# University of Castilla-La Mancha

A publication of the

**Department of Computer Science**

## Modelado, Transformación y Validación de Sistemas Cloud

by

Adrián Bernal  María Emilia Cambronero  Valentín Valero

Alberto Núñez

Technical Report **#DIAB-18-01-1** January 2018

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS
ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE CASTILLA-LA MANCHA
Campus Universitario s/n
Albacete - 02071 - Spain
Phone +34.967.599200, Fax +34.967.599224

**Resumen:**

*Durante la formación en tecnologías de la información y de las comunicaciones suele existir un acercamiento a los sistemas de computación distribuida. Éste es, en la mayoría de las ocasiones, un acercamiento teórico, debido a los problemas técnicos y económicos que surgen a la hora de su implantación. Se echa de menos poner en práctica algunos aspectos y observar resultados acerca de lo estudiado.*

*Existen algunas herramientas, como Simcan2Cloud, que permiten la simulación de entornos distribuidos. Estas herramientas necesitan ser configuradas, transcribiendo las características de los sistemas modelados a la configuración específica de la herramienta.*

*Por lo tanto, la intención principal de este proyecto es facilitar el acceso a los sistemas distribuidos en el ámbito académico. Para ello, se ha definido un perfil de UML, que nos permite modelar un sistema cloud, y se ha desarrollo un framework para generar automáticamente, a partir de dicho modelo, los ficheros de configuración del simulador Simcan2Cloud. Esto nos va a permitir analizar los modelos a la vez que observamos resultados.*

*El framework se ha desplegado como un plugin para la plataforma Eclipse, en espacial para la aplicación Papyrus Neon 2.0.2, que consta de la definición del perfil junto las restricciones necesarias para su validación, una personalización del editor gráfico para facilitar el modelado, un modelo de ejemplo que se puede usar como sistema base y la transformación que finalmente generará los ficheros necesarios. Dicho plugin es accesible desde la dirección http://dsi.uclm.es/cloud/simcan/uml2cloud.*

## Introducción:

Los sistemas distribuidos tienen un papel fundamental en el sector de las tecnologías de la información y de las comunicaciones (TIC). En los últimos años, debido a la rápida evolución de las redes de comunicaciones, han sufrido un importante incremento en su uso hasta pasar a formar parte de nuestro entorno cotidiano, por ejemplo, a través del uso de redes sociales o del almacenamiento en la nube. Como consecuencia de ello, la importancia de estos sistemas crece en los diferentes ámbitos, tanto en el académico y de la investigación como en el empresarial.

En el ámbito académico, uno de los mayores problemas de la enseñanza de sistemas distribuidos es la dificultad de acceso, por parte de los alumnos, a sistemas distribuidos reales para realizar las prácticas. Por un lado, las universidades no suelen tener recursos suficientes para adquirir y administrar la infraestructura necesaria. Por otro lado, la configuración necesaria para desarrollar las prácticas de sistemas distribuidos entra, generalmente, en conflicto con la política de seguridad informática de la universidad. Con el fin de impedir ataques a los ordenadores de los laboratorios informáticos, éstos suelen utilizar cortafuegos que impiden la comunicación entre varios ordenadores.

Para mitigar las dificultades existentes para el uso y ejecución de sistemas en la nube en los distintos ámbitos se propone utilizar herramientas de simulación que alivien, en la medida de lo posible, la situación actual de los sistemas distribuidos.

SIMCAN [1] es una plataforma de simulación de sistemas de computación en la nube de código abierto, enfocada a la simulación de sistemas distribuidos y desarrollada en la Universidad Complutense de Madrid. Esta plataforma nos permite comprobar la funcionalidad y el rendimiento de un modelo dado antes de su implantación.

El problema principal que encontramos a la hora de usarla es la ausencia de un lenguaje estandarizado para el modelado de los diferentes elementos de estos sistemas, así como de la comunicación entre sus componentes, lo que obliga a

aprender el lenguaje específico de la plataforma para su uso, disuadiendo de esta manera a muchos usuarios potenciales.

Una solución a este problema es el diseño de la infraestructura cloud mediante modelos independientes de la tecnología usada para la implementación. La Arquitectura Dirigida por Modelos (Model-Driven Architecture, MDA) [2] es un enfoque que nos proporciona un conjunto de estándares para la elaboración de dichos modelos y para transformarlos de manera automática.

Para facilitar esta tarea, se desarrollará un framework que nos permitirá modelar la infraestructura de la plataforma cloud, así como la interacción de los usuarios con ella, mediante lenguajes de modelado estandarizados. A partir de este diseño, el framework nos proporcionará automáticamente los ficheros necesarios para configurar la plataforma, sin necesidad de escribir código ni conocer su lenguaje, facilitando de esta forma su uso por usuarios no experimentados, ya que se parte de un modelo gráfico del mismo, que es más intuitivo y fácilmente entendible. A continuación, podremos realizar la simulación y obtener los resultados, tal como se muestra en el esquema de la Figura 1.

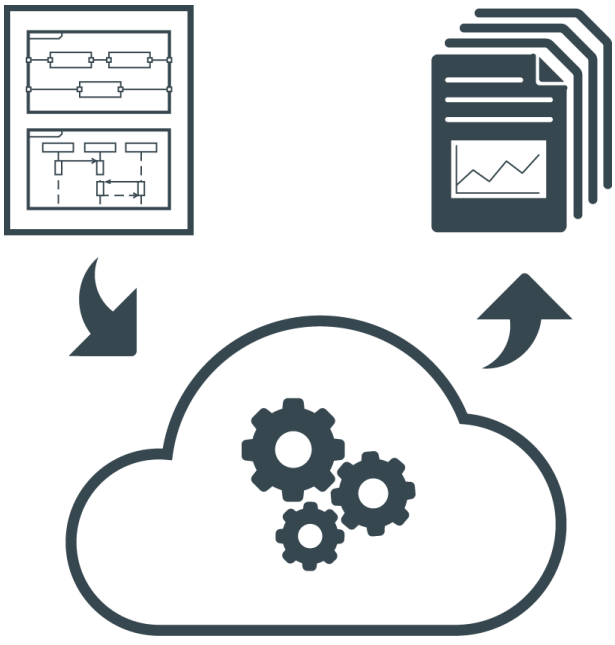

**Figura 1. Esquema de funcionamiento de SIMCAN y UML2SIMCAN**

Actualmente no existe disponible ninguna herramienta que nos permita modelar gráficamente un sistema cloud para el simulador SIMCAN. La forma actual para configurarlo es escribiendo los ficheros necesarios directamente.

Si bien no existen herramientas de modelado específicas para este simulador, los frameworks siguientes permiten modelar sistemas en UML para su posterior transformación a OMNeT++:

- **Syntony** [3] es un framework basado en Eclipse para el desarrollo y análisis automatizado de protocolos de red. Permite el modelado de los protocolos mediante diagramas de estructura compuesta, de máquina de estados y de actividad. Para ello utiliza UML[4], así como los perfiles MARTE y UTP. Desgraciadamente, la página web del proyecto ya no se encuentra disponible, por lo que no es posible conseguir la herramienta.
- **VeriTAS** [5] es un entorno de modelado que extiende Syntony, añadiendo la posibilidad de ejecutar casos de prueba para componentes individuales del sistema modelado. Actualmente tampoco hay acceso a esta herramienta.

Como vemos, otros trabajos han aplicado MDA para el modelado y simulación de otros tipos de sistemas, pero no para el modelado y simulación de sistemas cloud. Con la realización de este trabajo comprobamos la viabilidad de la aplicación de MDA para el modelado y simulación de infraestructuras cloud y la interacción de los usuarios con dichas infraestructuras.

A la hora de modelar el sistema cloud existen algunos trabajos, como el de Bergmayr y otros [6] y el de Kamali y otros [7] que proponen el uso de perfiles UML para el modelado de Cloud Computing, pero se centran más en el modelado de aplicaciones y su despliegue que de la infraestructura e interacción de los usuarios, por lo que no modelan elementos necesarios para SIMCAN.

## Materiales y Métodos:

# SIMCAN

SIMCAN [1] es una plataforma de simulación orientada a analizar y estudiar aplicaciones paralelas en sistemas distribuidos. Ha sido diseñada para proporcionar flexibilidad, precisión, rendimiento y escalabilidad. Estas características la convierten en una potente plataforma de simulación para diseñar, probar y analizar tanto arquitecturas reales como inexistentes. Ha sido desarrollada utilizando los frameworks de simulación INET[1] y OMNeT++[2]. La gama de sistemas a simular puede variar desde un único nodo de computación hasta un sistema distribuido de alto rendimiento completo. Para lograrlo, sigue un diseño modular que permite simular un amplio rango de arquitecturas con sólo unos pocos módulos implementados y una configuración bien definida.

Para mantener el equilibrio entre rendimiento, precisión, flexibilidad y escalabilidad, el núcleo principal de esta plataforma sigue un diseño flexible dirigido a la integración de cada sistema básico en un solo modelo de simulación. Los sistemas básicos son el sistema de almacenamiento, el sistema de memoria, el sistema de procesamiento y el sistema de red.

Las arquitecturas simuladas se modelan utilizando un conjunto de componentes existentes proporcionados por SIMCAN, que representan el comportamiento de componentes del mundo real, como discos, redes, memorias y sistemas de archivos. Estos componentes se organizan jerárquicamente dentro del repositorio de SIMCAN. Actualmente, la plataforma ofrece una amplia gama de componentes para modelar

[1] INET Framework es una biblioteca de modelos de código abierto para el entorno de simalación OMNeT++. Provee protocolos, agentes y otros modelos para trabajar con redes de comunicaciones. https://inet.omnetpp.org/

[2] OMNeT++ es una biblioteca de simulación C++ extensible, modular y basada en componentes, principalmente para construir simuladores de redes. https://omnetpp.org/

sistemas distribuidos completos con diferentes niveles de detalle y escalabilidad. Además de diseñar entornos simulados utilizando componentes proporcionados por SIMCAN, se pueden añadir nuevos componentes a su repositorio.

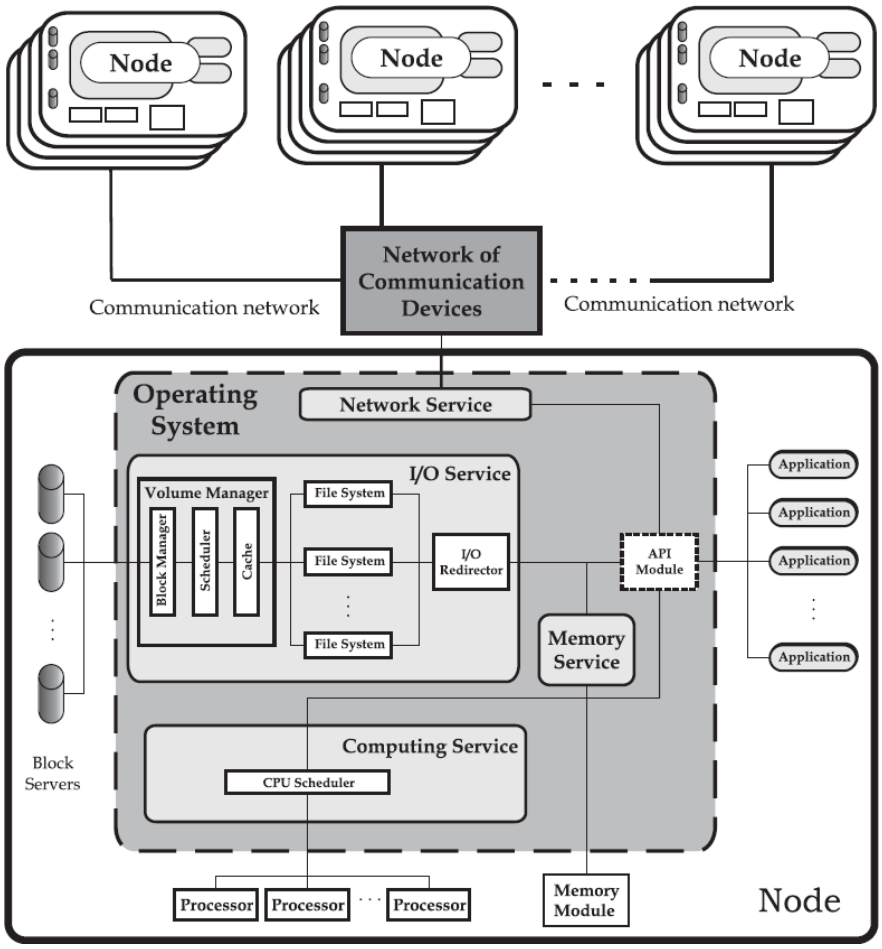


**Figura 2. Arquitectura global de SIMCAN**

En un sistema informático, el nodo es el componente más relevante. Del mismo modo, en SIMCAN un nodo es un bloque de construcción utilizado para crear entornos distribuidos. La simulación de un solo nodo se basa en la unión de los módulos que simulan las aplicaciones con los componentes de simulación de los cuatro sistemas básicos, como se puede ver en la Figura 2.

La plataforma SIMCAN incluye cuatro niveles jerárquicos: módulos simples que realizan tareas específicas, sistemas básicos que simulan cualquiera de los componentes principales de los sistemas distribuidos (almacenamiento, computación,

memoria y red), componentes principales que simulan las unidades físicas de las arquitecturas distribuidas (nodos), y agregaciones de componentes (racks).

En SIMCAN, la definición y configuración de un entorno modelado se divide en tres niveles diferentes, con su nivel de abstracción correspondiente. Así, cuanto mayor sea el nivel de configuración menor será el nivel de abstracción. Los niveles de configuración son:

- **Nivel 1:** Consiste en la definición del sistema correspondiente al entorno cloud modelado. Este nivel contiene básicamente una lista de los componentes con sus correspondientes conexiones. Esta descripción se especifica en un fichero de texto plano, con extensión *.ned*, usando el lenguaje NED, que facilita la definición modular de un sistema distribuido.
- **Nivel 2:** Consiste en la configuración de cada componente definido en el nivel 1. Esta configuración consiste básicamente en la asignación de valores a los parámetros de cada componente, para personalizar el comportamiento del entorno simulado. Los parámetros pueden ser cadenas de texto, numéricos, valores booleanos o un árbol de datos XML. Como en el nivel 1, esta configuración se especifica mediante un fichero de texto plano, con extensión *.ini*.
- **Nivel 3:** Consiste en ficheros de configuración adicionales para personalizar componentes específicos que lo requieran. Este nivel se usa solamente para modelar y simular entornos complejos que requieren un alto nivel de detalle y personalización.

Actualmente, la segunda versión de esta plataforma, *Simcan2Cloud*, está siendo desarrollado por un grupo de investigación de la Universidad Complutense de Madrid. La primera versión permitía la simulación de un clúster y obtener información acerca de su rendimiento y consumo, pero sin gestión de usuarios. La segunda versión deja atrás el control del consumo para centrarse en la simulación de

un sistema cloud, incluyendo la gestión de usuarios y obteniendo información sobre su rendimiento. La primera versión del framework, que ha sido desarrollado durante la realización de este trabajo, nos permite configurar los niveles 1 y 2, de la plataforma *Simcan2Cloud*, automáticamente a partir de un modelo dado.

## Ingeniería Dirigida por Modelos

En el desarrollo software, un enfoque prometedor para abordar la complejidad de la plataforma, y la incapacidad de los lenguajes de tercera generación para aliviar esta complejidad y expresar los conceptos de dominio de manera efectiva, es desarrollar tecnologías de Ingeniería Dirigida por Modelos (Model-Driven Engineering, MDE) [8] que combinen Lenguajes de Modelado de Dominio Específico (Domain-Specific Modeling Languages, DSMLs) [8], así como motores y generadores de transformaciones de forma automática o semiautomática. Esos DSMLs pueden adaptarse a través del meta-modelado para coincidir con la sintaxis y la semántica del dominio. Tener elementos gráficos que se relacionen directamente con un dominio no sólo ayuda a aplanar la curva de aprendizaje, sino que también ayuda a una gama más amplia de expertos en la materia a asegurar que los sistemas software satisfacen las necesidades de los usuarios. Además, las herramientas MDE imponen restricciones específicas del dominio y permiten validar el modelo, lo que ayuda a detectar y prevenir errores al principio del ciclo de vida.

Para crear un modelo de nuestro sistema, necesitamos un lenguaje de modelado y, para definir este lenguaje, un lenguaje de metamodelado.

Los lenguajes de modelado se pueden clasificar, dependiendo de su alcance en Lenguajes de Modelado de Dominio Específico (Domain Specific Modeling Languages, DSMLs) [8] o en Lenguages de Modelado de Propósito General (General Purpose Modeling Languages, GPMLs). Éste último, como es UML, está pensado para describir modelos en cualquier dominio.

## UML

UML se ha convertido en el lenguaje universal del desarrollo software, permitiendo a los ingenieros intercambiar sus diseños libremente. La capacidad de entender los modelos creados por otros ingenieros ha permitido la colaboración de distintas organizaciones, ayudando a la creación de proyectos software de mayor envergadura y de mayor calidad.

UML permite a los profesionales modelar [9] de una manera completa y correcta la estructura y el comportamiento de un sistema, evidenciando sus características más destacadas. Durante la construcción de los modelos, los arquitectos y diseñadores pueden analizar y ajustar el sistema en desarrollo, ya que los diagramas que construyen les ayudan a visualizar la estructura de alto nivel, las relaciones entre los componentes, y detalles de nivel inferior. Una vez completados los modelos, pueden ser presentados a los usuarios para su aprobación, y ser implementados y ejecutados. Algunas herramientas de modelado pueden ejecutar el modelo directamente, o generar el código y los ficheros auxiliares necesarios para desplegarlo y ejecutarlo.

La especificación de UML define dos tipos principales de diagramas, los diagramas de estructura y los diagramas de comportamiento, y que se pueden clasificar jerárquicamente como se muestra en la Figura 3 [10]. Los diagramas de estructura muestran la estructura estática del sistema y sus partes en diferentes niveles de abstracción e implementación y cómo están relacionados entre sí. Los diagramas de comportamiento muestran el comportamiento dinámico de los objetos en un sistema, que puede describirse como una serie de cambios en el sistema a lo largo del tiempo.

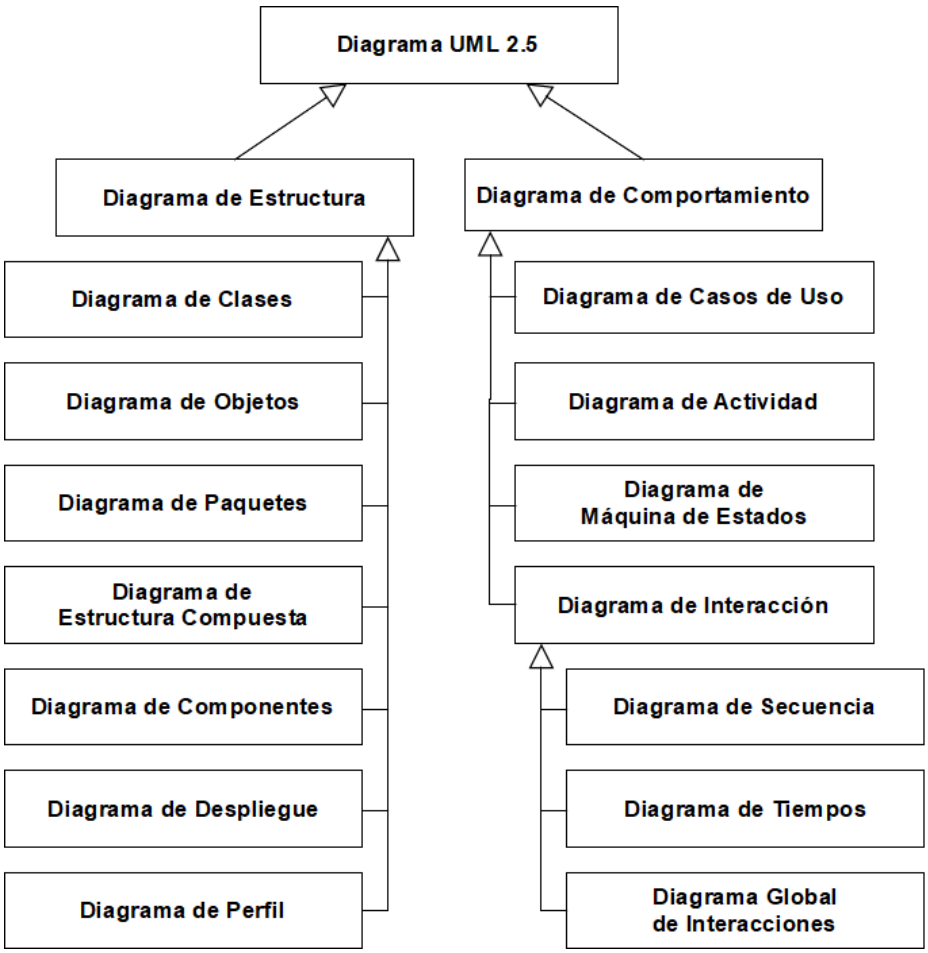


**Figura 3. Clasificación jerárquica de los diagramas de UML**

A continuación, se hará una breve descripción de los diagramas [10] que serán útiles en la realización de este trabajo:

- **Diagrama de componentes:** Muestra los componentes, las interfaces proporcionadas y necesarias, los puertos y las relaciones entre ellos. El desarrollo basado en componentes se basa en la suposición de que los componentes podrían ser reutilizados y que podrían ser reemplazados por otros componentes equivalentes si fuese necesario. Los componentes pueden ser físicos o lógicos.
- **Diagrama de perfil:** Diagrama UML auxiliar que permite definir estereotipos, valores etiquetados y restricciones como mecanismo de extensión para el estándar UML.

- **Diagrama de secuencia:** Es un diagrama de interacción que se centra en la secuencia de mensajes que se intercambian entre objetos. Un elemento importante de estos diagramas, además de las líneas de vida y los mensajes, son los fragmentos combinados, los cuales nos van a permitir definir el flujo de ejecución a través de las guardas que indicarán si se ejecuta o no el flujo especificado dentro de los fragmentos.

## Perfiles UML

UML es una notación de propósito muy general [11], lo que proporciona una gran flexibilidad y expresividad a la hora de modelar sistemas. Sin embargo, hay muchas ocasiones en las que se hace necesario algún lenguaje más específico para modelar y representar los conceptos de ciertos dominios particulares. Esto sucede cuando la sintaxis o la semántica de UML no permite expresar los conceptos específicos del dominio, o cuando se desea restringir y especializar los constructores propios de UML, que normalmente son demasiado genéricos y numerosos.

El OMG nos ofrece dos posibilidades a la hora de definir lenguajes específicos del dominio. Una de ellas es la definición de un nuevo lenguaje, como alternativa a UML, mediante la descripción de su metamodelo utilizando el MOF [12]. Esto nos va a permitir tener un mayor grado de expresividad y correspondencia con los conceptos del dominio de aplicación en particular. Sin embargo, y aun cuando esos nuevos lenguajes se describan con MOF, el hecho de no respetar el metamodelo estándar de UML va a impedir que las herramientas UML existentes puedan manejar sus conceptos de una forma natural. La otra posibilidad es extender el propio UML utilizando lo que se denominan Perfiles UML (UML Profiles). Los Perfiles UML constituyen el mecanismo que proporciona el propio UML para extender su sintaxis y su semántica para expresar los conceptos específicos de un determinado dominio de aplicación. La idea es utilizar los estereotipos de un perfil para marcar los elementos de un Modelo Independiente de la Plataforma (Platform Independent Model, PIM) y

así producir el Modelo Específico de la Plataforma (Platform Specific Model, PSM) correspondiente [13]. Una marca representa un concepto en el PSM, y se aplica a un elemento del PIM para indicar cómo debe ser transformado en el PSM objetivo.

## Papyrus: Herramienta de modelado y metamodelado

Papyrus Neon 2.0.2 [14] es un editor gráfico para UML2 de código abierto y basado en Eclipse. Esta herramienta nos va a permitir especificar el perfil y modelar de manera gráfica el sistema aplicando dicho perfil y aplicar restricciones sobre él mediante el lenguaje OCL (Object Constraint Language). Utilizaremos esta herramienta en este trabajo debido a que es de código abierto, a las posibilidades de personalización que nos ofrece y a que nos va a permitir reutilizar y extender los diagramas de UML.

## Perfil SIMCAN

Para permitir el modelado tanto de la infraestructura del sistema cloud como la interacción de los usuarios con él, y así indicar que elementos de UML corresponden con cada componente del simulador, se ha definido un perfil de UML.

### Infraestructura

Para modelar la infraestructura del sistema cloud hay que definir los centros de datos (*Data Centers*), dentro de los cuales habrá estantes (*Racks*) con nodos (*Nodes*) compuestos por un microprocesador (*CPU*), un disco duro (*Disk*) y memoria RAM (*Memory*), por lo que se han definido los estereotipos para modelar dichos componentes del simulador, así como para organizarlos (*Scenario, Repository*) (ver Figura 4).

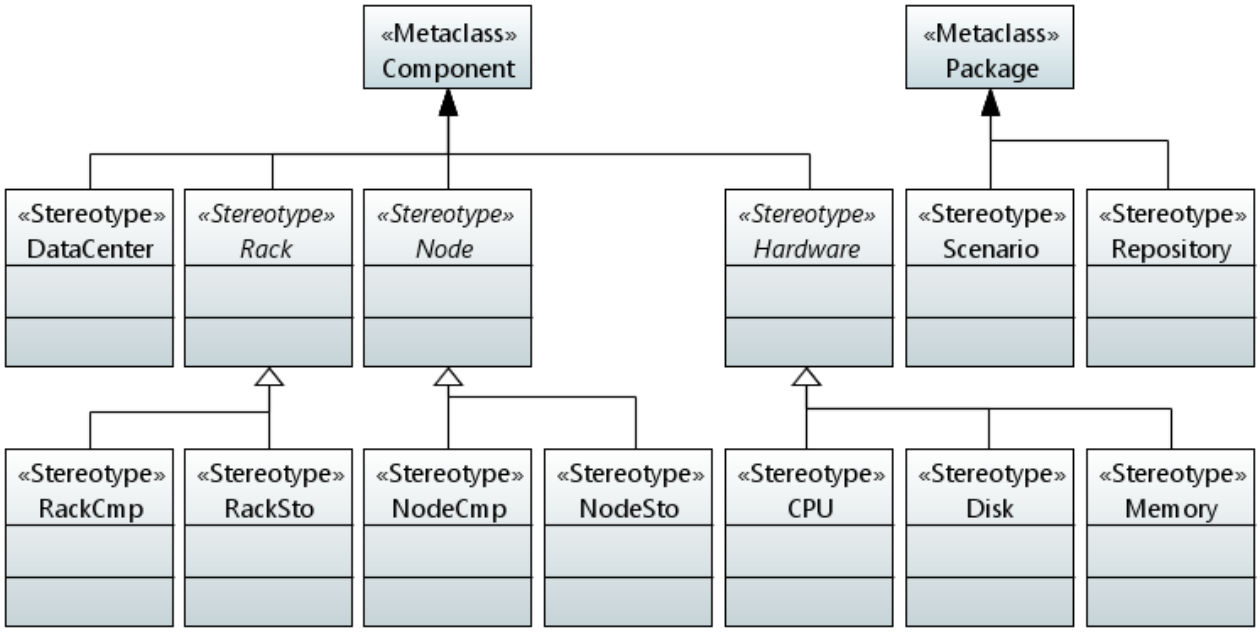


**Figura 4. Perfil SIMCAN - Definición de estereotipos de los componentes**

El estereotipo *Repository* no es ningún componente del simulador, pero se ha definido para organizar mejor los componentes modelados en repositorios dependiendo de su tipo. Los estereotipos *Rack*, *Node* y *Hardware* se han marcado como abstractos, ya que los componentes modelados deberán ser de un de tipo específico que herede de uno de ellos.

Probablemente algunos centros de datos tendrán estantes y nodos iguales, así como muchos nodos tendrán el mismo hardware, por lo que, para facilitar la reusabilidad de los componentes, y debido al diseño modular del simulador, se han definido sus relaciones como asociaciones entre los estereotipos (ver Figura 5). De esta manera, cada componente podrá ser referenciado, por otros componentes, siempre que sea necesario. También se han definido las propiedades, tipos de datos (DataType) y enumeraciones necesarias.

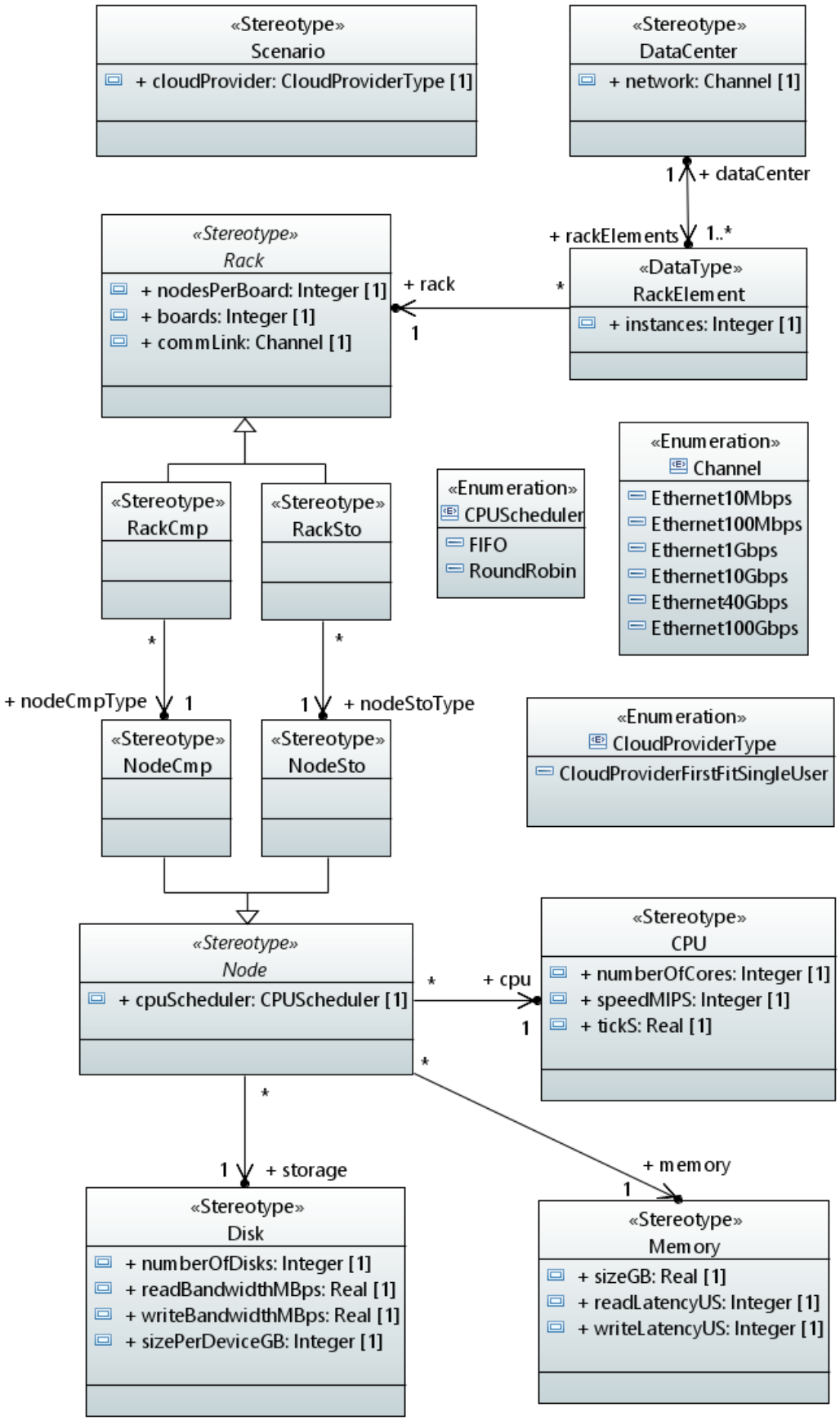


**Figura 5. Perfil SIMCAN - Definición de atributos de los componentes**

## Interacción

Para la interacción de los usuarios con el sistema cloud se ha definido, junto con el grupo de investigación que está desarrollando el simulador, un protocolo de comunicación, que será el modelado en el diagrama de secuencia.

Debido a que el simulador todavía está en desarrollo se decide que primero se definirá e implementará un protocolo sencillo, en el que cada usuario podrá solicitar una máquina virtual y ejecutar en ella una aplicación. Esta funcionalidad se irá ampliando posteriormente, permitiendo al usuario solicitar diferentes máquinas virtuales y ejecutar varias aplicaciones. La Figura 6 ilustra esquemáticamente los principales elementos involucrados en estas interacciones.

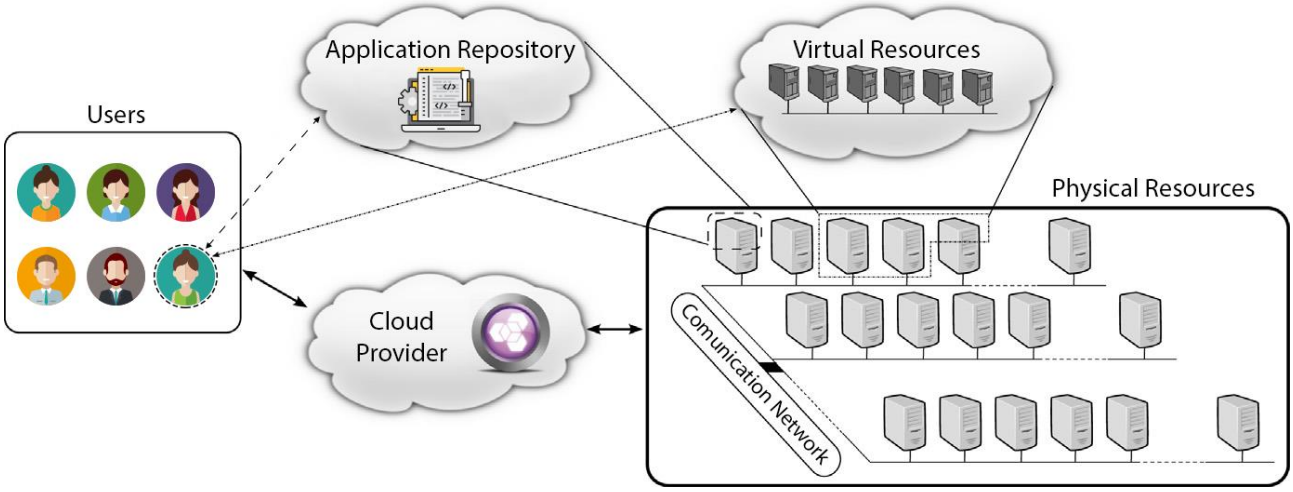


**Figura 6. Esquema de interacción entre usuarios y SIMCAN**

Se modela el protocolo en un diagrama de secuencia, tal como se muestra en la Figura 7. Este diagrama de secuencia representa la interacción de un usuario con el sistema cloud y será el simulador, cuando se ejecute, el que solicite información acerca de la cantidad de usuarios y el ritmo de llegada de éstos al sistema, dependiendo del tipo de simulación que se desee realizar.

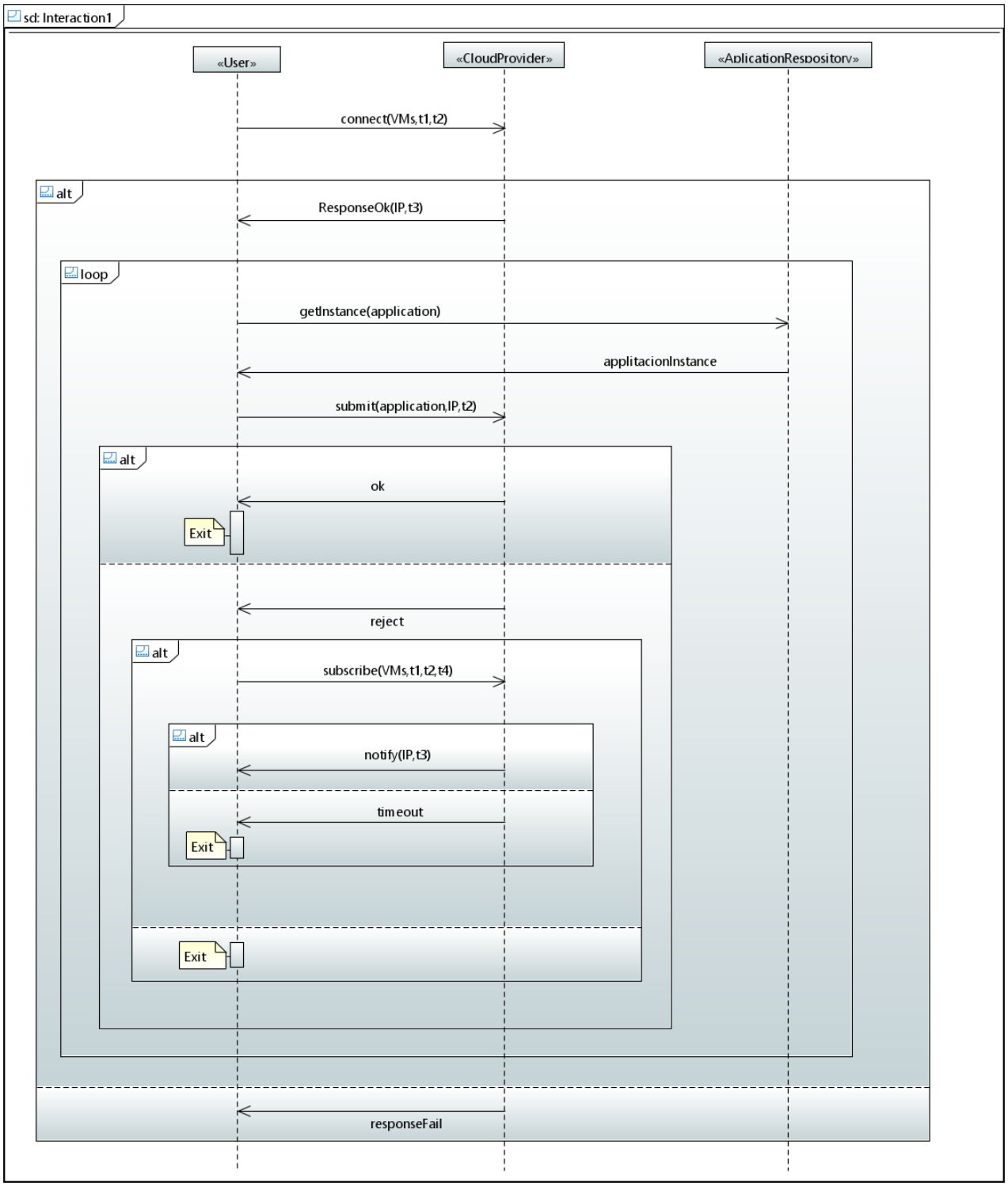


**Figura 7. Modelo protocolo de interacción del usuario con SIMCAN**

El protocolo comienza cuando el usuario llega al sistema cloud y solicita (*connect*) al (*Cloud Provider*) una máquina virtual (*VM*}), de la cual se especifican sus características en términos de CPU, memoria y capacidad de almacenamiento.

También indica la hora máxima hasta la que está dispuesto a esperar (*t1*) y cuánto tiempo la necesita (*t2*). El *Cloud Provider* contestará (*responseOK*) al usuario con la direción IP de la máquina y la hora a la que estima que estará disponible (*t3*) o bien con *responseFail* si no hay disponible ninguna máquina que pueda satisfacer al usuario dentro del tiempo que él necesita, y terminará la ejecución. Si ha recibido la IP de la máquina física que albergará la *VM*, solicitará (*getInstance*) al repositorio de aplicaciones (*ApplicationRepository*) una instancia de la aplicación (*applicationInstance*) y la enviará (*submit*) a la máquina que le indicó el *Cloud Provider*. Éste informará al usuario si la aplicación ha finalizado correctamente (*OK*), y terminará la ejecución, o bien si no se ha podido completar la ejecución (*reject*), por ejemplo, porque la máquina virtual no estuviera disponible finalmente. En éste último caso, si el usuario sigue interesado en la ejecución de la aplicación, hará una suscripción (*subscribe*), indicando hasta cuándo durará la suscripción (*t4*), para que se le notifique (*notify*) cuando haya una máquina que cumpla con su solicitud (*IP*, *t3*), si el tiempo de suscripción expira se produce un *timeout*. Si el usuario es notificado de que hay una máquina disponible dentro del tiempo de suscripción, se realizará la solicitud de la instancia de la aplicación (*getInstance*) y continuará la ejecución desde ese punto.

En la Figura 8 se muestran los estereotipos para este protocolo de interacción.

Se han definido los estereotipos *Application* y *VM* debido a que el usuario sólo podrá solicitar máquinas virtuales y aplicaciones que se hayan implementado previamente en el simulador y definido en el perfil. El estereotipo *Application* se ha marcado como abstracto y las aplicaciones implementadas deberán heredar de dicho estereotipo. También se definieron las asociaciones, propiedades, tipos de datos, enumeraciones y restricciones necesarias para poder modelar la interacción de los usuarios, como se muestra en la Figura 9.

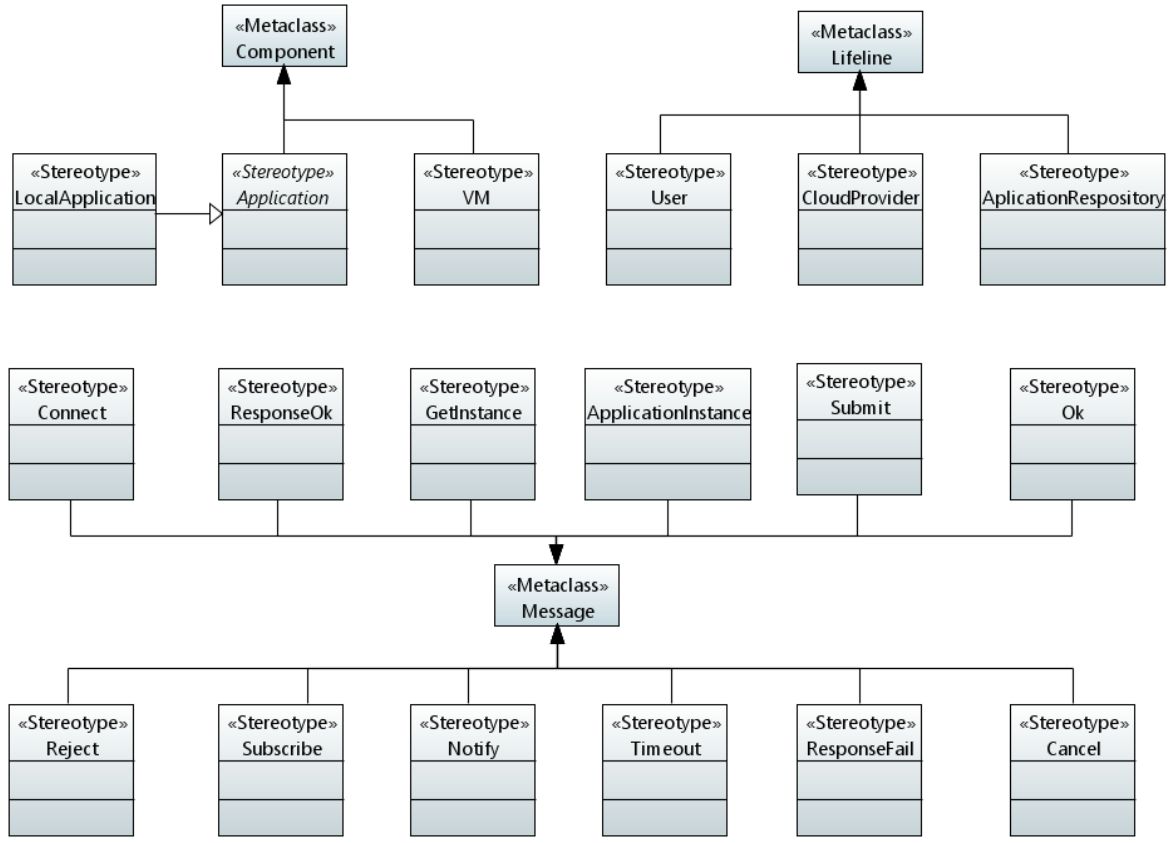


**Figura 8. Perfil SIMCAN - Definición de estereotipos de la interacción**

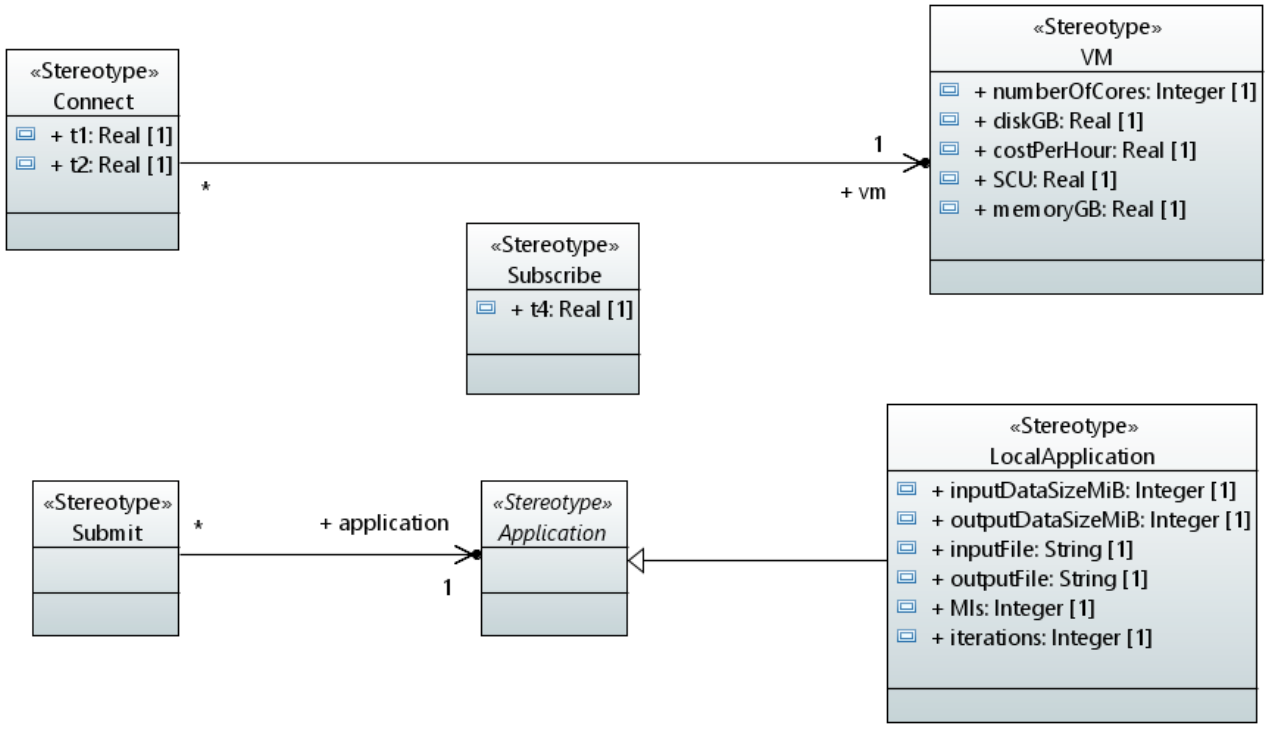


**Figura 9. Perfil SICMAN - Definición de atributos de la interacción**

### Despliegue del perfil

Una vez definido el perfil completo se debe registrar y desplegar en un plugin para una fácil instalación y uso del mismo. Para ello se han generado el *EMF Generator Model* y el *Ecore Model* a partir del perfil definido en un fichero *.uml*.

También se han definido los siguientes puntos de extensión siguiendo la documentación de Papyrus para la generación de perfiles estáticos[3]:

- org.eclipse.emf.ecore.generated_package
- org.eclipse.emf.ecore.uri_mapping
- org.eclipse.uml2.uml.generated_package
- org.eclipse.papyrus.uml.extensionpoints.UMLProfile

## Validación del modelo

En muchas ocasiones los lenguajes de modelado, tanto UML como un DSML definido por nosotros, son suficientes para proporcionar todos los aspectos relevantes de una especificación. Hay, entre otras cosas, una necesidad de describir restricciones adicionales sobre los objetos en el modelo para realizar la validación. La descripción de estas restricciones mediante lenguaje natural suele resultar en ambigüedades y la desventaja de los lenguajes formales tradicionales es que necesitan un alto conocimiento matemático para su utilización.

Debido a esto, surge la necesidad de utilizar un lenguaje que nos permita describir restricciones sobre nuestros modelos que cumplan con los estándares del OMG, como OCL, que es un lenguaje diseñado para este fin.

[3] Papyrus - Generating Static Profiles: https://help.eclipse.org/neon/topic/org.eclipse.papyrus.uml.diagram.profile.doc/target/generated-eclipse-help/users/generatingStaticProfiles.html

## OCL

Object Constraint Language (OCL) [15] es un lenguaje formal estandarizado por el OMG que nos permite describir estas restricciones. Es un lenguaje de especificación pura, por lo que no realiza cambios en el modelo cuando se evalúa una expresión OCL, simplemente devuelve un valor. Esto significa que el estado del sistema nunca cambiará debido a la evaluación de una expresión OCL. No es un lenguaje de programación, por lo que no permite escribir lógica de programa o control de flujo.

## Validación modelos SIMCAN

Para validar que los modelos, se han definido, a parte de las propias definidas por las asociaciones entre estereotipos y sus cardinalidades, una serie de restricciones según las características del simulador. Se muestra un ejemplo de algunas de estas restricciones en la Figura 10.

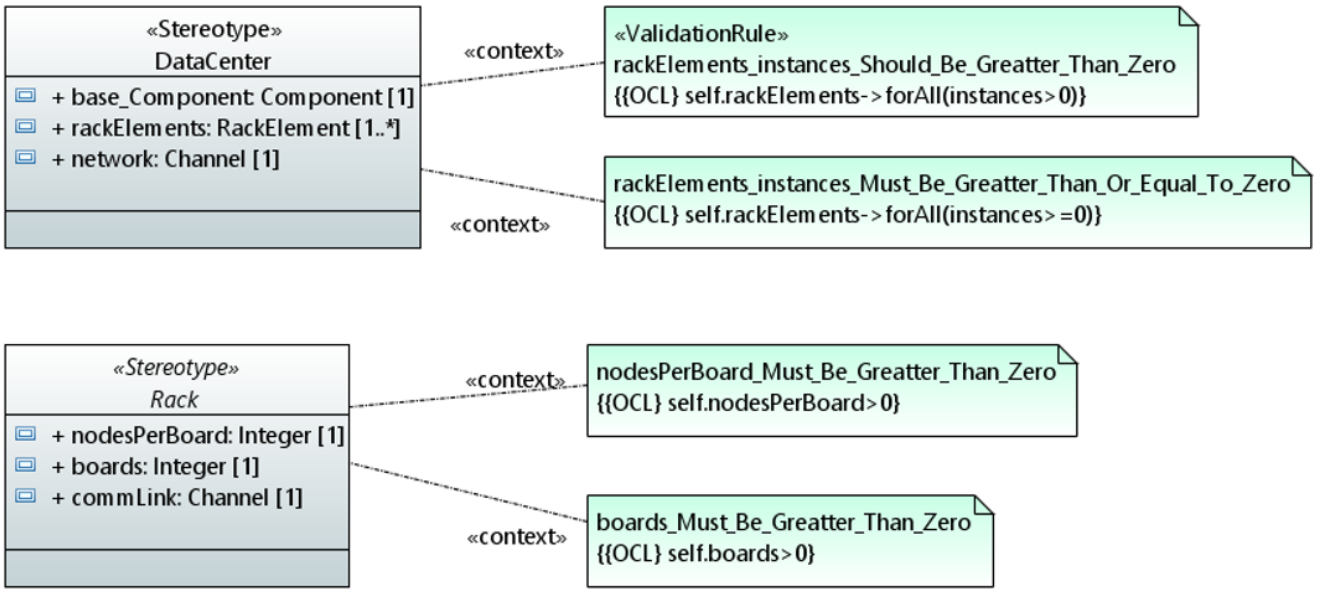


**Figura 10. Restricciones OCL sobre el perfil SIMCAN**

## Transformación

Una de las características más potentes del MDD es la posibilidad de automatizar las transformaciones. Para ello, es necesario definir estas transformaciones de una manera legible por una máquina para poder así ejecutarlas y generar el modelo destino automáticamente. Para abordar esta cuestión, el OMG define dos estándares dependiendo del tipo de transformación. Define el estándar QVT para transformaciones M2M y el estándar MOFM2T para transformaciones M2T.

### QVT

Query/Views/Transformation (QVT) [16] es el lenguaje estándar de OMG para especificar transformaciones en MDA. Este lenguaje nos va a permitir manipular modelos para realizar la transformación y que, generalmente, permite más de un modelo de entrada y más de un modelo de salida.

La plataforma Eclipse implementa este estándar con el lenguaje ATL.

### MOFM2T

MOF Model to Text (MOFM2T) [17] es el estándar del OMG para traducir un modelo a varios artefactos de texto como código, especificaciones de despliegue, informes y documentos. Este estándar aborda cómo transformar un modelo en una representación de texto lineal.

Para ello, utiliza un enfoque basado en plantillas en el que el texto que se va a generar a partir del modelo se especifica como un conjunto de plantillas de texto que están parametrizadas con elementos del modelo. Estas plantillas especifican la estructura del texto con marcadores de posición para los datos que se extraen de los modelos. Estos marcadores son esencialmente expresiones especificadas sobre las entidades del metamodelo, siendo las consultas el principal mecanismo para seleccionar y extraer los valores del modelo.

La plataforma Eclipse implementa este estándar con el lenguaje Acceleo [18], que está basado en módulos (ficheros *.mtl*), los cuales contienen plantillas (*Templates*), para generar código, y consultas (*Queries*), para extraer información del modelo. Durante el desarrollo de este trabajo se especificará una transformación desde un perfil de UML a los ficheros de configuración del simulador, por lo que será éste el lenguaje utilizado para su implementación.

**Transformación SIMCAN**

El objetivo de este trabajo es que el usuario pueda modelar gráficamente el escenario de un sistema cloud y obtener automáticamente los ficheros de configuración de este modelo para el simulador, por lo que la entrada de la primera transformación debe ser un modelo y la salida de la última transformación debe ser un fichero de texto.

Se ha creado el proyecto `es.uclm.SIMCAN.M2Ttransformation` para la implementación de las transformaciones, dentro del cual se han creado los siguientes paquetes, módulos y archivos:

- `es.uclm.SIMCAN.M2Ttransformation.files`: Este paquete contiene los módulos para generar cada uno de los ficheros.
    - `util.mtl`: Este módulo contiene consultas (*Query*) auxiliares para ser utilizadas en el resto de módulos.
    - `generateNEDFile.mtl`: Este módulo contiene las plantillas (*Template*) para generar el fichero *.ned*. y usa el fichero `util.mtl`.
    - `generateINIFile.mtl`: Este módulo contiene las plantillas (*Template*) para generar el fichero *.ini* y usa el fichero `util.mtl`.
- `es.uclm.SIMCAN.M2Ttransformation.main`: Este paquete contiene el módulo principal que será llamado cuando se ejecute la transformación y llamará al resto de módulos.
    - `generateSIMCANFiles.mtl`: Este es el módulo principal ya que

tiene la etiqueta `@main` y se ejecuta automáticamente al lanzar la transformación. Se encarga de ejecutar las plantillas principales del resto de módulos.

Para implementar las transformaciones que generan el fichero *.ned*, se ha creado el módulo `generateNEDFile.mtl`, dentro del cual se ha implementado una plantilla principal que crea el archivo en una carpeta con el nombre del escenario y después ejecuta otras plantillas que escriben el contenido del archivo, tal como se muestra en el Código 1. Módulo generateNEDFile.mtl Debido a que la configuración en el archivo *.ned* está escrita de forma anidada, muchas de las llamadas a otras plantillas también se hacen de forma anidada. Del mismo modo se ha creado el módulo `generateINIFile.mtl` para generar el fichero *.ini*.

```
[ comment encoding = UTF -8 /]
[ module generateNEDFile ('http :// www. uclm .es/ UML/ profiles / SIMCAN /1 ')]
[ import es :: uclm :: SIMCAN :: M2Ttransformation :: files :: util /]
[ template public generateNEDFile ( aScenario : Scenario )]
[ file ( aScenario . getFolder ()+'scenario . ned ', false , 'UTF -8 ')]
[ aScenario . generateHeaderNEDFile () /]
[ aScenario . getDataCenters (). generateDataCenter () /]
[ aScenario . generateScenario () /]
[/ file ]
[/ template ]
[ template public generateHeaderNEDFile ( aScenario : Scenario )]
        :
[/ template ]
[ template public generateDataCenter ( aDataCenter : DataCenter )]
        :
[/ template ]
        :
        :
```

**Código 1. Módulo generateNEDFile.mtl**

La transformación desde el modelo del sistema cloud a los archivos de configuración de SIMCAN es amplia y compleja y está compuesta por varias subtransformaciones, cada una de las cuales puede estar a su vez compuesta por otras subtransformaciones. Por ello, se ha usado un gráfico llamado diagrama de descomposición funcional para representar la transformación a los ficheros *.ned* (ver Figura 11) y *.ini* (ver Figura 12).

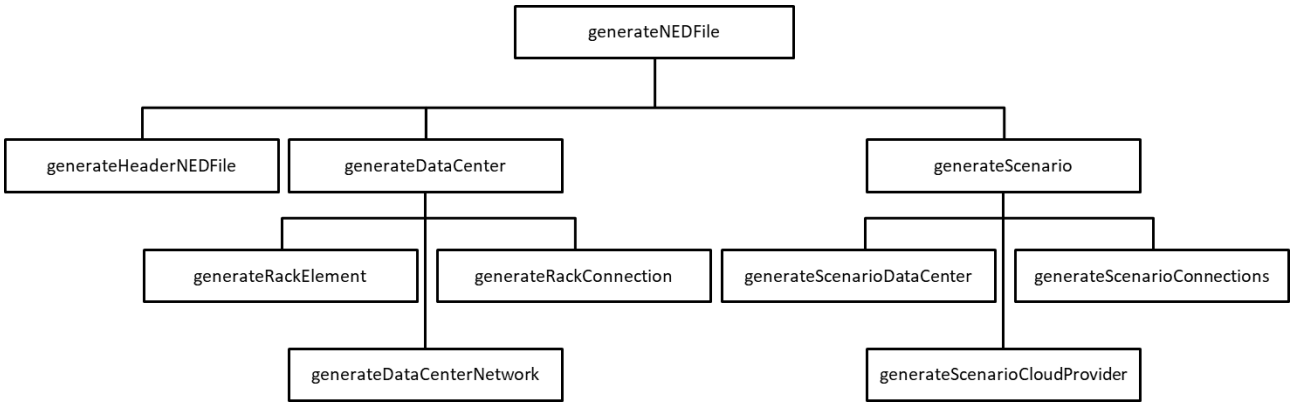


**Figura 11. Diagrama de descomposición funcional para la transformación M2T a .ned**

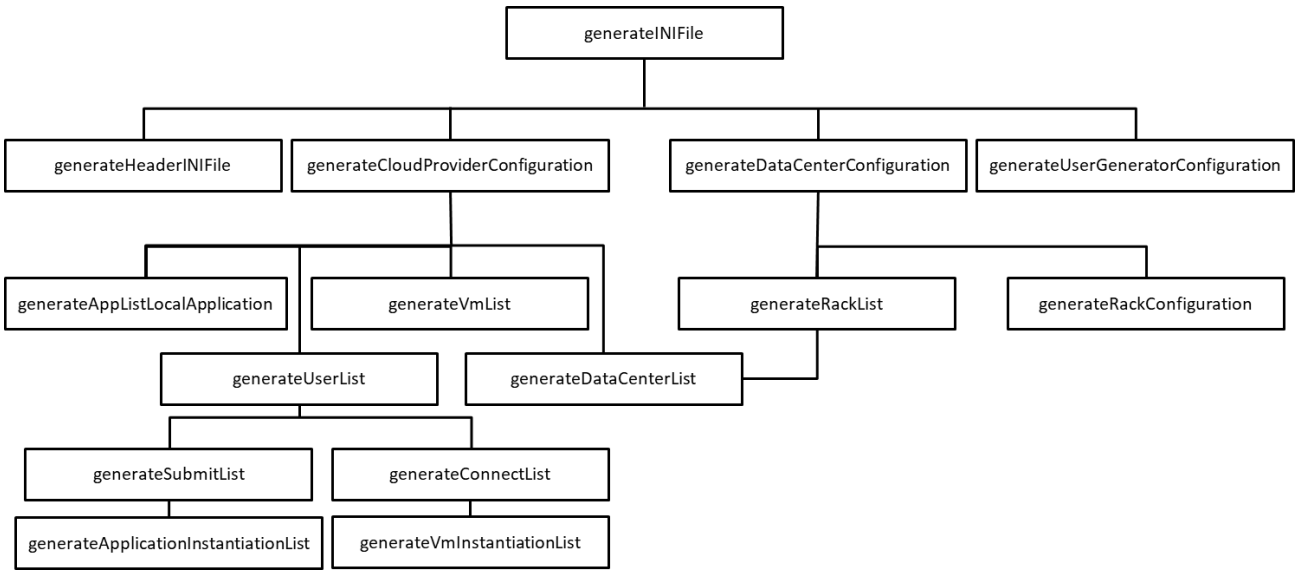


**Figura 12. Diagrama de descomposición funcional para la transformación M2T a .ini**

Cuando se ejecuta la transformación, cada elemento del modelo genera varios fragmentos en los ficheros de destino. Como se puede ver en el siguiente ejemplo, algunos elementos de un fragmento de un modelo simple de un sistema cloud (ver Figura 13) son transformados en varias líneas de texto a lo largo de los diferentes ficheros de configuración (ver Código 2, Código 3 y Código 4). Para facilitar la lectura, los fragmentos de texto se han simplificado omitiendo las líneas que no son generadas por los elementos mostrados en la Figura 13 y se han reemplazado por tres almohadillas azules (###). El texto generado por los valores de las propiedades de los elementos del modelo se muestra en verde.

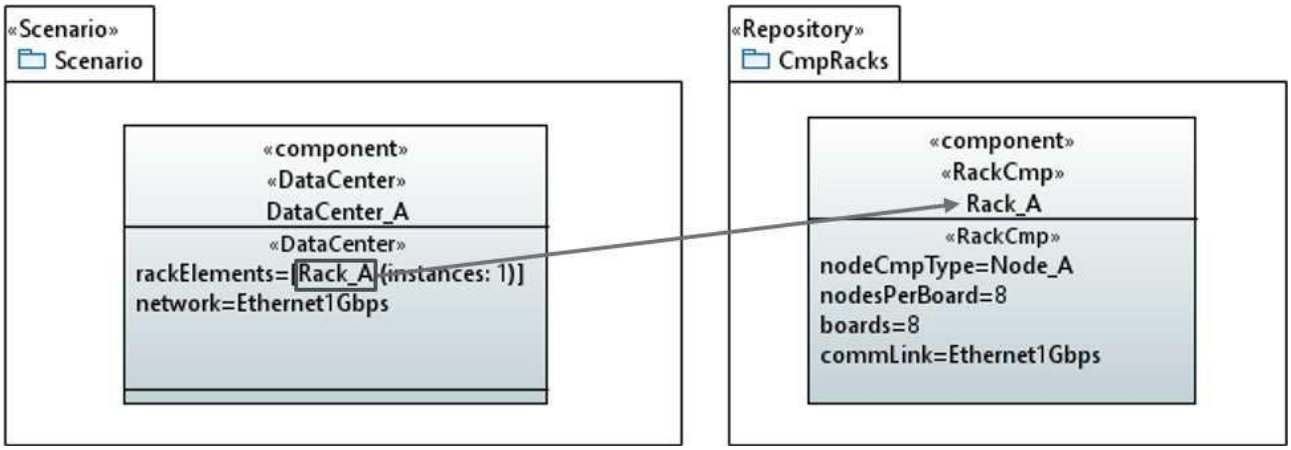


**Figura 13. Fragmento de un modelo de un sistema cloud**

```
network Scenario {
  ###
  submodules:
    dc_DataCenter_A_000:DataCenter_A {
      ###
    }
    cloudManager:CloudSystemManager {
      ###
      gates:
        fromDataCenter[1];
        toDataCenter[1];
    }
    ###
    connectionsallowunconnected:
      systemManager.toDataCenter++ --> ned.IdealChannel --> dc_DataCenter_A_000.in;
      systemManager.fromDataCenter++ <-- ned . IdealChannel <-- dc_DataCenter_A_000.out;
}
```

**Código 2. Fragmento 1 del fichero .ned generado**

```
module DataCenter_A {
  ###
  rackCmp_Rack_A_000:CloudRack;
  dataCenterNetwork:DataCenterNetwork {
    gates:
      inComm[1];
      outComm[1];
      inStorage[0];
      outStorage[0];
  }
  connectionsallowunconnected:
    cloudManager.out --> ned.IdealChannel --> toCloudProvider;
    cloudManager.in <-- ned.IdealChannel <-- fromCloudProvider;
    rackCmp_Rack_A_000.out --> DataCenterEth1G_channel --> dataCenterNetwork.inComm++;
    rackCmp_Rack_A_000.in <-- DataCenterEth1G_channel <-- dataCenterNetwork.outComm++;
}
```

**Código 3. Fragmento 2 del fichero .ned generado**

```
Scenario.appList = "2 LocalApplication AppCPUIntensive 6 inputDataSi z e i n t MiB 10 ### "
Scenario.vmList = "3 VM_large 23.0 4 4.0 1000.0 8.0 VM_medium 15.0 2 2.0 500.0 4.0 ### "
Scenario.userList = "2 User:A 32 AppCPUIntensive 4 AppDataIntensive 2 3 VM large 1 ### "
Scenario.dataCentersList = "1 DataCenter_A 1 1 Rack_A 8 8 Node_A 500 4 . 0 2 30000 2 ### "
Scenario.DataCenter_A.dcManager.dataCenterConfig ="DataCenter_A 1 1 Rack_A 8 8 Node_A###"
Scenario.dc_DataCenter_A_000.rackCmp_Rack A _ *. channelType = "DataCenterEth1G channel"
Scenario.dc_DataCenter_A_000.rackCmp_Rack A _ *. numBoards = 8
Scenario.dc_DataCenter_A_000.rackCmp_Rack A _ *. bladesPerBoard = 8
Scenario.dc_DataCenter_A_000.rackCmp_Rack A _ *.board[*].blade [*].staticAppAssignment ###
Scenario.dc_DataCenter_A_000.rackCmp_Rack A _ *.board[*].blade [*].isVirtualHardware ###
Scenario.dc_DataCenter_A_000.rackCmp_Rack A _ *.board[*].blade [*].maxUsers ###
Scenario.dc_DataCenter_A_000.rackCmp_Rack A _ *.board[*].blade [*].maxVMs ###
Scenario.dc_DataCenter_A_000.rackCmp_Rack A _ *.board[*].blade [*].numCpuCores ###
Scenario.dc_DataCenter_A_000.rackCmp_Rack A _ *.board[*].blade [*].cpu.cpuCore[*].speed ###
Scenario.dc_DataCenter_A_000.rackCmp_Rack A _ *.board[*].blade [*].cpu.cpuCore [*].tick ###
Scenario.dc_DataCenter_A_000.rackCmp_Rack A _ *.board[*].blade [*].appsVectors[*]. ###
```

**Código 4. Fragmento 1 del fichero .ini generado**

## Validación de la transformación

La calidad y éxito del proceso MDE llevado a cabo depende en gran medida de la calidad de los modelos y de las transformaciones. La fiabilidad del modelo resultante se incrementa mediante la validación de las transformaciones para evitar errores en cascada. Esta validación incluye por extensión la verificación de la corrección sintáctica de la misma y de los modelos, tanto el de origen como el de destino, así como la corrección semántica de la transformación, es decir, la correspondencia del sistema modelado mediante el modelo destino con el sistema modelado mediante el modelo fuente.

Existen diferentes enfoques a la hora de realizar la verificación sintáctica y/o semántica de la transformación [19]:

- **Basado en casos de prueba**: Se centra en la verificación de la transformación de un conjunto definido de modelos fuente y de sus correspondientes modelos destino. Dicha verificación se realiza para un conjunto representativo del dominio, generando, de esta forma, cierto nivel de confianza para el resto de transformaciones.
- **Verificación de modelos, (model-checking):** Consiste en verificar propiedades de forma automática sobre los modelos origen y destino, y sobre la transformación. Dicha verificación se realiza para todas las transformaciones a partir de un conjunto de restricciones definidas a nivel de metamodelo.
- **Métodos deductivos**: Centra la verificación sobre los metamodelos en lugar de los modelos. Se utilizan definiciones formales de los metamodelos y se definen las transformaciones como un conjunto de fórmulas lógicas, permitiendo realizar razonamientos formales. De esta forma, se garantiza que las propiedades verificadas se cumplen para todos los modelos capturados por dicho metamodelo.

Los enfoques de verificación sobre modelos se realizan a un nivel de abstracción inferior que la verificación sobre metamodelos, lo que permite validar un mayor

número de propiedades en los primeros, pero que no les permite asegurar la corrección en términos absolutos.

La validación de las transformaciones M2T es limitada, ya que la validación de los ficheros de texto también lo es, por ejemplo, impidiendo el uso de OCL sobre el resultado o la utilización de cualquier tipo de grafo, por lo que se ha realizado la validación basada en casos de prueba. Se ha validado la salida de cada una de las reglas de transformación, es decir, se han realizado test unitarios, con expresiones regulares, como la solución propuesta por Tiso y otros [20]. El usuario deberá validar el modelo antes de ejecutar la transformación teniendo en cuenta que si hay algún error en el modelo es posible que el resultado de la transformación no sea correcto, por lo que el modelo de pruebas deberá cumplir con las restricciones del perfil.

Dentro del proyecto `es.uclm.SIMCAN.M2Ttransformation` se han creado los siguientes paquetes, módulos y archivos para validación de las transformaciones:

- `es.uclm.SIMCAN.M2Ttransformation.unitTests`: Este paquete contiene los ficheros y módulos necesarios para la validación de las plantillas (*Template*) mediante pruebas unitarias.
    - TestServices.java: Este fichero contiene algunas funciones auxiliares implementadas en java que podrán ser ejecutadas desde Acceleo mediante *Java services wrappers*, es decir, permitirá extender el lenguaje con java y es usado para implementar algunas funciones auxiliares.
    - `testServices.mtl`: Este módulo contiene una plantilla por cada función implementada en el fichero `TestServices.java` para facilitar su ejecución.
    - `testSIMCANM2Ttransformation.mtl`: Este módulo contiene una plantilla de validación para cada una de las plantillas de transformación de los ficheros `generateNEDFile.mtl` y `generateINIFile.mtl` y genera un fichero

`unitTestResults.html` para mostrar los resultados de la validación. Usa todos los ficheros anteriores.

En el módulo `testSIMCANM2Ttransformation.mtl` se ha implementado una plantilla principal que crea el archivo `unitTestResults.html` y su estructura dentro de la cual se llama a las plantillas que realmente validan las transformaciones, tal como se muestra en el Código 5.

```
[comment encoding = UTF -8 /]
[module testSIMCANM2Ttransformation('http://www.uclm.es/UML/profiles/SIMCAN/1')/]
[import es :: uclm :: SIMCAN :: M2Ttransformation :: files :: generateNEDFile /]
[import es :: uclm :: SIMCAN :: M2Ttransformation :: files :: generateINIFile /]
[import es :: uclm :: SIMCAN :: M2Ttransformation :: files :: util /]
[import es :: uclm :: SIMCAN :: M2Ttransformation :: services :: testService /]
[template public testSIMCANM2Ttransformation ( aScenario : Scenario )]
[ file ( aScenario.getFolder ()+'unitTestResults.html ', false , 'UTF -8 ')]
<! DOCTYPE html >
<html >
  <head >
    <meta charset =" ISO -8859 -1" >
    <title > SIMCAN M2T Transformation Unit Tests </ title >
  </head >
  <body >
    <table style ="..." >
      <tr>
        <td colspan ="3" > SIMCAN M2T Transformation Unit Tests </td >
      </tr>
      <tr>
        <td colspan ="3" > File under test: generateNEDFile.mtl </td >
      </tr>
      [aScenario.testGenerateHeaderNEDFile() /]
      [aScenario.siblings( DataCenter ).testGenerateDataCenter() /]
            :
    </table >
  </body >
</html >
```

**Código 5. Módulo testSIMCANM2Ttransformation.mtl**

Las plantillas de validación ejecutan la plantilla `machRe(String, String)` del módulo `testServices.mtl`, pasándole como parámetros la expresión regular y la salida de la función en pruebas, y escribe `OK` o `FAIL` dependiendo de si ésta devuelve `true` o `false`, tal como se muestra en el Código 6.

```
        :
[ template public testGenerateDataCenter( aDataCenter : DataCenter )]
<tr>
   <td colspan ="3" > Function under test : generateDataCenter ( aDataCenter : DataCenter ) </td >
</tr>
   [printHeader() /]
<tr style =" background-color:# FFFFFF ">
   <td >[aDataCenter.base_Component.name /] </td >
      [if ( matchRE ('package ((\\ s *\\/\\/.*[\\ r\\n ]+\\ s*) +|\\ s+) simcan2 \... ',
        aDataCenter . generateDataCenter ()))]
   <td style =" background - color : green ">OK </td>
   <td ></td>
      [ else ]
   <td style =" background - color : red ">FAIL </td>
   <td>[ aDataCenter . generateDataCenter () /] </td>
      [/if]
</tr>
[/ template ]
        :
[template public printHeader( arg : OclAny )]
<tr >
   <td > Model Element </td >
   <td > Test Result </td >
   <td > Text Fragment </td >
</tr >
[/template ]
```

**Código 6. Módulo testSIMCANM2Ttransformation.mtl**

La plantilla `machRe()` ejecuta la función con el mismo nombre del archivo `TestServices.java` mediante la operación `invoke()`, implementada en Acceleo, que nos permite hacer uso de los *Java Service Wrapper*[4], tal como se muestra en el Código 7.

[4] Acceleo - Java Service Wrapper: https://wiki.eclipse.org/Acceleo/Getting_Started#Java_services_wrappers

```
[ module testService (’http://www.uclm.es/UML/profiles/SIMCAN/1’)/]
[ query public matchRE ( arg0 : String , arg1 : String ) : Boolean
= invoke (’es. uclm . SIMCAN . M2Ttransformation . services . TestService ’, ’
matchRE ( java.lang.String,  java.lang.String )’, Sequence {arg0 , arg1}).oclAsType( Boolean )/]
```

**Código 7. Módulo testService.mtl**

Finalmente la función `machRe()`, implementada en Java, devuelve `true` o `false` dependiendo de si el texto generado cumple con el patrón o no, tal como se muestra en el Código 8.

```
import java.util.regex.Matcher ;
import java.util.regex.Pattern ;
public class TestService {
   public Boolean matchRE ( String regularExpresion , String texto ){
      Pattern pat = Pattern.compile( regularExpresion );
      Matcher mat = pat.matcher( texto );
      return mat.matches();
   }
}
```

**Código 8. Archivo TestService.java**

Como hemos visto, se han usado expresiones regulares para comprobar que obtenemos el resultado esperado de cada plantilla o regla de transformación. Para definir estas expresiones regulares se ha repasado la estructura de los ficheros de configuración y se le han añadido grupos de expresiones regulares como `((\s*\/\/.*[\r\n]+\s*)+|\s+)` la cual permite añadir espacios y comentarios o `[\s\S]*` que representa cualquier texto, para permitir la inserción de texto por otra plantilla. Esta segunda plantilla tiene su propia función de validación por lo que no es necesario volver a comprobar su resultado cuando es ejecutada por otra plantilla.

Para comprobar que la implementa una transformación es correcta se ejecuta el módulo `testSIMCANM2Ttransformation.mtl` y se revisan los resultados en el fichero `unitTestResults.html`, cuyo resultado se puede ver en la Figura 14. Resultado de la validación en unitTestResults.html, en la cual también se muestra

como aparece si la salida de una plantilla no es correcta.

| SIMCAN M2T Transformation Unit Tests | | |
|---|---|---|
| File under test: generateNEDFile.mtl | | |
| Function under test: generateHeaderNEDFile(aScenario : Scenario) | | |
| Model Element | Test Result | Text Fragment |
| Scenario1 | OK | |
| Function under test: generateDataCenter(aDataCenter : DataCenter) | | |
| Model Element | Test Result | Text Fragment |
| DataCenter1 | FAIL | module DataCenter1 {<br>parameters:<br>string appList;<br>string vmList;<br>string userList;<br><br>gates:<br>input in;<br>output out;<br>input fromCloudProvider;<br>output toCloudProvider; |

**Figura 14. Resultado de la validación en unitTestResults.html**

Una vez implementada y validada la transformación completa se creó el proyecto `es.uclm.SIMCAN.M2Ttransformation.ui`, seleccionando la opción nuevo *Acceleo UI Launcher Project* desde el asistente y referenciando el proyecto anterior tal como se muestra en la documentación de Acceleo[5] para poder ejecutar la transformación fácilmente desde el menú contextual de Papyrus.

[5] Acceleo - Creating a UI launcher: https://wiki.eclipse.org/Acceleo/Getting_Started\#Creating_a_UI_launcher

## Personalización del editor gráfico

Para la personalización del editor gráfico se ha creado el proyecto `es.uclm.SIMCAN.customization` dentro del cual se han creado los siguientes carpetas y archivos:

- `viewpoint`:
    - `SIMCANModeling.configuration`: Este archivo contiene la configuración general de la personalización. En él se especifican los diagramas a extender y las rutas del resto de archivos que son los que contendrán la personalización.
- `palettes`:
    - `Palette_ComponentSIMCAN.xml`: Este archivo contiene la personalización de la paleta para el diagrama de componentes. Añade a la paleta de herramientas los componentes del simulador ya estereotipados con el fin de disponer de los elementos necesarios para modelar la infraestructura del sistema cloud.
    - `Palette_SequenceSIMCAN.xml`: Este archivo contiene la personalización de la paleta para el diagrama de secuencia. Añade a la paleta de herramientas las líneas de vida y mensajes ya estereotipados con el fin de disponer de los elementos necesarios para modelar la interacción de los distintos roles del sistema.
- `properties`:
    - `SIMCAN.ctx`: Este fichero contiene la personalización de la vista de propiedades. Añade una pestaña nueva a la vista de propiedades que permite ver y editar las propiedades de los estereotipos de los elementos del modelo.
    - `ui`: Esta carpeta contiene un conjunto de archivos *.xwt* que se generan

automáticamente a partir del archivo anterior y que contienen la información gráfica de la vista de propiedades.

- `styles`:
    - `Style_SIMCAN.css`: Este archivo contiene la personalización de la apariencia de los componentes modelados implementada como una hoja de estilos.

**Paletas de herramientas**

Para personalizar las paletas de herramientas se han creado los ficheros `Palette_ComponentSIMCAN.xml` y `Palette_SequenceSIMCAN.xml` usando el editor de paletas de Papyrus, mostrado en la **Figura 15**, y accesible presionando con el botón derecho en la barra de herramientas y seleccionando `Customize`. Este editor genera los ficheros *.xml* en la ruta `<workspace-directory>\.metadata\.plugins\org.eclipse.papyrus.uml.diagram.common` y han sido copiados a la carpeta `palettes` del proyecto.

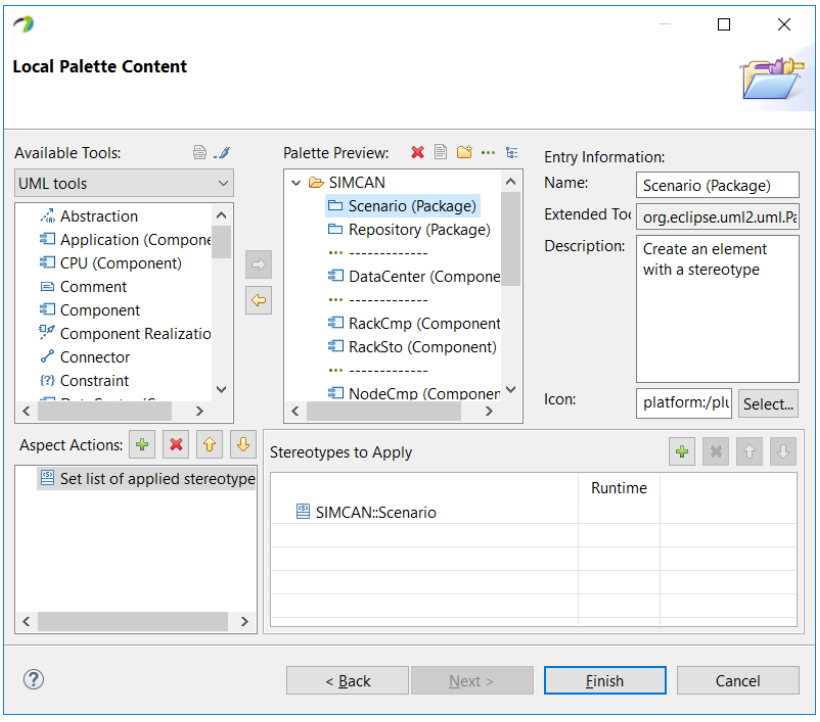


**Figura 15. Edición de paleta del diagrama de Componentes**

## Vista de propiedades

Para la personalización de la vista de propiedades se ha creado el archivo `SIMCAN.ctx` a partir del archivo *.uml* del perfil siguiendo la ayuda de Payrus sobre la personalización de la vista de propiedades[6] y se ha asignado una prioridad alta a la pestaña de SIMCAN para que aparezca en primer lugar. El fichero generado sólo incluye en la vista de propiedades los atributos del perfil, por lo que para editar el nombre del componente había que cambiar a la pestaña de UML. Para facilitar la edición se ha añadido también la propiedad *name* a la pestaña de propiedades de SIMCAN, tal como se muestra en la Figura 16, y se ha cambiado el tipo de *Widget* de los tipos de datos *RackElement*, *VMInstantiation*, y *ApplicationInstantation* a *Property Editor Type MultireferenceWithPropertyView* para ver las propiedades del elemento en la misma vista de propiedades, tal como se muestra en la Figura 17.

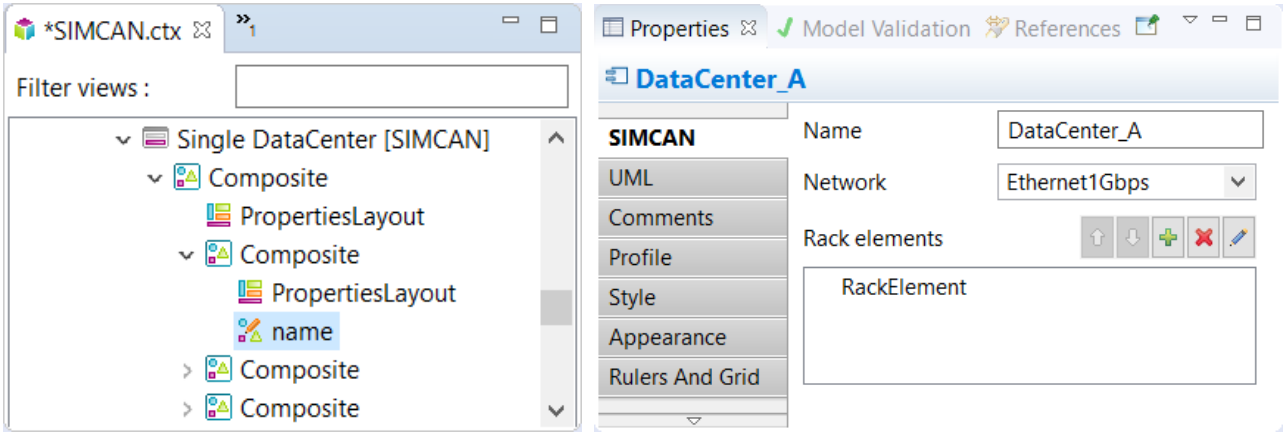


**Figura 16. Personalización archivo SIMCAN.ctx - *name***

[6] Papyrus - Properties View Generation Tool: https://help.eclipse.org/neon/topic/org.eclipse.papyrus.views.properties.doc/target/generated-eclipse-help/properties-view.html#Generation_Tool

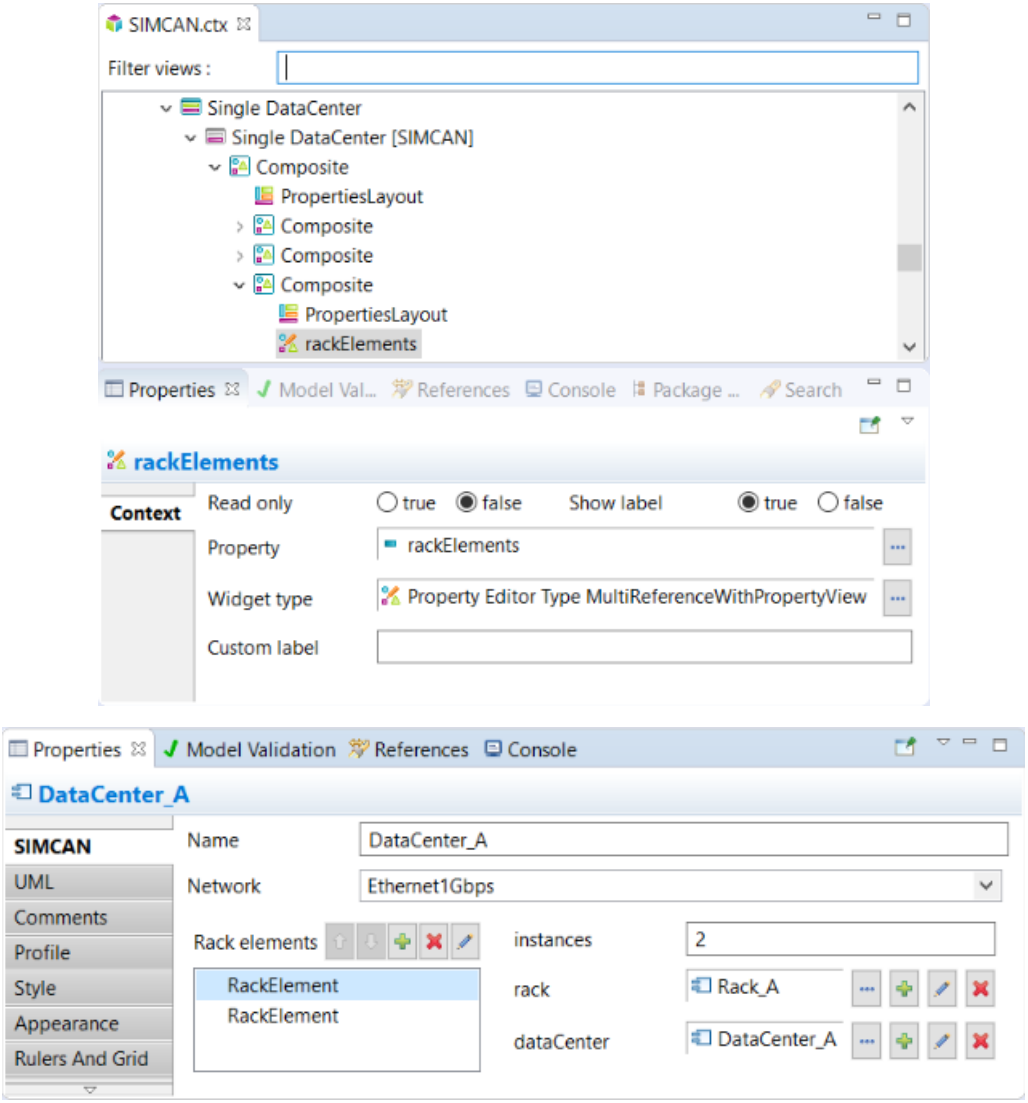


**Figura 17. Personalización archivo SIMCAN.ctx - *rackElements***

## Apariencia

Para personalizar la apariencia de los elementos en el diagrama se ha creado el archivo `Style_SIMCAN.css` (ver Código 9) siguiendo la documentación de Papyrus acerca de personalización con hojas de estilos[7].

```
Compartment[type=StereotypeCompartment]{
    visible:true;
}
```

**Código 9. Archivo Style_SIMCAN.css**

[7] Papyrus - CSS Stylesheets:
https://help.eclipse.org/neon/topic/org.eclipse.papyrus.infra.gmfdiag.css.doc/target/generated-eclipse-help/css.html

## Viewpoint

Para reunir todas las personalizaciones y aplicarlos a diagramas UML, y así reutilizarlos, se ha definido el punto de vista (*Viewpoint*) en el fichero `SIMCANModeling.configuration` siguiendo la documentación de Papyrus acerca de los *Viewpoints*[8] y se han especificado los diagramas a extender quedando configurado el diagrama de componentes como se muestra en la Figura 18 y el diagrama de secuencia como se muestra en la Figura 19.

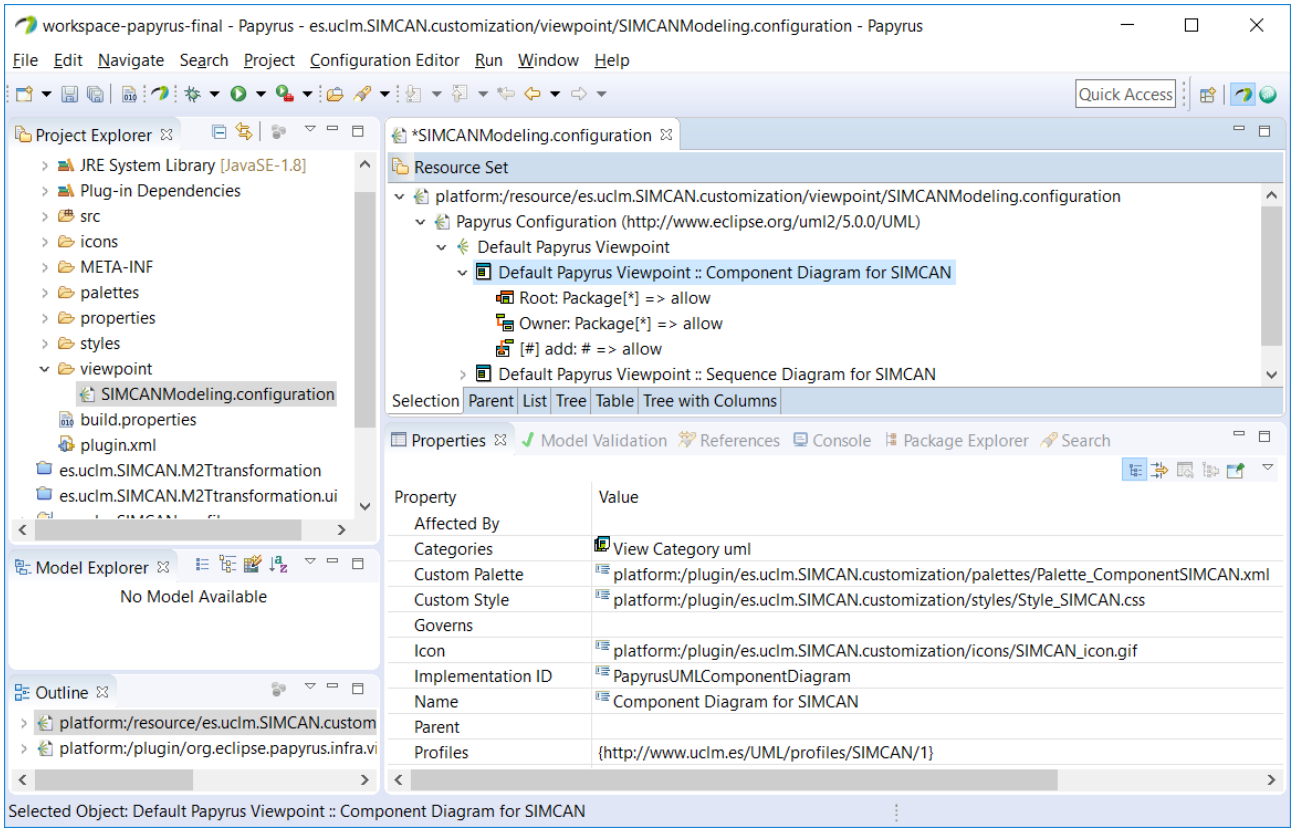


**Figura 18. Extensión del diagrama de componentes mediante viewpoint**

[8] Papyrus - Viewpoints: https://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.papyrus.infra.viewpoints.doc%2Ftarget%2Fgenerated-eclipse-help%2Fviewpoints.html

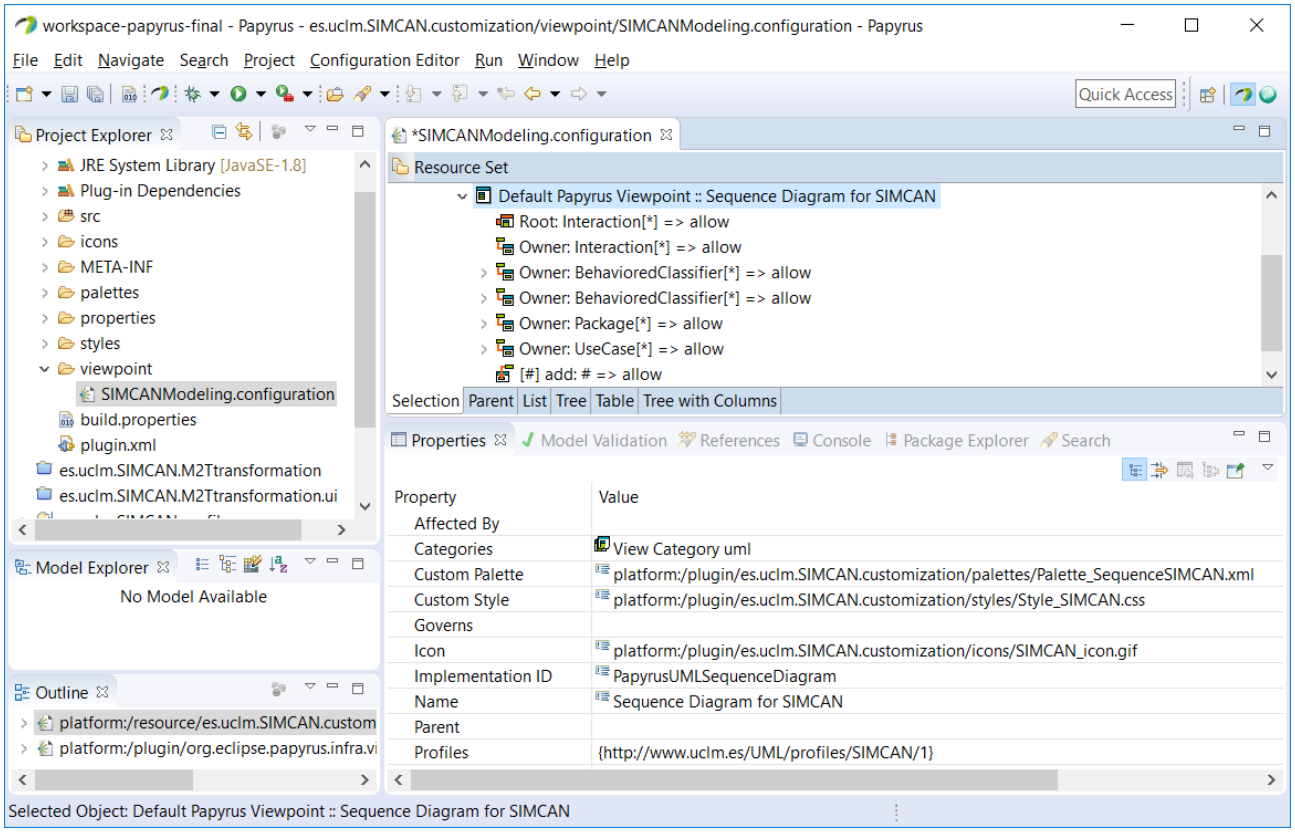


**Figura 19. Extensión del diagrama de secuencia mediante viewpoint**

### Despliegue de la personalización

Una vez definidas y configuradas todas las personalizaciones se han definido los siguientes puntos de extensión para desplegar la personalización:

- org.eclipse.papyrus.infra.viewpoints.policy.custom
- org.eclipse.papyrus.infra.properties.contexts

## Modelo base de sistema cloud

Para permitir al usuario modelar más rápidamente un sistema cloud, se ha modelado un sistema cloud completo con diferentes elementos en los repositorios para poder ser reutilizados.

Para permitir crear un proyecto con el modelo de ejemplo desde el asistente de Papyrus se han definido los siguientes puntos de extensión siguiendo la

documentación de Papyrus sobre la creación de proyectos de ejemplo[9]:

- org.eclipse.ui.newWizards
- org.eclipse.emf.common.ui.examples

## Despliegue de la herramienta

Una vez creados y desplegados todos los plugins es necesario volver a desplegarlos en un sitio web o en un solo archivo *.zip* para permitir una fácil distribución e instalación. Para ello se ha creado el proyecto `es.uclm.SIMCAN.feature`, como un nuevo *Feature Project*, que incluye el resto de plugins implementados hasta ahora. Por último, se ha creado el proyecto `es.uclm.SIMCAN.site`, como un nuevo *Update Site Project*, y se le ha añadido la *feature*. Este proyecto contiene todos los plugins en formato *.jar* y es el que se debe distribuir. Se puede acceder desde la dirección http://umltosimcan.esy.es/SIMCAN2/eclipse.

[9] Papyrus - Example Project: https://wiki.eclipse.org/Papyrus_for_Information_Modeling/Customization_Guide#Example_Project

## Resultados:

Durante la realización de este trabajo se pretendía facilitar el estudio de los sistemas cloud, para ello se ha seguido el proceso mostrado en la Figura 20 con el fin de aplicar la Ingeniería Dirigida por Modelos a la simulación de sistemas cloud.

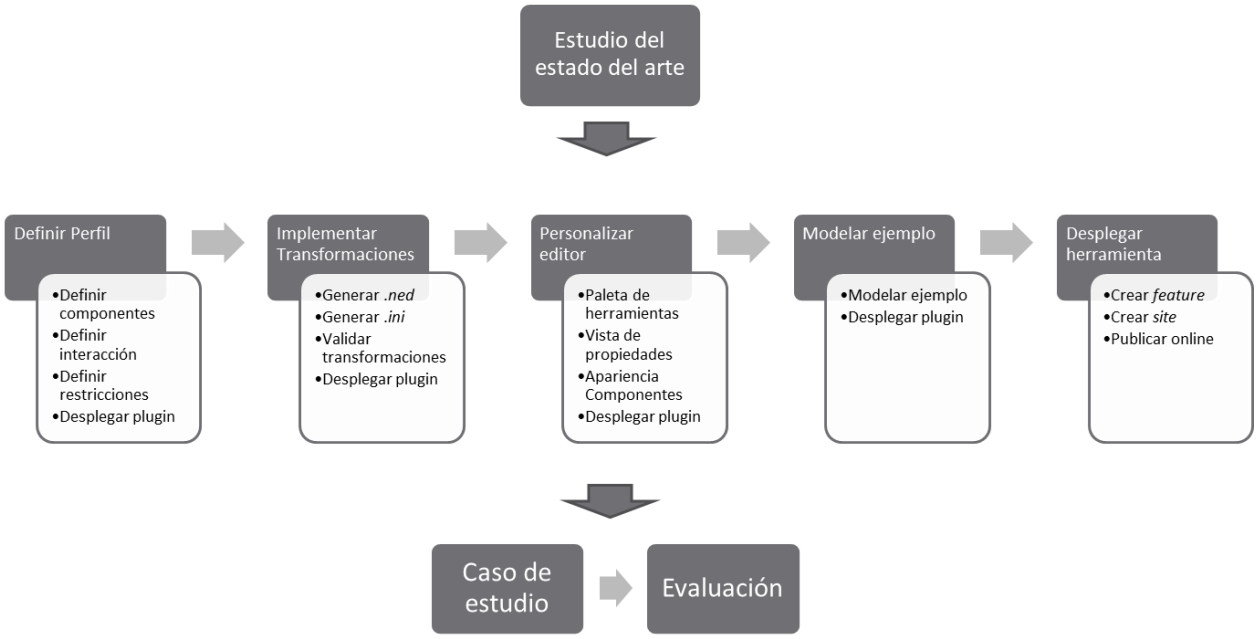


**Figura 20. Proceso de realización del trabajo**

De esta manera se ha obtenido un perfil UML que nos permite modelar una infraestructura cloud y la interacción de los usuarios con dicha arquitectura, y se ha definido una transformación de dicho perfil a los ficheros de configuración del simulador SIMCAN. Además, se ha obtenido un plugin que personaliza el editor gráfico de Papyrus (ver Figura 21) y que, de esta manera, facilita el modelado de sistemas cloud. Esta personalización nos permite modelar un sistema cloud a partir de un modelo base ya modelado, editar dicho modelo o crear uno nuevo añadiendo elementos mediante la paleta de herramientas personalizada y editando sus propiedades desde la pestaña SIMCAN de la vista de propiedades, y generar los ficheros de configuración de SIMCAN con respecto a estos modelos ejecutando la transformación fácilmente desde el menú contextual (ver Figura 22).

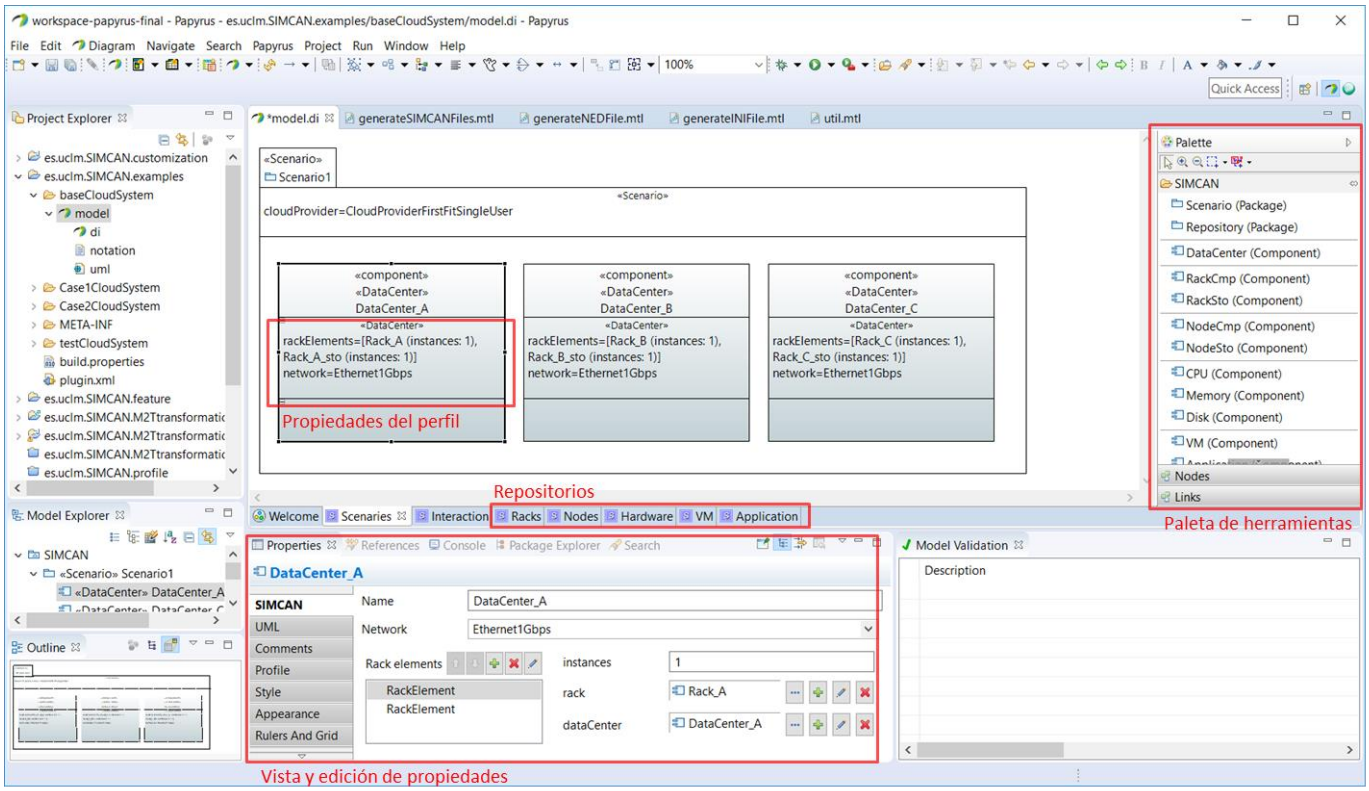


**Figura 21. Editor gráfico de Papyrus personalizado**

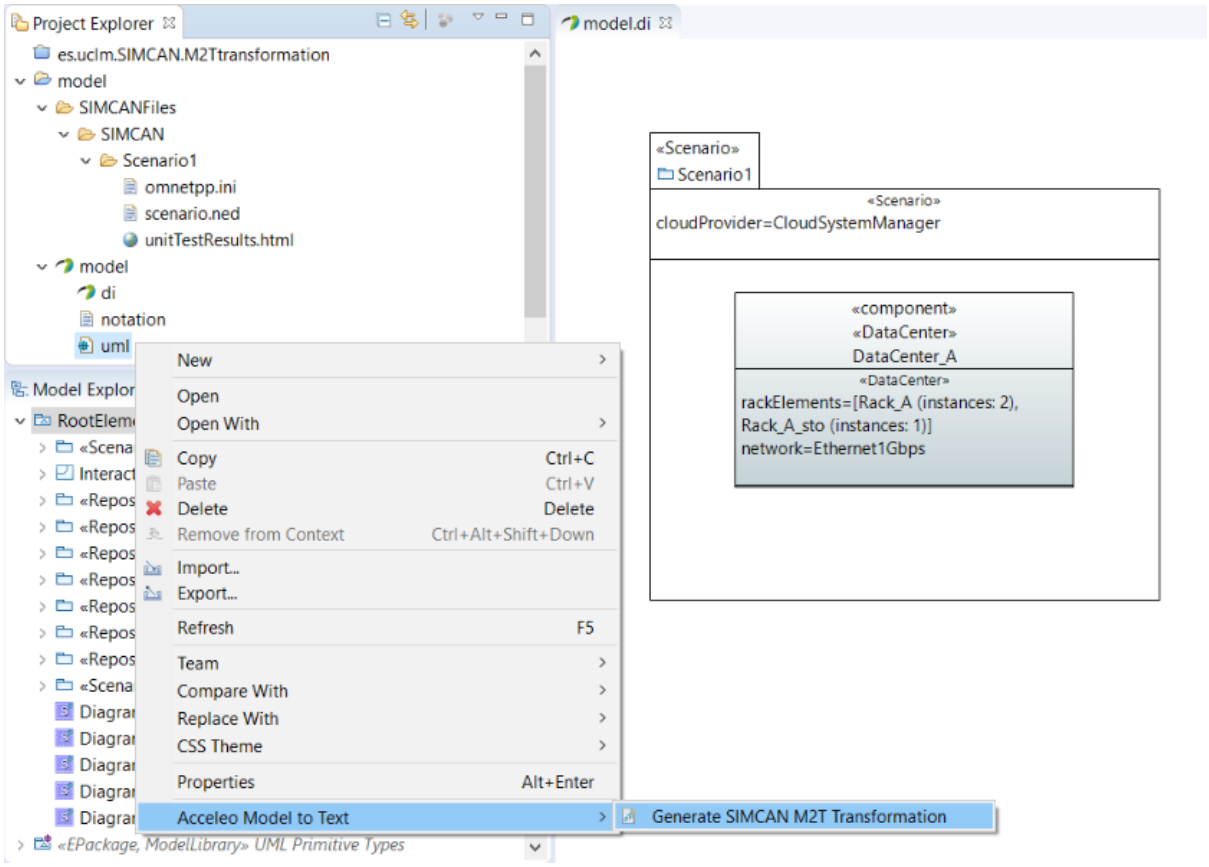


**Figura 22. Ejecución de la transformación desde el menú contextual**

Una vez obtenidos los ficheros de configuración del simulador con respecto a los sistemas cloud modelados (ver Figura 23) se debe ejecutar el simulador para conocer el comportamiento del sistema en cada uno de los escenarios modelados.

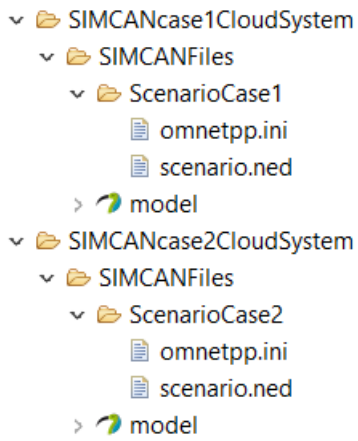


**Figura 23. Ficheros de configuración**

Al ejecutar el simulador se obtiene un fichero de resultados que contiene, para cada usuario, un identificador y el tiempo transcurrido desde que llega al sistema y realiza la petición hasta que termina la ejecución de la aplicación.

A partir de dichos ficheros, se pueden generar gráficas utilizando la utilidad Gnuplot [21]. Para este caso hemos simulado dos sistemas con el mismo tipo de nodos, uno con 64 nodos (ver Figura 24) y otro con 128 nodos (ver Figura 25), a los que llegan 1000 usuarios realizando la misma petición.

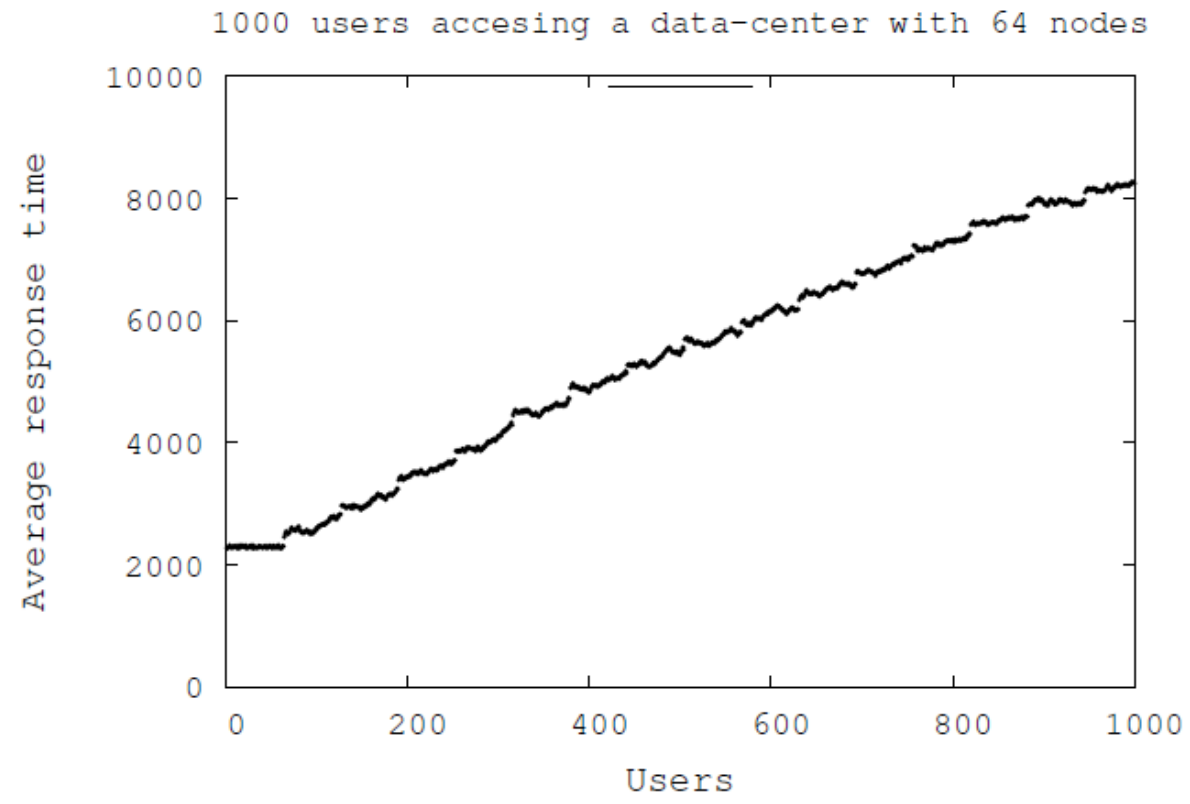


**Figura 24. Resultado del caso de estudio 1**

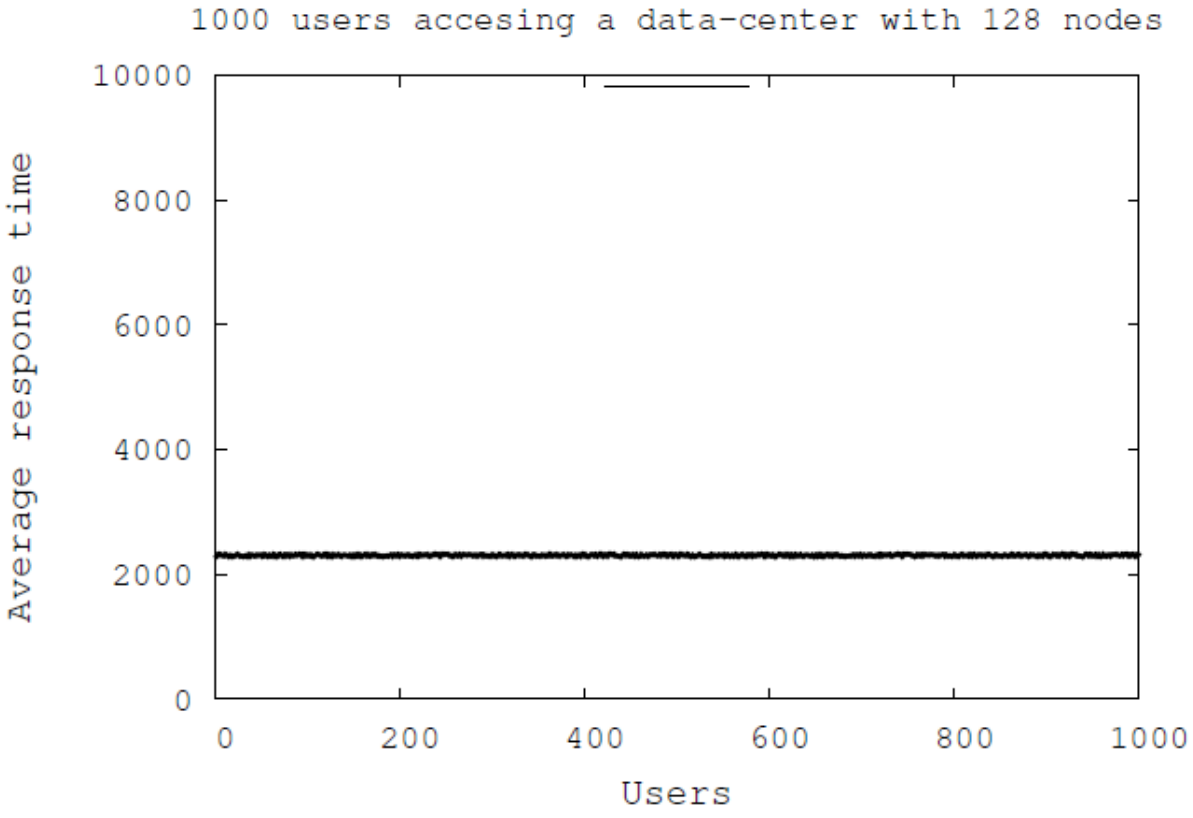


**Figura 25. Resultado del caso de estudio 2**

En las gráficas podemos observar como en el caso de estudio 1 los primeros usuarios que realizan peticiones al sistema cloud son atendidos rápidamente y ejecutan la aplicación en el mismo tiempo, pero, al ser un cloud relativamente pequeño, cuando van llegando más usuarios se encuentran con los recursos ocupados y necesitan esperar. Como el ritmo al que llegan los usuarios es más alto que el de liberación de los recursos, nos encontramos con un cuello de botella, resultando inviable atender a todos los usuarios si éstos siguen realizando peticiones a ese ritmo.

## Discusión:

Durante el inicio de este trabajo se planteaba si era viable la aplicación de MDA al modelado y simulación de infraestructuras cloud y de la interacción de los usuarios con estas infraestructuras. Aplicar esta tecnología a la simulación cloud nos permitiría elevar el nivel de abstracción y aplanar, de esta manera, la curva de aprendizaje del simulador.

Para comprobar que efectivamente facilita la tarea de modelado y simulación, se ha realizado un test de usabilidad siguiendo el estándar ISO/IEC 25062:2006 [22] en el que cada participante ha modelado tres sistemas cloud.

Debido a que esta herramienta estará orientada principalmente al uso académico, los participantes en el estudio han sido estudiantes del grado de Ingeniería Informática. Se ha diferenciado entre estudiantes de la rama de Ingeniería del Software (IS) y estudiantes de Tecnologías de la Información (TI), ya que los primeros tienen más contacto con el modelado y sus herramientas.

Hubo 4 variables dependientes: tasa de finalización de tareas sin asistencia, tiempo para completar la tarea, número de errores y número de asistencias. También se administró un cuestionario subjetivo al final para obtener datos cualitativos exploratorios, cada participante cumplimentó un cuestionario System Usability Scale

(SUS) con escala Likert de 5 puntos.

El resumen de los datos de la realización de las tareas se muestra en la Tabla 1 y la Figura 26, y el resumen del cuestionarios SUS se muestra en la Figura 27.

**Tabla 1. Resumen de los resultados de la realización de las tareas**

| Tipo de usuario | Tasa de finalización sin asistencia | Tasa de finalización con asistencia | Tiempo total de la tarea (h:m:s) | Errores | Asistencias |
|---|---|---|---|---|---|
| Alumnos Ingeniería del software | 0,5 | 0,5 | 00:39:05 | 0,5 | 2 |
| Alumnos Tecnologías de la infomración | 0,166 | 0.83 | 00:56:08 | 2.5 | 4 |
| Media | 0,33 | 0,66 | 00:47:36 | 1,5 | 3 |
| Mínimo | 0 | 0.33 | 00:28:30 | 0 | 1 |
| Máximo | 0,66 | 1 | 00:58:15 | 3 | 5 |

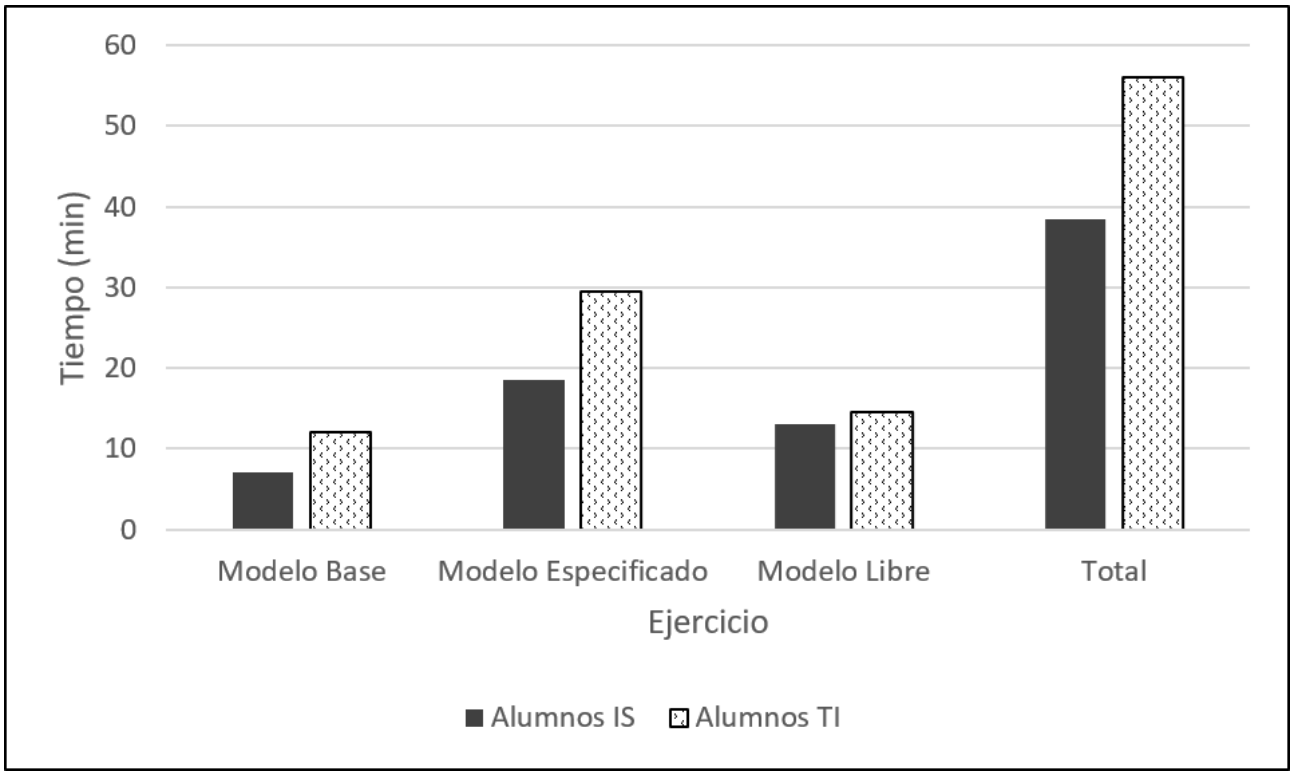


**Figura 26. Tiempo medio para cada tarea por tipo de usuario**

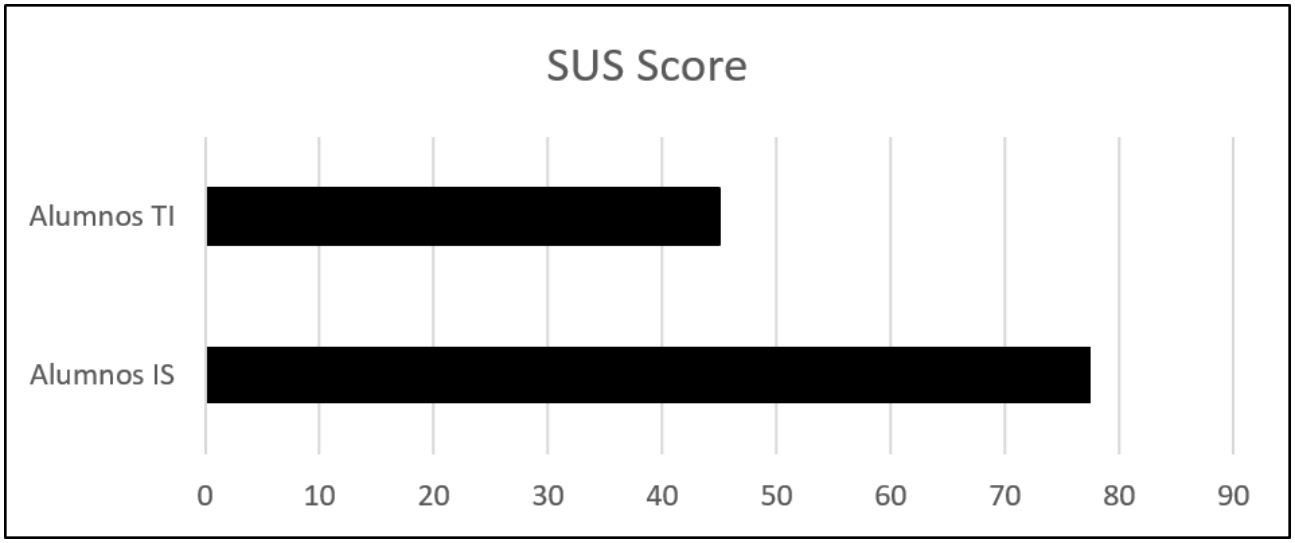


**Figura 27. Puntuación SUS por tipo de usuario**

Claramente el uso de la herramienta es más sencillo para los alumnos de la rama de Ingeniería del Software, debido a que tienen más conocimientos de modelado, pero se pudo observar como las mayores limitaciones venían dadas por el desconocimiento de la plataforma Eclipse y Papyrus.

La mayoría de las asistencias realizadas fueron para resolver dudas en el funcionamiento general de Papyrus, y que, en caso de no disponer de esa asistencia, podrían haber sido resueltas por la documentación disponible sobre la plataforma.

En cualquier caso, todos los alumnos terminan las tareas de modelar tres sistemas cloud en menos de una hora sin ninguna formación previa sobre el simulador, por lo que se valora que la aplicación MDA a la simulación de sistemas cloud, mediante el framework desarrollado, facilita su diseño y estudio.

## Conclusiones:

Durante la elaboración de este proyecto se ha colaborado con el grupo de investigación de testing y evolución del rendimiento de la Universidad Complutense de Madrid, que está desarrollando *Simcan2Cloud*. En esta colaboración se ha ayudado en la aplicación de los estándares de modelado al simulador.

El problema que existía era la dificultad que tenían los alumnos para observar el rendimiento de los sistemas de computación en la nube modelados durante su formación.

Por ello, el objetivo principal que se quería conseguir era acercar la parte práctica a la teórica permitiendo a los alumnos modelar un sistema cloud del que pueda obtener datos sobre su rendimiento. De esta manera se podrían estudiar modelos y contrastar los resultados obtenidos.

Para permitir al usuario modelar el sistema, se ha definido un perfil de UML con el que se pueden modelar todos los componentes del simulador. La ventaja de usar un perfil de UML es que se pueden modelar otros componentes del sistema cloud que se necesiten plasmar en el modelo para capturar más información, pero que no sean relevantes para el simulador.

Pero, puede que, en otros casos, las clases fueran directamente a la práctica debido a que el modelado podía resultar algo pesado, y más si esos modelos solo servían para comunicar información posteriormente. El framework facilita esta tarea de modelado y da utilidad al modelo generando los ficheros de configuración del simulador mediante la arquitectura dirigida por modelos. De esta manera anima a modelar los sistemas cloud antes de simularlos, ya que así no será necesario configurar el simulador manualmente.

En definitiva, el objetivo principal se ha visto cumplido y se espera que el framework desarrollado ayude a los alumnos a comprender mejor los sistemas distribuidos y su funcionamiento.

El simulador *Simcan2Cloud* es una herramienta en desarrollo que, actualmente, está continuamente creciendo y mejorando, así este framework debe crecer en el futuro. El equipo de desarrollo de *Simcan2Cloud* pretende ir ampliando los protocolos de comunicación, creando nuevos y añadiendo parámetros. El framework deberá ir madurando a la vez que el simulador, permitiendo modelar estos nuevos protocolos. Esto implica la extensión del perfil UML y en esta extensión se pueden

incluir, además, la posibilidad de solicitar más de una máquina virtual y más de una aplicación por usuario, así como el control de precios y franjas horarias para la toma de decisiones de los usuarios.

**Referencias:**

[1] A. Núñez, J. Fernández, R. Filgueira, F. Garcá, and J. Carretero, "SIMCAN: A flexible, scalable and expandable simulation platform for modelling and simulating distributed architectures and applications," *Simulation Modelling Practice and Theory*, vol. 20, no. 1, pp. 12–32, 2012.

[2] ORMSC, "A Definition of MDA," http://users.dsic.upv.es/workshops/dsdm04/-files/MDA-ormsc-040802.pdf/, 2004, accessed: 2016-11-20.

[3] I. Dietrich, *Syntony: A Framework for UML-Based Simulation, Analysis, and Test with Applications in Wireless Networks*. Verlag Dr. Hut, August 2010, accesed: 2017-01-10. [Online]. Available: http://www.dr.hut-verlag.de/9783868535655.html

[4] OMG, "What is UML," http://www.uml.org/what-is-uml.htm, 2005, accessed: 2016-11-15.

[5] A. Djanatliev, W. Dulz, R. German, and V. Schneider, "Veritas - a Versatile Modeling Environment for Test-driven Agile Simulation," in *Proceedings of the Winter Simulation Conference*. Winter Simulation Conference, 2011, pp. 3662–3671, accesed: 2017-01-09. [Online]. Available: http://dl.acm.org/-citation.cfm?id=2431518.2431953

[6] A. Bergmayr, J. Troya, P. Neubauer, M. Wimmer, and G. Kappel, "Uml-based cloud application modeling with libraries, profiles, and templates." in *CloudMDE@ MoDELS*, 2014, pp. 56–65.

[7] A. Kamali, S. Mohammadi, and A. A. Barforoush, "Ucc: Uml profile to cloud computing modeling: using stereotypes and tag values," in *Telecommunications (IST), 2014 7th International Symposium on*. IEEE, 2014, pp. 689–694.

[8] D. C. Schmidt, “Guest Editor’s Introduction: Model-Driven Engineering,” *Computer*, vol. 39, no. 2, pp. 25–31, Feb. 2006, accesed: 2017-01-25. [Online]. Available: http://dx.doi.org/10.1109/MC.2006.58

[9] OMG, “Welcome to OCUP 2 - OMG’s UML 2.5 Certification!” http://-www.omg.org/ocup-2/, 2016, accessed: 2016-11-20.

[10] uml-diagrams.org, “UML 2.5 Diagrams Overview,” http://www.uml-diagrams.org/uml-25-diagrams.html, 2016, accessed: 2016-11-01.

[11] L. Fuentes and A. Vallecillo, “Una introducción a los perfiles UML,” *Novática*, vol. 168, pp. 6–11, 2004.

[12] S. J. Mellor, S. Kendall, A. Uhl, and D. Weise, *MDA Distilled: Principles of Model-driven Architecture*. Addison Wesley Longman Publishing Co., Inc., 2004.

[13] Á. Jiménez Rielo, “Incorporando la Gestión de la Trazabilidad en un entorno de Desarrollo de Transformaciones de Modelos Dirigido por Modelos,” Master’s thesis, Universidad Rey Juan Carlos, March 2012.

[14] Eclipse Foundation, Inc., “Papyrus,” http://www.eclipse.org/papyrus/-download.html, 2016, accessed: 2016-11-01.

[15] OMG, “OMG Object Constraint Language (OMG OCL) Version 2.4,” http://-www.omg.org/spec/OCL/2.4, 2014, accessed: 2016-11-13.

[16] I. Kurtev, *State of the Art of QVT: A Model Transformation Language Standard*. Springer Berlin Heidelberg, 2008, pp. 377–393, accesed: 2016-12-03. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-89020-1_26

[17] OMG, “OMG MOF Model to Text Transformation Language (OMG MOFM2T) Version 1.0,” http://www. omg. org/spec/MOFM2T/1.0, 2008, accessed: 2016-11-08.

[18] Eclipse Foundation, Inc., “Acceleo,” https://www.eclipse.org/acceleo/, 2016, accessed: 2016-11-01.

[19] H. López, F. Varesi, M. Viñolo, D. Calegari, and C. Luna, “Estado del arte de verificación de transformación de modelos,” *Reportes Técnicos 10-07*, 2010.

[20] A. Tiso, G. Reggio, and M. Leotta, “Unit Testing of Model to Text Transformations.” in *AMT@ MoDELS*, 2014, pp. 14–23.

[21] gnuplot, “Gnuplot homepage,” http://www.gnuplot.info/, 2017, accessed: 2017-06-12.

[22] ISO/IEC 25062:2006, *Software engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Common Industry Format (CIF) for usability test reports*, International Organization for Standardization/International Electrotechnical Commission Std., Oct. 2008.