. El propósito de establecerlos nos permite estudiar el comportamiento de la aplicación ante diferentes instancias de simulación que compitan por la ejecución de las simulaciones.

El modelo empleado en esta prueba modela un sistema de colas tal y como se muestra en la Figura 4.4. En dicho sistema se generan los distintos paquetes con un tiempo entre generación distribuido exponencialmente y estos se envían a una cola que será consultada por un servidor. Este script permite modelar los modelos de colas M/M/1 y M/D/1 según la notación de Kendall [Ken53], al tener los paquetes generados una longitud distribuida exponencialmente y fija, respectivamente.

Figura 4.4: *Modelo de colas simulado en el script de la práctica 2.1 de la asignatura "Ingeniería de Teletráfico en Redes Telemáticas".*

Los parámetros empleados en esta simulación son `tmax`, para definir la duración de la simulación, fijado en 10000 segundos para tener resultados con una varianza aceptable, y `rep`, que define una semilla aleatoria para cada una de las simulaciones. El estudio del tiempo de simulación a utilizar se realizó durante el desarrollo de dicha práctica. La petición enviada para realizar estas simulaciones en el DNSE3 se muestra en el siguiente documento JSON.

```
{
     "name":"prueba multiples simulaciones",
     "type":"SINGLE_SIM",
     "parameters":[
          {
               "name":"tmax",
               "type":"INTEGER_VALUE",
```

```
                "value":"10000
            },
            {
                "name":"rep",
                "type":"RANDOM_INTEGER"
            }
        ],
        "numSimulations":XXX
}
```

Donde `XXX` define el número de simulaciones a efectuar, perteneciendo a alguno de los valores indicados previamente.

En la Figura 4.5 se muestran los tiempos obtenidos tras la ejecución de las diferentes simulaciones.

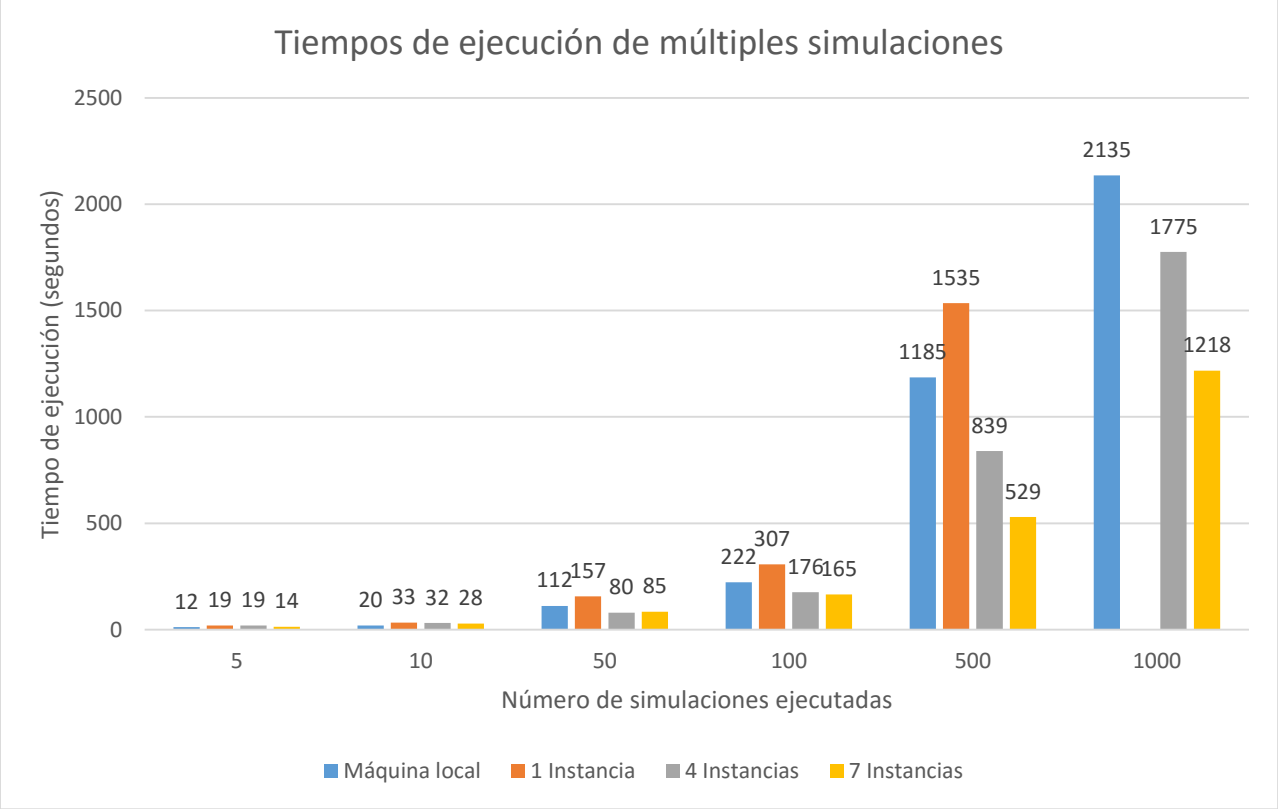

Figura 4.5: *Tiempos de ejecución obtenidos tras la simulación de 5, 10, 50, 100, 500 y 1000 simulaciones en diferentes entornos de pruebas.*

En el caso de una única instancia no se ha podido realizar la prueba de 1000 simulaciones debido a que el servicio de simulación cesaba su funcionamiento cuando el número de simulaciones seguidas que debía realizar era elevado. Tras la ejecución manual de dicho servicio se ha comprobado que este error se debe al uso de Java Service Wrapper que, por razones desconocidas, considera oportuno finalizar la ejecución del servicio. Al igual que se ha comentado en ocasiones anteriores, esta herramienta se sustituirá por GlassFish en futuras iteraciones de la aplicación, donde se estudiará de nuevo dicho rendimiento.

Se puede observar que, cuando se deben de ejecutar pocas simulaciones, debido a los intercambios de peticiones realizadas entre los diferentes servicios, la aplicación tarda un poco más en comparación a su ejecución en una máquina local. Sin embargo, a medida que aumenta el número de simulaciones a ejecutar, el tiempo de ejecución se ve reducido en comparación con el ofrecido por la máquina local, siempre y cuando se utilice más de una instancia en la nube.

En relación al número de instancias empleadas, se puede observar que los tiempos de ejecución no dependen mucho de este número cuando las simulaciones a ejecutar son pocas. Esto se debe a que se parte de un estado de reposo, por lo que se necesita de un cierto tiempo para que se generen el resto de máquinas empleadas y se muestre el rendimiento de la aplicación.

En la ejecución de las simulaciones en el entorno del DNSE3 hay que tener en cuenta que se necesita compilar la simulación, defininida por el modelo contenido en el paquete de simulación, cada vez que se quiera realizar, definida por la descripción de simulación. Aunque dicho tiempo puede no ser significativo en algunas de las ocasiones, cuando el número de simulaciones a ejecutar es elevado se produce un incremento del tiempo total de ejecución. Una forma de poder solucionar este problema sería que los clientes de simulación, antes de realizar la descarga del modelo, comprobaran que existe ya una versión compilada. Dado que todas las instancias de simulación comparten las mismas características, estos ficheros ya compilados se podrán ejecutar en cualquiera de ellas. En caso de no haber ninguna versión compilada, el cliente de simulación será el encargado de añadirla en el servicio de almacenamiento para su uso por parte de las otras instancias de simulación. Antes de realizar dicha carga se deberá comprobar que no existe ya una versión ya preparada por alguna de las otras.

En relación al rendimiento presentado por la aplicación cuando hay múltiples instancias en ejecución se puede producir problemas de rendimiento al producirse solapamientos en las peticiones de simulaciones pendientes. Esta situación se produce al consultar la tarea siguiente mientras hay otra instancia comprobando la existencia del modelo, y, dada la implementación actual de la cola, se le devuelve la misma tarea. Esta situación estaba prevista en el diseño del servicio de colas y, de acuerdo con dicho diseño, se conseguía que cada una de las instancias de simulación ejecutase una simulación diferente del resto por medio de su reserva, pero no evita que las otras instancias intenten solicitar dicha tarea. El efecto visible es que se alcanzan tiempos de ejecución de todas las simulaciones bastante similares a los alcanzados con un menor número de instancias. Una forma de poder solucionar este problema sería implementar una asignación de tareas por medio de *Round-Robin*, de forma que se ofrecieran diferentes tareas cuando haya muchas tareas disponibles.

4.3. Fallos detectados

A lo largo de la ejecución de las diferentes pruebas, se han detectado una serie de *bugs* menores que pueden afectar a la ejecución del sistema. El primero de ellos es la no transmisión al usuario de los errores producidos en algunos de los intercabios de mensajes realizados entre los servicios. Este fallo se ha detectado cuando, por ejemplo, no se tenía acceso a los servicios de la nube, indicando al usuario el éxito de la operación. Se deberá

de evaluar la captura de estos fallos en las peticiones y enviar al usuario una representacón en la cual se le indique tanto el error como su causa.

Otro de los problemas detectados que no se ha mencionado en las pruebas anteriores es la captura de los parámetros de una simulación por parte del servicio de informe, cuando dichos parámetros contienen el carácter '-', como sucede con los números negativos. Dado que ese carácter es empleado en la división de los diferentes parámetros empleados, se produce una confusión en los parámetros recogidos. Para solventar este problema se puede utilizar otro carácter de uso poco frecuente por los usuarios en las simulaciones, bastante difícil de lograr dado el gran abanico de posibles parámetros empleados. Otra solución podría ser el uso de un carácter de escape, de forma que se pueda diferenciar el carácter empleado en la división de los parámetros de los contenidos en ellos.

4.4. Conclusiones

La implementación actual del DNSE3 no cubre todas las funciones que debe ofrecer a los usuarios, por lo que aún no es apta para su despliegue final. Sin embargo, de las funcionalidades que presenta actualmente, se ha podido comprobar que se pueden utilizar en entornos reales. A excepción de una de las pruebas efectuadas, los resultados obtenidos han sido positivos en su realización. Se puede pensar que, al haber utilizado los modelos de las prácticas de Teletráfico, el uso que se le podría dar al DNSE3 se limita a dichas prácticas. Sin embargo, con cada una de las pruebas se cubrían escenarios de uso que se pueden dar en situaciones más allá del alcance de dicha asignatura. Se ha estudiado el comportamiento del simulador ante la ejecución de múltiples simulaciones para agrupar los resultados y ante modelos con múltiples ficheros de código fuente. La única limitación presente en el uso del DNSE3 es que, en el caso de ejecutar múltiples simulaciones, ya sea por barrido de parámetros o por repetición de alguna de ellas, el resultado de su ejecución debe aparecer por la salida estándar o de error. De no ser así, no se podrán capturar los resultados por la forma en la que ha sido implementado el cliente de simulación.

Otro aspecto a considerar en la implementación de la aplicación es que se ha planteado de forma que sea agnóstica a la tecnología implementada. Esto incluye también al simulador empleado. La única configuración que se ha realizado con el simulador es su correcta instalación en las instancias de simulación, de forma que se puedan utilizar las librerías que incorpora. Se podría plantear extender la utilidad del DNSE3 para que permita su uso con otros simuladores y se pueda utilizar en un mayor número de entornos.

Si nos volvemos a centrar en la realización de las pruebas, se ha comentado que cada una de las instancias de simulación era capaz de ejecutar una única simulación a la vez, al igual que se empleaba un sistema de escalado bastante agresivo. Debido a esto, el rendimiento alcanzado con cada uno de los recursos utilizados no ha sido óptimo. En futuras iteraciones de trabajo se deberá de incorporar el trabajo en paralelo de las simulaciones en cada una de las instancias al igual que el uso de reglas de escalado más avanzadas, como se contempló en el diseño de la aplicación en el capítulo Capítulo 3. Aun así, los resultados obtenidos son bastante esperanzadores y permiten seguir trabajando con la aplicación.

Capítulo 5. Conclusiones

Los simuladores resultan una gran herramienta en el campo de la investigación. A pesar de sus limitaciones, como son las estimaciones de los resultados, resultan una alternativa interesante en comparación a otras técnicas, en las que no se pueden realizar estudios complejos del escenario o bien su uso resulta inviable. También resulta interesante su uso en entornos educativos, donde ofrecen una serie de ventajas que motivan su uso frente a otros modelos de enseñanza. Por un lado, se evita el uso de formulaciones que pueden resultar tediosas para los alumnos tras su uso continuo. Por otra parte, su uso no se ve limitado al material disponible en el laboratorio, permitiendo realizar prácticas fuera de las sesiones de laboratorio.

El principal problema que presenta su uso es el elevado tiempo necesario para realizar las simulaciones de barrido de parámetros. La realización de estas simulaciones se suele realizar ejecutando diferentes simulaciones con conjuntos de parámetros distintos, pero dentro de un rango específico. Dentro de las aulas, puede suponer un problema al requerir demasiado tiempo para obtener los resultados con los que poder trabajar, pudiendo llegar a superar el periodo disponible en las sesiones de laboratorio. Este problema se ha experimentado, por ejemplo, en algunas de las asignaturas de la rama de Telemática en las diferentes titulaciones de telecomunicaciones ofrecidas en la Universidad de Valladolid.

Aunque los simuladores actuales presentan herramientas que permiten reducir este tiempo, no resultan lo suficientemente efectivas cuando se incrementa el número de simulaciones. De esta forma surgieron aplicaciones como el DNSE y el DNSE3 que permiten su ejecución en entornos de grid y nube computacional respectivamente. En este TFG se ha partido del traajo realizado en estas dos aplicaciones y se ha propuesto un nuevo diseño de la aplicación DNSE3.

Tras la realización de este TFG, la aplicación presenta una arquitectura orientada a servicios, en la que se reparten las diferentes funciones de la aplicación en diferentes servicios que se comunican entre sí para efectuar el trabajo global. Este tipo de diseño resulta adecuado en entornos distribuidos, ya que permite una mejor flexibilidad en el desarrollo y despliegue de las diferentes funcionalidades del sistema, al igual que permite su reparto entre las diferentes máquinas que conforman el entorno. Todos estos servicios, a su vez, presentaban un diseño RESTful en el que se propone el uso de una API REST para ofrecer las diferentes funciones del servicio, accesibles a través de los recursos ofrecidos. Las ventajas que ofrece este diseño en comparación a otros como puede ser

SOAP, son la mayor flexibilidad en el diseño de los servicios, al igual que ofrecen un mejor escalado al no mantener ninguna información relativa a los usuarios activos. Por otro lado, también presenta un uso más eficiente de las representaciones enviadas en las diferentes peticiones, al ofrecer un formato variable y al ser, por lo general, livianas, lo que resulta apropiado en un entorno de nube donde se paga por el uso de los recursos empleados. También ayuda a mantener un sistema homogéneo ya que la mayoría de gestores de nube computacional ofrecen sus servicios a través de una API REST.

Esta arquitectura propuesta permite realizar cada una de las funciones que debe desempeñar la aplicación. Sin embargo, presenta alguna serie de problemas a nivel de diseño que deberán de cambiarse en futuras iteraciones de la aplicación. Una de las más relevantes es la gestión de la cola de tareas del servicio de colas. Este servicio ofrece una cola de tipo FIFO en la que se registran los diferentes trabajos que debe procesar la aplicación. El problema que plantea este diseño es el acceso a los trabajos pendientes. Si se utiliza la cola diseñada hasta ahora, los usuarios que deseen realizar multitud de simulaciones podrían acaparar el uso de la propia aplicación e impedir que simulaciones de menor tamaño de otros usuarios, a pesar de haberse mandado más tarde, experimenten mayores tiempos de ejecución. Una posible solución a este problema sería realizar una planificación *round-robin* a nivel del usuario solicitante, que impediría la apropiación del sistema ante peticiones de gran envergadura.

Adicionalmente, uno de los objetivos propuestos en el rediseño de la arquitectura era la integración de diferentes servicios ofrecidos por los gestores de nube computacional para realizar aquellas tareas que implicaran gestionar la infraestructura utilizada. De los diferentes servicios propuestos, se ha conseguido que la aplicación pueda hacer uso de ellos de forma correcta. Su integración se ha realizado de forma transparente a la implementación de los diferentes servicios, con el propósito de, además de ofrecer compatibilidad con los servicios ofrecidos por AWS, poder llegar a ofrecer compatibilidad con otros entornos de nube siempre y cuando se estimará oportuno y provechoso su uso.

El diseño que se ha planteado de la arquitectura se ha centrado en el uso por parte de los usuarios finales, de forma que se cubran sus necesidades. Sin embargo, no se ha tratado en ningún momento las tareas de administración de la aplicación, que permitan controlar el uso que se hace de la propia aplicación. Algunas de las tareas que se podrán integrar son el acceso a las diferentes funciones de la aplicación, pudiendo restringir su uso o limitarlo, y la monitorización y gestión de los servicios desplegados, para comprobar el correcto funcionamiento global de la aplicación.

Si continuamos con la implementación de la aplicación, se han conseguido implementar aquellos servicios que realizan las funciones del núcleo de la aplicación, es decir, la ejecución de las simulaciones. Esta implementación se ha evaluado para comprobar su utilidad en escenarios genéricos de uso que cubran diferentes aspectos con los que deba tratar la aplicación. A pesar de haber obtenido resultados favorables en las pruebas, el margen de mejora presente es elevado. Algunos de dichos aspectos atañen a la contienda de las instancias de simulación, que se deberá conseguir que no se les ofrezca la misma simulación ante peticiones próximas en el tiempo, y al procesado de las simulaciones, que infrautiliza los recursos ofrecidos por la propia infraestructura.

A pesar de todas estas limitaciones, se ha logrado alcanzar una serie de hitos que permiten continuar con su desarrollo. Aparte de la ejecución distribuida por parte de las instancias de simulación, se ha conseguido aprovechar los servicios ofrecidos por la nube para realizar diferentes tareas, entre las que se encuentran el almacenamiento remoto de ficheros, la monitorización de los servicios propios de la aplicación y el escalado automático de los recursos, vital para el despliegue final de la aplicación. A partir de las métricas enviadas al servicio de monitorización, se han podido establecer una serie de reglas que permiten adaptar los recursos de la infraestructura a la demanda de simulaciones.

Para alcanzar un estado que permita su despliegue en un entorno real, se deberán realizar una serie de iteraciones que permitan la implementación de los servicios pendientes de desarrollo al igual que la integración de las carencias encontradas en la implementación actual. Algunas de estas carencias son la falta de una interfaz web que permita el acceso y uso de la aplicación y la transmisión de los problemas originados en el intercambio de peticiones entre los servicios y sus causas. Es debido a todos estos factores que se propone una posible planificación para completar su despliegue.

5.1. Trabajo futuro

De los diferentes servicios pendientes de implementar que son críticos para el correcto despliegue de la aplicación es el servicio de gestión de sesiones. Aunque se pueda realizar una implementación propia de este servicio, tal y como se comentó en la sección 3.2.4 se deberá buscar un posible proveedor de *tokens* OAuth 2.0 que pueda ser utilizado en la aplicación. Se considera oportuno dedicar un plazo de 2 a 3 semanas para localizarlo mientras se completan el resto de funcionalidades.

A su vez, se deberá de realizar una implementación del servicio de estadística que permita la generación de los informes solicitados por los usuarios. La primera propuesta de implementación será el uso de *frameworks* que permitan acceder al software de cálculo matemático GNU Octave y comprobar que ofrece el rendimiento esperado. Sin embargo, se planteará el desarrollo de *scripts* escritos en C/C++, procedentes de los *scripts* de estos programas de cálculo matemático y se determinará la solución que sea más adecuada para la aplicación. Se prevé que esta tarea pueda llevar en torno a 2 semanas su correcta implementación y uso por parte del servicio de simulación.

Una vez se hayan cubierto las diferentes funciones de la aplicación, se migrará la implementación a Java EE y se ajustará el funcionamiento de los diferentes servicios para mejorar el rendimiento ofrecido. Se propone el siguiente orden para la implantación de las mejoras, el cual puede ser modificado por aquellas personas que las integren. En primer lugar, se mejorará el uso de las instancias por parte de los servicios de simulación, por lo que se tratará de ampliar el número de simulaciones a ejecutar por parte de cada una de las instancias y se integrarán las reglas de escalado avanzadas propuestas en el diseño de la aplicación. A continuación, se mejorará la gestión de las tareas del servicio de colas, integrando la planificación *round-robin* para evitar el intento de ejecución de la misma simulación en peticiones próximas temporalmente y el acaparamiento de la aplicación por parte de los usuarios que soliciten tareas de simulación que necesiten de una considerable cantidad de tiempo. De forma paralela a la realización de estas mejoras

se diseñará una interfaz web usable y accesible que permita acceder a las diferentes funciones ofrecidas por el DNSE3 de manera intuitiva, al igual que la incorporación de ficheros de configuración para aquellos servicios que requieran mantener datos entre diferentes sesiones de trabajo. Se considera adecuado dedicar un tiempo mínimo de 3 semanas para realizar la mejor gestión de las simulaciones y del acceso a los trabajos pendientes.

Por otra parte, será necesario migrar dicha configuración a la ofrecida por AWS para permitir la distribución de la aplicación y su uso en el mayor número de entornos posibles, ya que hasta ahora se ha estado utilizando la configuración de OpenStack. Este cambio de configuración implica modificar el acceso que se hace a los servicios de la nube, el diseño de las métricas empleadas, así como de las plantillas utilizadas en el servicio de orquestación. Se prevé que se requiera en torno a 2 semanas de trabajo para realizar esta puesta a punto.

Además de la correcta implementación de las funciones anteriores, se deberán introducir medidas de seguridad en el uso de la aplicación. Entre estas se encuentra la ejecución de las simulaciones en un entorno protegido, no realizado hasta ahora al utilizarse modelos de origen y funcionamiento conocidos. Adicionalmente, se utilizará de forma conjunta la capa de aplicación HTTPS y el protocolo de autenticación OAuth 2.0 para proteger tanto las comunicaciones de los usuarios, impidiendo en la medida de lo posible la usurpación de la identidad de estos, como las comunicaciones internas de los servicios propios de la aplicación, de forma que se evite la inserción de peticiones dañinas para su funcionamiento.

Una vez completado la implementación del DNSE3, se procederá a realizar la etapa de prueba de esta. Por una parte, se intentará mejorar el rendimiento ofrecido por cada uno de los servicios propios, aunque estarían centradas principalmente en la ejecución y distribución de las simulaciones y de los informes solicitados. Por otro lado, se realizará un despliegue de la aplicación orientado a su uso en un entorno real. El objetivo de esta prueba es comprobar el funcionamiento de la aplicación ante la llegada de múltiples usuarios y la valoración de estos en relación a su uso y rendimiento.

Para realizar esta última prueba mencionada, se propone el uso del DNSE3 por parte de los alumnos en las sesiones prácticas de las asignaturas de “Ingeniería de Teletráfico en Redes Telemáticas” y “Teletráfico. A lo largo de la misma se monitorizará el uso que se hace de la aplicación, al igual que se recogerá la retroalimentación de los usuarios para conocer su opinión sobre el manejo de la aplicación y la ventaja de su uso. Se propone el uso de la aplicación a lo largo del cuatrimestre en el que se imparta dichas asignaturas. Con todas estas pruebas se analizará el uso real de la aplicación y se considerará su posible distribución para su uso libre y gratuito por aquellas entidades que lo soliciten.

Referencias

[AFG+10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andy Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, 2010.

[AP16a] J. I. Asensio-Pérez. Práctica 2 - Teoría de colas y simulación con ns-3 - "ingeniería de teletráfico en redes telemáticas". Technical report, Universidad de Valladolid, 2016.

[AP16b] J. I. Asensio-Pérez. Práctica 3 - Calidad de servicio en redes tcp/ip: un caso de estudio con diffserv - "ingeniería de teletráfico en redes telemáticas". Technical report, Universidad de Valladolid, 2016.

[BLAPGS+12]Miguel L Bote-Lorenzo, Juan I Asensio-Pérez, Eduardo Gómez-Sánchez, Guillermo Vega-Gorgojo, and Carlos Alario-Hoyos. A grid service-based distributed network simulation environment for computer networks education. *Computer Applications in Engineering Education*, 20(4):654–665, 2012.

[BN13] P. Bryan and M. Nottingham. Javascript Object Notation (JSON) PATCH. RFC 6902, RFC Editor, April 2013.

[Bra14] T. Bray. The Javascript Object Notation (JSON) data interchange format. RFC 7159, RFC Editor, March 2014.

[Can12] Rafael Cano. Trabajo Fin de Máster - Entorno de simulación de redes TCP/IP usando servicios rest basado en nube computacional, Universidad de Valladolid, 2012.

[CDK+02] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the web services web: an introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing*, 6(2), 2002.

[CS03]Mark Carson and Darrin Santay. NIST Net: a Linux-based network emulation tool. *ACM SIGCOMM Computer Communication Review*, 33(3):111–126, 2003.

[DS10]L. Dusseault and J. Snell. PATCH method for HTTP. Rfc, RFC Editor, 2010.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, RFC Editor, June 1999.

[FK04]I. Foster and C. Kesselman, editors. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 2004.

[Fos01] Ian T. Foster. The Globus toolkit for grid computing. In *First IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2001), May 15-18, 2001, Brisbane, Australia*, page 2, 2001.

[FR14]R. Fielding and J. Reschke. Hypertext Transfer Orotocol (HTTP/1.1): Conditional requests. RFC 7232, RFC Editor, June 2014.

[Fuj16] Richard M. Fujimoto. Research challenges in parallel and distributed simulation. *ACM Transactions on Modeling and Computer Simulation*, 26, may 2016.

[FV07]Kevin Fall and Kannan Varadhan. The network simulator (ns-2). *URL: http://www. isi. edu/nsnam/ns*, 2007.

[GRKAF10] Mohsen Guizani, Ammar Rayes, Bilal Khan, and Ala Al-Fuqaha. *Network Modelling and Simulation: A practical perspective*. Wiley, 2010.

[Har12] D. Hardt. The OAuth 2.0 authorization framework. RFC 6749, RFC Editor, October 2012.

[JBR00] I. Jacobson, G. Booch, and J. Rumbaugh. *El Proceso Unificado de Desarrollo de Software*. Pearson Education, 2000.

[Ken53] David G. Kendall. Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded markov chain. *The Annals of Mathematical Statistics*, 24(3):338–354, 1953.

[Lar03] Craig Larman. *UML y Patrones*. Pearson Education, 2Âª edition, 2003.

[Law07] Averill M. Law. *Simulation Modelling and Analysis*. McGraw-Hill, 4 edition, 2007.

[Ltd07] OpenSim Ltd. OMNet++: Setting up parallel simulations, 2007.

[MG11] Peter Mell and Tim Grance. The nist definition of cloud computing. 2011.

[OAS] WSRF OASIS. Web services resource framework, oasis standard description.

[Ram10] S. Ramroop. *Performance evaluation of DiffServ networks using the ns-3 simulator*. PhD thesis, University of West Indies, 2010.

[RR07] Leonar Richardson and Sam Ruby. *RESTful Web Services*. O'Reilly, Sebastopol, first edition edition, 2007.

[Seg10] Guillaume Seguin. Multithread parallelization. In *Workshop on ns-3*, may 2010.

[Ser16a] Sergio Serrano. Diseño e implementación del simulador de redes TCP/IP basado en nube computacional dnse3, servicio de colas. *Universidad de Valladolid*, 2016.

[Ser16b] Sergio Serrano. Diseño e implementación del simulador de redes TCP/IP basado en nube computacional dnse3, servicio de estadística. *Universidad de Valladolid*, 2016.

[Ser16c] Sergio Serrano. Diseño e implementación del simulador de redes TCP/IP basado en nube computacional dnse3, servicio de informe. *Universidad de Valladolid*, 2016.

[Ser16d] Sergio Serrano. Diseño e implementación del simulador de redes TCP/IP basado en nube computacional dnse3, servicio de orquestación. *Universidad de Valladolid*, 2016.

[Ser16e] Sergio Serrano. Diseño e implementación del simulador de redes TCP/IP basado en nube computacional dnse3, servicio de simulación. *Universidad de Valladolid*, 2016.

[Vin07] Steve Vinoski. REST eye for the SOA guy. *IEEE Internet Computing*, 11(1), 2007.

[Win95] Mark A. Windschitl. *Using computer simulations to enhance conceptual change: the roles of constructivist instruction and student epistemological beliefs*. Number Paper 15946 in Retrospective Theses and Dissertations. Iowa State Universit, 1995.

[WT12] Kishor Wagh and Ravindra Thool. A comparative study of SOAP vs REST web services provisioning techniques for mobile host. *Journal of Information Engineering and Applications*, 2(5), 2012.

[WvLW09] Elias Weingärtner, Hendrik vom Lehn, and Klaus Wehrle. A performance comparison of recent network simulators. *2009 IEEE International Conference on Communications*, 2009.

[ZA03] Zacharias Zacharia and O. Roger Anderson. The effects of an interactive computer-based simulation prior to performing a laboratory inquiry-based experiment on students' conceptual understanding of physics. *American Journal of Physics*, 71(6):618–629, 2003.

Apéndice A. Fichero de compilación de los modelos utilizados por el cliente de simulación del DNSE3

En este apéndice se muestran el fichero `makefile` genérico empleado en la compilación de los diferentes modelos de simulación para su posterior ejecución.

A.1. makefile

```
CC_SRCS := $(wildcard tmp/*.cc)

CC_DEPS := $(CC_SRCS:.cc=.d)

OBJS := $(CC_SRCS:.cc=.o)

# Librerías de ns-3 a enlazar
LIBS := -lns3.16-csma-debug -lns3.16-lte-debug -lns3.16-wimax-
debug -lns3.16-spectrum-debug -lns3.16-applications-debug -lns3.16-
virtual-net-device-debug -lns3.16-uan-debug -lns3.16-energy-debug -
lns3.16-flow-monitor-debug -lns3.16-nix-vector-routing-debug -lns3.16-
tap-bridge-debug -lns3.16-internet-debug -lns3.16-bridge-debug -
lns3.16-point-to-point-debug -lns3.16-mpi-debug -lns3.16-wifi-debug -
lns3.16-propagation-debug -lns3.16-mobility-debug -lns3.16-config-
store-debug -lns3.16-tools-debug -lns3.16-emu-debug -lns3.16-stats-
debug -lns3.16-topology-read-debug -lns3.16-network-debug -lns3.16-
core-debug -lrt -lm -ldl -lm -ldl -lpthread -lns3.16-aodv-debug -
lns3.16-dsdv-debug -lns3.16-mesh-debug -lns3.16-test-debug -lns3.16-
csma-layout-debug -lns3.16-point-to-point-layout-debug -lns3.16-olsr-
debug


# All Target
all: sim

# Tool invocations
sim: $(OBJS)
  @echo 'Building target: $@'
  @echo 'Invoking: GCC C++ Linker'
  g++ -L/usr/lib -L. -L/home/sergio/ns-allinone-3.16/ns-
3.16/build -o "sim" $(OBJS) $(USER_OBJS) $(LIBS)
  @echo 'Finished building target: $@'
  @echo ' '

# Other Targets
```

FICHERO DE COMPILACIÓN DE LOS MODELOS UTILIZADOS POR EL CLIENTE DE SIMULACIÓN DEL DNSE3

```
	clean:
	  -$(RM)
$(CC_DEPS)$(C++_DEPS)$(EXECUTABLES)$(C_UPPER_DEPS)$(CXX_DEPS)$(OBJS)$(
CPP_DEPS)$(C_DEPS) sim
	  -@echo ' '

	# Puede ser cc o cpp, de momento usaremos .cc
	%.o: %.cc
	  @echo 'Building file: $<'
	  @echo 'Invoking: GCC C++ Compiler'
	  g++ -DNS3_LOG_ENABLE -I/home/sergio/ns-allinone-3.16/ns-
3.16/build -I. -I/home/sergio/ns-allinone-3.16/ns-
3.16/src/internet/model -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -
MF"$(@:%.o=%.d)" -MT"$(@)" -o "$@" "$<"
	  @echo 'Finished building: $<'
	  @echo ' '

	print-%  : ; @echo $* = $($*)

	.PHONY: all clean dependents
```

Apéndice B. Plantillas HOT del servicio de orquestación de la nube computacional para su uso en el DNSE3

En esta sección, se muestran las plantillas empleadas en el servicio de orquestación de la nube computacional para realizar el escalado de las instancias de simulación. Estas plantillas están escritas en YAML y hacen uso de la especificación HOT de la versión `2015-04-30`.

B.1. dnse3.yaml

```
     heat_template_version: 2015-04-30

     description: Heat template for scaling the simulation
service

     parameters:
         flavor:
             type: string
             label: Flavor used by the instances
             description: Flavor used by the instances
             default: m1.small
             #Definición de la configuración hardware de
las instancias
         key:
             type: string
             label: SSH key
             description: SSH key
             #Par de claves SSH para el acceso a las
instancias
         private_network:
             type: string
             label: Network to connect the instances
             description: Network to connect the instances
             default: need-one-network
```

```
            #Red privada a la que se conectarán las
instancia. Necesita de la definición de una red válida por
defecto para su uso

        queue_address:
            type: string
            label: Queue Server IP address
            description: Queue Server IP address
            #Dirección IP del servicio de colas al que
realizar las consultas
        userID:
            type: string
            label: User ID
            description: User ID
            #ID de usuario del servicio de identificación
de la nube
        password:
            type: string
            label: User password
            description: User password
            hidden: true
            #Contraseña de usuario del servicio de
identificación de la nube
            #Tanto el usuario como la contraseña se
emplean para realizar las peticiones a los servicios de la
nube


    resources:
        instance_group:
            type: OS::Heat::AutoScalingGroup
            #Grupo autoescalable de recursos. Su
definición se encuentra en el fichero serverDefinition.yaml
            properties:
                min_size: 1
                max_size: 7
                desired_capacity: 1
                cooldown: 60
                resource:
                    type: serverDefinition.yaml
                    properties:
                        queue_address: {get_param:
queue_address}
                        userID: {get_param: userID}
                        password: {get_param: password}
                        flavor: {get_param: flavor}
                        key: {get_param: key}
```

```
                        private_network: {get_param:
private_network}


        scale_up:
            type: OS::Heat::ScalingPolicy
            #Politica de escalado hacia arriba del grupo
de instancias de simulación
            properties:
                adjustment_type: change_in_capacity
                auto_scaling_group_id: {get_resource:
instance_group}
                scaling_adjustment: 1

        scale_down:
            type: OS::Heat::ScalingPolicy
            #Politica de escalado hacia abajo del grupo de
instancias de simulación
            properties:
                adjustment_type: change_in_capacity
                auto_scaling_group_id: {get_resource:
instance_group}
                scaling_adjustment: '-1'

        queue-alarm-high:
            type: OS::Ceilometer::Alarm
            #Alarma empleada para activar el escalado
hacia arriba del grupo de instancias de simulación. Está
activa si hay alguna simulación pendiente en cada intervalo
de dos segundos
            properties:
                meter_name: dnse3.queue.size
                threshold: 1
                alarm_actions:
                    - { get_attr: [scale_up, alarm_url]}
                comparison_operator: gt
                description: Alarma de Clientes en la cola
                evaluation_periods: 1
                statistic: avg
                period: 2

        queue-alarm-low:
            type: OS::Ceilometer::Alarm
            #Alarma empleada para activar el escalado
hacia abajo del grupo de instancias de simulación. Está
activa si no hay alguna simulación pendiente en cada
intervalo de dos segundos
```

```
            properties:
                meter_name: dnse3.queue.size
                threshold: 1
                alarm_actions:
                    - { get_attr: [scale_down, alarm_url]}
                comparison_operator: le
                description: Alarma de Cola vacía
                evaluation_periods: 1
                statistic: avg
                period: 2

    outputs:
        #Atributos de interés de la pila para
mantenimiento y gestión
        scale_up_url:
            description: Webhook of the Scaling Up Policy
            value: {get_attr: [scale_up, alarm_url]}
        scale_down_url:
            description: Webhook of the Scaling Down
Policy
            value: {get_attr: [scale_down, alarm_url]}
        current_size:
            description: Tamaño del AutoScalingGroup
            value: {get_attr: [instance_group,
current_size]}
```

B.2. serverDefinition.yaml

```
heat_template_version: 2015-04-30

description: Definition of the server configuration

parameters:
  queue_address:
    type: string
    label: Queue Server IP address
    description: Queue Server IP address
    #Dirección del servicio de colas al que realizar
las peticiones
  userID:
    type: string
    label: User ID
    description: User ID
    #ID del usuario del servicio de identificación de
la nube
  password:
    type: string
    label: User password
    description: User password
    hidden: true
    #Contraseña del usuario del servicio de
identificación de la nube
  flavor:
    type: string
    label: Flavor used by the instances
    description: Flavor used by the instances
    #Configuración hardware de la instancia
  key:
    type: string
    label: SSH key
    description: SSH key
    #Par de claves SSH para acceder a la instancia
  private_network:
    type: string
    label: Network to connect the instances
    description: Network to connect the instances
    default: need-one-network
    #Red privada a la que se conectará la instancia.
Necesita de la definición de una red válida por defecto
para su uso

resources:
  server:
```

```
        type: OS::Nova::Server
        properties:
            image: dnse3-0705
            key_name: {get_param: key}
            flavor: {get_param: flavor}
            networks:
              - network: {get_param: private_network}
            user_data:
              #Script ejecutado al iniciarse la instancia.
En futuras iteraciones se incluirá también un script de
apagado de la instancia.
              str_replace:
                template: |
                  #!/bin/bash
                  echo "Running Simulation Client"
                  echo
"wrapper.app.parameter.2=$queueAddress" >>
/home/ubuntu/Simulator/conf/wrapper.conf
                  echo "wrapper.app.parameter.3=$userID"
>> /home/ubuntu/Simulator/conf/wrapper.conf
                  echo
"wrapper.app.parameter.4=$passAccess" >>
/home/ubuntu/Simulator/conf/wrapper.conf
                  /home/ubuntu/Simulator/bin/simulation
start
                  echo "Simulation client booted"
                params:
                  $queueAddress: {get_param:
queue_address}
                  $userID: {get_param: userID}
                  $passAccess: {get_param: password}
            user_data_format: RAW
```

Apéndice C. Modelos de simulación empleados en las pruebas de la aplicación

Se muestran a continuación los *scripts* de simulación empleados para realizar las pruebas de funcionamiento y rendimiento de la implementación desarrollada hasta el momento del DNSE3. Estos *scripts* proceden de las prácticas de laboratorio de la asignatura "Ingeniería de Teletráfico en Redes Telemáticas" impartida por el profesor J.I. Asensio-Pérez en el Grado en Tecnologías de la Telecomunicación de la Universidad de Valladolid.

C.1. first.cc

Modelo de simulación empleado en la práctica 1 de la asignatura "Ingeniería de Teletráfico en Redes Telemáticas" del Grado en Tecnologías de la Telecomunicación de la Universidad de Valladolid.

```
      /* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*-
*/
      /*
       * This program is free software; you can redistribute it and/or
modify
       * it under the terms of the GNU General Public License version
2 as
       * published by the Free Software Foundation;
       *
       * This program is distributed in the hope that it will be
useful,
       * but WITHOUT ANY WARRANTY; without even the implied warranty
of
       * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
       * GNU General Public License for more details.
       *
       * You should have received a copy of the GNU General Public
License
       * along with this program; if not, write to the Free Software
       * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
02111-1307  USA
       */

      #include "ns3/core-module.h"
```

```
      #include "ns3/network-module.h"
      #include "ns3/internet-module.h"
      #include "ns3/point-to-point-module.h"
      #include "ns3/applications-module.h"

      using namespace ns3;

      NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

      int
      main (int argc, char *argv[])
      {
        LogComponentEnable ("UdpEchoClientApplication",
LOG_LEVEL_INFO);
        LogComponentEnable ("UdpEchoServerApplication",
LOG_LEVEL_INFO);

        NodeContainer nodes;
        nodes.Create (2);

        PointToPointHelper pointToPoint;
        pointToPoint.SetDeviceAttribute ("DataRate", StringValue
("5Mbps"));
        pointToPoint.SetChannelAttribute ("Delay", StringValue
("2ms"));

        NetDeviceContainer devices;
        devices = pointToPoint.Install (nodes);

        InternetStackHelper stack;
        stack.Install (nodes);

        Ipv4AddressHelper address;
        address.SetBase ("10.1.1.0", "255.255.255.0");

        Ipv4InterfaceContainer interfaces = address.Assign (devices);

        UdpEchoServerHelper echoServer (9);

        ApplicationContainer serverApps = echoServer.Install
(nodes.Get (1));
        serverApps.Start (Seconds (1.0));
        serverApps.Stop (Seconds (10.0));

        UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
        echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
        echoClient.SetAttribute ("Interval", TimeValue (Seconds
(1.0)));
        echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

        ApplicationContainer clientApps = echoClient.Install
(nodes.Get (0));
        clientApps.Start (Seconds (2.0));
        clientApps.Stop (Seconds (10.0));

        Simulator::Run ();
        Simulator::Destroy ();
```

```
  return 0;
}
```

C.2. queue-mm1-mg1.cc

Modelo de simulación empleado en la primera parte de la práctica 2 de la asignatura “Ingeniería de Teletráfico en Redes Telemáticas” del Grado en Tecnologías de la Telecomunicación de la Universidad de Valladolid.

```
	// Author: Juan I. Asensio (ETSIT Telecomunicación -
	// Universidad de Valladolid - juaase@tel.uva.es)
	// 22/March/2015
	// Based on code by M. Lacage published in
	// E. Altman and T. Jiménez, "NS Simulation for Beginners", Morgan &
Claypool, 2012.
	//
	// Simulation of a M/M/1 or M/G/1 queue

	#include <string>
	#include <fstream>
	#include "ns3/core-module.h"
	#include "ns3/packet.h"
	#include "ns3/ptr.h"
	#include "ns3/random-variable.h"
	#include "ns3/callback.h"
	#include "ns3/simulator.h"
	#include "ns3/nstime.h"
	#include "ns3/command-line.h"
	#include "ns3/point-to-point-module.h"

	#include <list>


	using namespace ns3;
	using namespace std;
	class UVAQueue
	{
		public:
			UVAQueue();
			void SetBitRate(double bitrate);
			void SetAfterProcessed (Callback <void,Ptr <Packet> >
receiver);
			void SetStreamQueue(Ptr<OutputStreamWrapper> stream);
			void SetStreamEndWork(Ptr<OutputStreamWrapper> stream);
			void Enqueue(Ptr<Packet> p);
			double GetLengthIntegrator();
		private:
			void StartWork (int id); //Tiene como parÃ¡metro el
identificador de servidor
			void EndWork (Ptr<Packet> work, int id); //Tiene como
parÃ¡metro el identificador de servidor
			Callback<void,Ptr<Packet> > m_receiver;
			std::list<Ptr<Packet> > m_queue;
			EventId m_working;
			EventId m_working_2; //Modela el segundo servidor
			double m_bitRate; //(bps)
			Ptr<OutputStreamWrapper> m_streamQueue;
			Ptr<OutputStreamWrapper> m_streamEndWork;
			//These variable are for computing Average QueueLength
			long m_clientsCount;
			double m_lengthIntegrator;
			double m_queueLength;
			double m_lastEvent;
	};
```

```
	UVAQueue::UVAQueue()
		: m_clientsCount(0),
		m_lengthIntegrator(0),
		m_queueLength(0),
		m_lastEvent(0)
		{}


	void
	UVAQueue::SetBitRate (double bitRate)
	{
		m_bitRate = bitRate;
	}

	void
	UVAQueue::SetAfterProcessed (Callback <void ,Ptr <Packet> > receiver)
	{
		m_receiver = receiver;
	}

	void
	UVAQueue::SetStreamQueue(Ptr<OutputStreamWrapper> stream) {

		m_streamQueue = stream;
	}

	void
	UVAQueue::SetStreamEndWork(Ptr<OutputStreamWrapper> stream) {

		m_streamEndWork = stream;
	}

	void
	UVAQueue :: Enqueue (Ptr <Packet> p)
	{
		//Updating the "integrator" for calculating average queue size
		//std::cout<< Simulator::Now().GetNanoSeconds() <<
"\t\tm_lastEvent: " << m_lastEvent << "\tqueue: " << m_queue.size() <<
"\tm_lengthIntegrator: " << m_lengthIntegrator << "\n" << std::endl;
		m_lengthIntegrator +=
m_queue.size()*(Simulator::Now().GetNanoSeconds() - m_lastEvent);
		m_lastEvent = Simulator::Now().GetNanoSeconds();

		m_queue.push_back (p);
		//This is a change wrt original code. It is important to print
the
		//queue size AFTER its size has been updated.
		if(m_streamQueue != NULL) {
			*m_streamQueue->GetStream() <<
Simulator::Now().GetNanoSeconds() << "\t\t" << m_queue.size () << std:: endl;
		}
		if (! m_working.IsRunning ()) //If there are no packets being
processed right now... i.e., this packet is at first position in the queue and
the server is "idle"
		{
			StartWork (1);
		}
		else if(! m_working_2.IsRunning ()) //SÃ³lo si estÃ¡ ocupado el
primer servidor se mira si estÃ¡ libre el segundo
		{
			StartWork(2);
		}
```

```
	}

	void
	UVAQueue :: StartWork (int id)
	{
		double serviceDuration;

		//Updating the "integrator" for calculating average queue size
		//std::cout<< Simulator::Now().GetNanoSeconds() <<
"\t\tm_lastEvent: " << m_lastEvent << "\tqueue: " << m_queue.size() <<
"\tm_lengthIntegrator: " << m_lengthIntegrator << "\n" << std::endl;
		m_lengthIntegrator +=
m_queue.size()*(Simulator::Now().GetNanoSeconds() - m_lastEvent);
		m_lastEvent = Simulator::Now().GetNanoSeconds();

		Ptr <Packet> work = m_queue.front ();
		m_queue.pop_front ();
		serviceDuration = (work->GetSize() * 8)/m_bitRate;


		//This is a change wrt original code. It is important to print
the
		//queue size AFTER its size has been updated.
		if(m_streamQueue != NULL) {
			*m_streamQueue->GetStream()  <<
Simulator::Now().GetNanoSeconds() << "\t\t" << m_queue.size () << std:: endl;
		}

		if(id==1) //Se determina quiÃ©n debe de trabajar en funciÃ³n del
id pasado
			m_working = Simulator :: Schedule (Seconds
(serviceDuration),
				&UVAQueue::EndWork, this, work, id);
		else
			m_working_2 = Simulator :: Schedule (Seconds
(serviceDuration),
				&UVAQueue::EndWork, this, work, id);
	}


	void
	UVAQueue :: EndWork (Ptr <Packet> work, int id)
	{
		if(m_streamEndWork != NULL) {
			*m_streamEndWork->GetStream()  <<
Simulator::Now().GetNanoSeconds() <<  std:: endl;
		}
		m_receiver (work);
		if (!m_queue.empty ())
		{
			StartWork (id);
		}
	}

	double
	UVAQueue :: GetLengthIntegrator()
	{
			return m_lengthIntegrator;
	}


	class Sender
	{
```

```
	public:
		Sender();
		void SetCreationInterval (RandomVariable v);
		void SetPacketSize (RandomVariable v);
		void SetFixedPacketSize(int size);
		void SetReceiver (Callback <void,Ptr <Packet> > receiver);
		void SetStream(Ptr<OutputStreamWrapper> stream);
		void Start (void);
		void Stop (void);
	private:
		void DoSend (void);
		RandomVariable m_creationInterval;
		Callback <void ,Ptr <Packet> > m_receiver;
		EventId m_sending;
		RandomVariable m_packetSize;
		int m_fixedPacketSize;
		Ptr<OutputStreamWrapper> m_stream;

};


Sender::Sender()
	: m_fixedPacketSize(0)
	{}

void
Sender::DoSend (void)
{
	Ptr<Packet> p;
	if(m_fixedPacketSize==0){
		p = Create<Packet> (m_packetSize.GetValue());
	} else {
		p = Create<Packet> (m_fixedPacketSize);
	}
	if(m_stream != NULL) {
		*m_stream->GetStream() << Simulator::Now
().GetNanoSeconds() << "\t" << p->GetSize() << std::endl;  //In simulation
Time Now() a packet of p-GetSize() bytes has been generated
	}
	m_receiver (p);
	double sendInterval = m_creationInterval.GetValue ();
	m_sending = Simulator :: Schedule (Seconds (sendInterval),
		&Sender::DoSend,this );
}

void
Sender::SetStream(Ptr<OutputStreamWrapper> stream) {

	m_stream = stream;
}

void
Sender::SetPacketSize (RandomVariable v)
{
	m_packetSize = v;
}

void
Sender::SetFixedPacketSize (int v)
{
	m_fixedPacketSize = v;
}
```

```
	void
	Sender::SetCreationInterval (RandomVariable v)
	{
		m_creationInterval = v;
	}


	void
	Sender::SetReceiver (Callback <void,Ptr <Packet> >  receiver)
	{
		m_receiver = receiver;
	}

	void
	Sender::Start (void)
	{
		DoSend ();
	}

	void
	Sender::Stop(void)
	{
		m_sending.Cancel ();
	}


	//Function to call (Callback) when receiving a packet for servicing

	static void funcAfterProcessed (Ptr <Packet> p)
	{
		//NOT NEEDED FOR THIS EXAMPLE
	}



	int main (int argc, char *argv[])
	{
		bool tracing = false;
		bool fixedlength = false;
		unsigned int rep = 1;
		double lambda = 9.0; //service requests per second
		double tmax = 10000.0;
		double meanPacketSize = 1000.0; //(bytes)
		double bitRate = 100000.0; //(bps)
		double ro;



		AsciiTraceHelper asciiTraceHelper;
		Ptr<OutputStreamWrapper> streamArrivals = NULL;
		Ptr<OutputStreamWrapper> streamClientsInQueue = NULL;
		Ptr<OutputStreamWrapper> streamOutputs = NULL;


		CommandLine cmd;
		cmd.AddValue ("tracing", "Tracing", tracing);
		cmd.AddValue ("rep","Rep",rep);
		cmd.AddValue ("lambda","Lambda",lambda );
		cmd.AddValue ("meanPacketSize","MeanPacketSize
(bytes)",meanPacketSize);
		cmd.AddValue ("bitRate", "BitRate (bps)", bitRate);
		cmd.AddValue ("tmax","Simulation time (seconds)",tmax );
```

```
            cmd.AddValue ("fixedlength","Using fixed or exponential packet
lengths (1=fixed, 0=exponential). Default: exponential", fixedlength);
            cmd.Parse(argc,argv);

            ro = lambda/(bitRate/(meanPacketSize*8));

            streamArrivals = asciiTraceHelper.CreateFileStream ("P3-ns-
QueueingTheory-arrivals.trace");
            streamClientsInQueue = asciiTraceHelper.CreateFileStream("P3-ns-
QueueingTheory-clientsinqueue.trace");
            streamOutputs = asciiTraceHelper.CreateFileStream("P3-ns-
QueueingTheory-outputs.trace");

            SeedManager::SetRun(rep);
            UVAQueue *queue = new UVAQueue ();
            queue->SetBitRate (bitRate);
            queue->SetAfterProcessed(MakeCallback(& funcAfterProcessed ));
            queue->SetStreamQueue(streamClientsInQueue);
            queue->SetStreamEndWork(streamOutputs);

            Sender *sender = new Sender ();
            sender->SetCreationInterval(ExponentialVariable (1.0/ lambda ));
            if(!fixedlength){
                  sender->SetPacketSize (ExponentialVariable
(meanPacketSize));
            }else{
                  sender->SetFixedPacketSize(meanPacketSize);
            }
            sender->SetReceiver(MakeCallback (&UVAQueue::Enqueue, queue));
            sender->SetStream(streamArrivals);

            Simulator::Schedule(Seconds (0.0001), &Sender::Start, sender);
            Simulator::Schedule(Seconds (tmax), &Sender::Stop, sender);
            Simulator::Run();
            Simulator::Destroy();

            if(tracing) {
                  if(!fixedlength){
                        std::cout << "Expected Average Queue Size (M/M/1):
";
                        std::cout << (ro*ro)/(1-ro) << "\n";
                        std::cout << "Obtained Average Queue Size: ";
                  }else{
                        std::cout << "Expected Average Queue Size (M/G/1):
";
                        std::cout <<
((lambda*lambda)*((meanPacketSize*8)/bitRate)*((meanPacketSize*8)/bitRate))/(2
*(1-ro)) << "\n";
                        std::cout << "Obtained Average Queue Size: ";

                  }
            }
            std::cout<< (queue->GetLengthIntegrator()/1000000000/tmax) <<
std::endl;

            delete sender;
            delete queue;
            return 0;
      }
```

C.3. queue-datanetwork.cc

Modelo de simulación empleado en la segunda parte de la práctica 2 de la asignatura "Ingeniería de Teletráfico en Redes Telemáticas" del Grado en Tecnologías de la Telecomunicación de la Universidad de Valladolid.

```
	// Author: Juan I. Asensio (ETSIT Telecomunicación -
	// Universidad de Valladolid - juaase@tel.uva.es)
	// 22/March/2015
	// Parts based on code by M. Lacage published in
	// E. Altman and T. Jiménez, "NS Simulation for Beginners", Morgan &
Claypool, 2012.
	//
	// Simulation of a simple UDP network. Comparison with queueing theory.


	#include "ns3/core-module.h"
	#include "ns3/network-module.h"
	#include "ns3/internet-module.h"
	#include "ns3/point-to-point-module.h"
	#include "ns3/applications-module.h"
	#include <string>
	#include <sstream>

	using namespace ns3;

	NS_LOG_COMPONENT_DEFINE ("QueueDataNetwork");

	////////////////////////////////////////////////////////////////////
	// SENDER CLASS (UDP Application with arbitrary traffic pattern)
	////////////////////////////////////////////////////////////////////

	class Sender
	{
		public:
			Sender();
			void SetCreationInterval (RandomVariable v);
			void SetPacketSize (RandomVariable v);
			void SetFixedPacketSize(int size);
			void SetStream(Ptr<OutputStreamWrapper> stream);
			void SetUdpSocket(Ptr<Socket>);
			void HandleRead (Ptr<Socket> socket);
			long GetPacketCounter();
			void Start (void);
			void Stop (void);
		private:
			void DoSend (void);
			RandomVariable m_creationInterval;
			//Callback <void ,Ptr <Packet> > m_receiver;
			EventId m_sending;
			RandomVariable m_packetSize;
			int m_fixedPacketSize;
			Ptr<Socket> m_socket;
			Ptr<OutputStreamWrapper> m_stream;
			unsigned long m_counter;


	};


	Sender::Sender()
		: m_fixedPacketSize(0)
```

```
	{}

	void
	Sender::DoSend (void)
	{

		Ptr<Packet> p;
		if(m_fixedPacketSize==0){
			p = Create<Packet> (m_packetSize.GetValue());
		} else {
			p = Create<Packet> (m_fixedPacketSize);
		}
		if(m_stream != NULL) {
			*m_stream->GetStream() << Simulator::Now
().GetNanoSeconds() << "\t" << p->GetSize() << std::endl;  //In simulation
Time Now() a packet of p-GetSize() bytes has been generated
		}
		m_socket->Send(p);
		m_counter++;
		double sendInterval = m_creationInterval.GetValue ();
		m_sending = Simulator :: Schedule (Seconds (sendInterval),
			&Sender::DoSend,this );
	}

	long
	Sender::GetPacketCounter() {
		return m_counter;
	}


	void
	Sender::SetUdpSocket(Ptr<Socket> socket) {

		m_socket = socket;
	}


	void
	Sender::SetStream(Ptr<OutputStreamWrapper> stream) {

		m_stream = stream;
	}

	void
	Sender::SetPacketSize (RandomVariable v)
	{
		m_packetSize = v;
	}

	void
	Sender::SetFixedPacketSize (int v)
	{
		m_fixedPacketSize = v;
	}

	void
	Sender::SetCreationInterval (RandomVariable v)
	{
		m_creationInterval = v;
	}


	void
```

```
	Sender::Start (void)
	{
		m_counter=0;
		DoSend ();
	}

	void
	Sender::Stop(void)
	{
		m_sending.Cancel ();
	}

	void
	Sender::HandleRead (Ptr<Socket> socket)
	{
	  NS_LOG_FUNCTION (this << socket);
	  Ptr<Packet> packet;
	  Address from;
	  while (packet = socket->RecvFrom (from))
	    {
	      if (InetSocketAddress::IsMatchingType (from))
	        {
	          InetSocketAddress address = InetSocketAddress::ConvertFrom
(from);
	          NS_LOG_INFO ("Received " << packet->GetSize() << " bytes from
" <<
	            address.GetIpv4());
	        }
	    }
	}

	//////////////////////////////////////////////////////////////////
	//Global Variables for calculating Average Buffer use
	Ptr<DropTailQueue> senderqueue;
	double lengthIntegrator=0;
	double lastEvent=0;



	//////////////////////////////////////////////////////////////////
	//Queue Tracing callbacks
	static void
	senderDrop (Ptr<OutputStreamWrapper> stream, Ptr<const Packet> p)
	{
		NS_LOG_UNCOND("sender QueueDrop: " << Simulator::Now
().GetNanoSeconds ());
		*stream->GetStream () << Simulator::Now ().GetNanoSeconds () <<
std::endl;
	}

	static void SinkRx (Ptr<OutputStreamWrapper> stream, Ptr<const Packet>
p, const Address &ad)
	{
		*stream->GetStream() << Simulator::Now().GetNanoSeconds() <<
std::endl;
	}

	static void
	senderEnqueue (Ptr<OutputStreamWrapper> stream, Ptr<const Packet> p)
	{
		//IMPORTANT: the Queue class invokes this callback BEFORE the
size of the Queue is updated
		//see queue.cc (Enqueue method)
```

```
            *stream->GetStream () << Simulator::Now().GetNanoSeconds() <<
"\t\t" << senderqueue->GetNPackets() +1 << std:: endl;
            lengthIntegrator += (senderqueue->GetNPackets()
)*(Simulator::Now().GetNanoSeconds() - lastEvent);
            lastEvent = Simulator::Now().GetNanoSeconds();
      }

      static void
      senderDequeue (Ptr<OutputStreamWrapper> stream, Ptr<const Packet> p)
      {
            //IMPORTANT: the Queue class invokes this callback AFTER the size
of the Queue is updated
            //see queue.cc (Enqueue method)
            *stream->GetStream () << Simulator::Now().GetNanoSeconds() <<
"\t\t" << senderqueue->GetNPackets() << std:: endl;
            lengthIntegrator += (senderqueue->GetNPackets()
+1)*(Simulator::Now().GetNanoSeconds() - lastEvent);
            lastEvent = Simulator::Now().GetNanoSeconds();
      }



      ////////////////////////////////////////////////////////////////
      // MAIN FUNCTION
      ////////////////////////////////////////////////////////////////////

      int
      main (int argc, char *argv[])
      {


            bool tracing = false;
            bool fixedlength = false;
            unsigned int rep = 1;
            double lambda = 9.0; //service requests per second
            double tmax = 2000.0; //simulation time length (default value
might not be suitable)
            double meanPacketSize = 1000.0; //(bytes)
            double bitRate = 100000; //bps
            uint32_t senderIPBufferSize = 300; // In segments
            std::string linkDelay = "300ns";


            CommandLine cmd;
            cmd.AddValue ("tracing", "Tracing. Default: 0", tracing);
            cmd.AddValue ("rep","Rep",rep);
            cmd.AddValue ("lambda","Lambda. Default: 9 pkts/sec.",lambda );
            cmd.AddValue ("meanPacketSize","MeanPacketSize (bytes). Default:
1000 bytes.",meanPacketSize);
            cmd.AddValue ("bitRate", "Link BitRate (bps). Default: 100000
bps", bitRate);
            cmd.AddValue ("tmax","Simulation time (seconds). Default: 10000
sec.",tmax );
            cmd.AddValue ("fixedlength","Using fixed or exponential packet
lengths (1=fixed, 0=exponential). Default: exponential", fixedlength);
            cmd.AddValue ("senderIPBufferSize", "Number of IP packets the
sender IP entity should be able to store in its buffer for forwarding.
Default: 300 packets)", senderIPBufferSize);
            cmd.AddValue ("linkDelay", "Delay of the sender->r1 and r2->s2
p2p links. Default: 300ns.", linkDelay);

            cmd.Parse(argc,argv);
```

```
              //Setting up trace files...
              AsciiTraceHelper asciiTraceHelper;
              Ptr<OutputStreamWrapper> streamArrivals = NULL;
              streamArrivals = asciiTraceHelper.CreateFileStream ("P3-ns-
QueueingTheory-arrivals.trace");



              SeedManager::SetRun(rep);


              //Setting up topology and communications architecture
              NodeContainer nodes;
              nodes.Create (2);

              PointToPointHelper pointToPoint;


              std::string bitRate_String;
              std::stringstream bitRate_Stream;
              bitRate_Stream << bitRate/1000 << "Kbps\n";
              pointToPoint.SetDeviceAttribute ("DataRate",
StringValue(bitRate_Stream.str()));



              pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

              pointToPoint.EnablePcapAll ("tcp-bulk-send", false);

              NetDeviceContainer devices;
              devices = pointToPoint.Install (nodes);



              // Setting up queue at sender's device
              senderqueue = CreateObject<DropTailQueue>();
              senderqueue-
>SetAttribute("MaxPackets",UintegerValue(senderIPBufferSize));
              Ptr<PointToPointNetDevice> devsenderp2p =
DynamicCast<PointToPointNetDevice> (devices.Get(0));
              devsenderp2p->SetQueue(senderqueue);

              //Setting callback for queue tracing
              Ptr<OutputStreamWrapper> senderstreamDrop =
asciiTraceHelper.CreateFileStream ("sender.queuedropped.trace");
              senderqueue->TraceConnectWithoutContext ("Drop",
MakeBoundCallback(&senderDrop, senderstreamDrop));
              Ptr<OutputStreamWrapper> senderstreamEnqueue =
asciiTraceHelper.CreateFileStream ("sender.enqueue.trace");
              senderqueue->TraceConnectWithoutContext ("Enqueue",
MakeBoundCallback(&senderEnqueue, senderstreamEnqueue));
              Ptr<OutputStreamWrapper> senderstreamDequeue =
asciiTraceHelper.CreateFileStream ("sender.dequeue.trace");
              senderqueue->TraceConnectWithoutContext ("Dequeue",
MakeBoundCallback(&senderDequeue, senderstreamDequeue));


              //Addressing, setting up the UDP server
              InternetStackHelper stack;
```

```
            stack.Install (nodes);

            Ipv4AddressHelper address;
            address.SetBase ("10.1.1.0", "255.255.255.0");

            Ipv4InterfaceContainer interfaces = address.Assign (devices);

            PacketSinkHelper receiverHelper ("ns3::UdpSocketFactory",
InetSocketAddress(interfaces.GetAddress(1), 9));
            ApplicationContainer serverApps =
receiverHelper.Install(nodes.Get(1));


            Ptr<OutputStreamWrapper> receptorTrace =
asciiTraceHelper.CreateFileStream ("receiverUDP.trace");
             Config::ConnectWithoutContext
("/NodeList/1/ApplicationList/0/$ns3::PacketSink/Rx",MakeBoundCallback
(&SinkRx, receptorTrace));

            //Traffic generation
            Sender *sender = new Sender ();
            sender->SetCreationInterval(ExponentialVariable (1.0/ lambda ));
            if(!fixedlength){
                  sender->SetPacketSize (ExponentialVariable
(meanPacketSize));
            }else{
                  sender->SetFixedPacketSize(meanPacketSize);
            }
            sender->SetStream(streamArrivals);


            //Socket creation (obtained from udp-echo-client.cc
            TypeId tid = TypeId::LookupByName("ns3::UdpSocketFactory");
            Ptr<Socket> udp_socket;
            udp_socket = Socket::CreateSocket(nodes.Get(0),tid);
            udp_socket->Bind();
            udp_socket->Connect(InetSocketAddress(interfaces.GetAddress(1),
9));
            udp_socket->SetRecvCallback(MakeCallback(&Sender::HandleRead,
sender));
            sender->SetUdpSocket(udp_socket);


            //Simulation Management
            Simulator::Schedule(Seconds (0.0001), &Sender::Start, sender);
            Simulator::Schedule(Seconds (tmax), &Sender::Stop, sender);
            serverApps.Start (Seconds (0.0001));
            serverApps.Stop (Seconds (tmax));
            Simulator::Run();
            Simulator::Destroy();

            //Tracing
            if(tracing) {
                  std::cout << sender->GetPacketCounter() << " UDP Datagrams
Sent\n";
                  std::cout << senderqueue->GetTotalReceivedPackets() << "
Packets received by the sender queue\n";
                  std::cout << senderqueue->GetTotalDroppedPackets() << "
Packets dropped from the sender queue\n";
            }
```

```
            std::cout<< (lengthIntegrator/1000000000/tmax) << std::endl;

            //Freeing resources
            delete sender;

            return 0;
      }
```

C.4. P3_trafficanalysis.m

Script de GNU Octave empleado para el visionado de los resultados obtenidos tras la ejecución del modelo de simulación empleado en la práctica 3 de la asignatura "Ingeniería de Teletráfico en Redes Telemáticas" del Grado en Tecnologías de la Telecomunicación de la Universidad de Valladolid.

```
      % ITRT, P3
      % Visualization of generated traffic (s1), received traffic (s2)
and dropped
      % traffic at r1 as regulated by the policing strategy employed
(e.g. tocket bucket)
      % NOTE: only dropped packets are considered (i.e. if the policy
is marking the
      % packets with a different traffic class, these packets are not
displayed)
      %
      % Author: Juan I. Asensio-Perez
      % ETSIT Telecomununicacion, Universidad de Valladolid, Spain
      % 29-april-2014


      clf;


      %How many plots will be displayed?
      global plotx;
      global ploty;


      %Function that generates a bitrate/time plot from a trace of
(time, packet_size)
      %     resolution: is the time columns in the trace file in
seconds, miliseconds, ...? (e.g.: resolution=1e9 => nanoseconds)
      %     step: size of the time slot in which the traffic is
aggregated
      function plottraffic(data, plotposition, mytitle, mycolor,
resolution, step)

            global plotx;
            global ploty;

            l=length(data);
            tmax=data(l,1);
            steps=(tmax/resolution)/step;

            traffic=0;
            traffictime=0;
            index=1;
            indexprev=1;
            for i=1:steps
                index=lookup(data(:,1),i*step*resolution);
                  if(index==0)
                        dropstraffic(i)=0;
                        indexprev=1;
                  else
```

```
                aggr = sum(data(indexprev:index,2));
                dropstraffic(i)=(aggr*8)/step/1000; %Kbps
                      indexprev=index;
                endif
              dropstime(i)=i*step;
            end

            bytessent = sum(data(:,2));
            averagetraffic = ((bytessent*8)/(tmax/resolution))/1000;
%Kbits per second
            subplot(plotx,ploty,plotposition);
            plot(dropstime,dropstraffic,"color",mycolor,"marker",'*');
            title(sprintf("%s\n Average bitrate: %f Kbps", mytitle,
averagetraffic));
            xlabel("Time (sec.)");
            ylabel("Kbps");


      endfunction


      plotx=1;
      ploty=3;
      step=10; %sec (integration interval)
      resolution=1e9; %times in traces are in nanoseconds...



      %%GENERATED TRAFFIC AT s1
      s1data=load('37080_P3_DiffServ_s1_arrivals.trace');
      plottraffic(s1data, 1, "TRAFFIC GENERATED AT s1", "black",
resolution, step);

      %RECEIVED TRAFFIC AT s2
      s2data=load('37080_P3_DiffServ_s2_arrivals.trace');
      plottraffic(s2data, 2, "TRAFFIC RECEIVED AT s2", "black",
resolution, step);

      %%DROPPED NON-CONFORMANT PACKETS AT THE POLICING POINT (before
enqueueing)
      if((r1DropNonConformantAFBE=load('37080_r1.DropNonConformantAFBE
.trace')))
            plottraffic(r1DropNonConformantAFBE, 3, "TRAFFIC DROPPED
(before enqueueing) AT r1", "red", resolution, step);
      endif;
```