Noname manuscript No.
(will be inserted by the editor)

Improving Cloud Architectures using UML Profiles and M2T Transformation Techniques

Adrián Bernal · M. Emilia Cambronero · Alberto Núñez · Pablo C. Cañizares · Valentín Valero

Received: date / Accepted: date

Abstract In this paper, we present an approach with the goal to improve the underlying architecture of cloud systems. For this, we propose `UML2Cloud`, a framework targeted at modeling and checking cloud systems. The main core of `UML2Cloud` uses UML profiles to capture the main elements of a cloud system including, among other elements, its underlying architecture and the interaction with clients. Additionally, `UML2Cloud` uses Model to Text (M2T) transformation techniques to automatically generate configuration documents representing complex cloud scenarios.

In this work, we use these documents as input for a cloud simulation tool, called Simcan2Cloud, to simulate the behavior of different systems. Thus, the analysis of the performance results obtained from the simulations allows us to draw some conclusions about how to improve the efficiency of the studied clouds by adjusting the hardware resource configuration.

Keywords Modeling · *UML Profile* · Cloud · MDA · Performance Analysis

A. Bernal, M.E. Cambronero, V. Valero
Albacete Research Institute of Informatics
E-mail: adrian.bernal@uclm.es, memilia.cambronero@uclm.es, valentin.valero@uclm.es

P. C. Cañizares, A. Núñez
School of Informatics
E-mail: pablocc@ucm.es, alberto.nunez@pdi.ucm.es

1 Introduction

Cloud computing provides technology for users and enterprises with ubiquitous, flexible and on-demand access over the Internet to a shared pool of configurable computing resources, including servers, databases, software applications, storage capacity and computing power [18]. Cloud computing services have two main mandatory characteristics: *on-demand self-service* and *elasticity of provision*. On-demand self-service refers to the use of cloud services by end users according to their computing requirements, and elasticity of provision to the capabilities for cloud scalability in response to changes in the number of users or required services in general.

Cloud computing is a strategic technology and many barriers must yet be overcome for its widespread use in enterprises. In the European Union, according to Eurostat, 21% of enterprises reported the use of cloud services in 2016, mainly e-mail and storage services. Only 11% of them reported the use of advanced services (financial applications, CRMs, computing power). Compared with 2014, this represents an increase of about 7% in the use of cloud computing services.

During the last years, cloud optimization has become a hot topic in the scientific community [32, 33, 49]. In this paper, we present an approach for improving cloud architectures using modeling and simulation techniques. For modeling purposes, a UML profile [41] is proposed. Modeling is a proven and well-accepted engineering technique, and to help users to visualize the final product, we usually build models.

UML (Unified Modeling Language [42]) is a well-known modeling language which includes a complete set of tools for the design of real-time systems. We focus on component and sequence diagrams, which allow us to model the structural relationships between the system components and the dependencies among them, the flow of actions, and the interactions between different roles in the system, respectively.

To our knowledge, there are no previous works modeling the cloud client interactions with a cloud simulator using UML interaction diagrams. As shown in the related work section, most works focus on modeling the hardware characteristics of cloud systems and then perform simulations using specific workloads. We use UML structure and interaction diagrams to model the entire cloud system since they allow us to model the underlying cloud infrastructure, as well as the cloud resource allocation to users, and finally, to analyze the system performance. Furthermore, UML profiling techniques allow us to define cloud-specific elements using stereotypes. We also check the correctness of the defined cloud models, which ensures that these models faithfully represent the cloud system under study.

Setting up large cloud deployments is a laborious and complex process. A common approach used to alleviate these issues is to use simulators. However, these simulators also need to be configured in their language. This task is facilitated by using the proposed modeling techniques to create, visualize, edit and export cloud system configurations. Specifically, the defined UML profile cap-

tures the user interactions with a cloud provider, as well as the main features of the underlying cloud infrastructure. This UML profile stages the interactions between users and cloud providers, using subscriptions to be notified when the required cloud infrastructure is available. Thus, with the proposed and implemented framework, we can create models for specific user behaviors and cloud infrastructure characteristics. Then, M2T (Model To Text) transformation techniques are used to automatically generate the configuration files for a cloud simulator. Our framework has been designed to support different cloud simulators. At present, it provides support for a preliminary version of Simcan2Cloud, an open-source simulation tool targeted at modeling and simulating cloud environments, which is currently under development. The main strength of this simulator lies in its flexibility and scalability. In addition, each system can be modeled by fully customizing the data-centers that support the cloud. In general, this configuration includes the size of each data-center, the main features of each physical and virtual machine, and the communication networks. Furthermore, users can include new policies and algorithms for representing key aspects of cloud systems, such as the behavior of the clients who access the cloud and resource allocation policies, among others.

The paper is structured as follows. The background of Model-Driven Architecture (MDA) is described in Section 2. The UML profile and the tool that we have implemented to create and edit cloud system models are presented in Section 3. The cloud simulator we use, namely Simcan2Cloud, is introduced in Section 4, and the transformation tool to obtain the input files for Simcan2Cloud from the cloud system models is described in Section 5.

Several experiments that illustrate the complete methodology are presented in Section 6, including performance and scalability analysis and the execution of real traces over a modeled cloud architecture. Section 7 covers the related work, and finally, Section 8 contains the conclusions and future lines of work.

2 Modeling and Meta-modeling

Model-Driven Engineering (MDE) [46] is a software development methodology for creating and exploiting domain models effectively. In MDE, models are used to accomplish the phases of software engineering, so it provides a larger process definition than Model-Driven Development (MDD) [26], which only focuses on development. In MDD we model a system that is transformed into the real thing, such as code, documentation or reports, among others. Model-Driven Architecture (MDA) [39] is an Object Management Group (OMG) initiative that proposes a set of standards that specify interoperable technologies for implementing MDD.

The evolution towards MDE has brought with it new concepts such as *Model*, *Metamodel* and *Model transformation*. A *software model* is a description of a system written in a well-defined language, with specific syntax and semantics, which can be interpreted by a computer [22]. A *meta-model* is a model that defines the structure of a modeling language. If a model is an

abstraction of the real world, a meta-model is an abstraction of a model. Depending on the system goals we can use different model artifacts, so *model transformation* techniques are used to automatically transform models that have been created by using a specific technology into equivalent models that are based on a different technology.

Mens and Van Gorp [28] have established a taxonomy of model transformation techniques depending on the transformation target, distinguishing between *endogenous* and *exogenous* transformations. Endogenous transformations are performed between models expressed in the same language, while exogenous transformations are performed between models expressed using different languages. Thus, in *Model-to-Model* transformations (M2M) both source and target are models whereas, in *Model-to-Text* transformations (M2T), text artifacts (e.g. source code, reports, configurations, documentation, etc.) are obtained from models.

Model transformations are performed automatically by using several standards provided by OMG, such as QVT [43] (for M2M transformations) and MOFM2T [37] (for M2T transformations). In this paper, we use Acceleo [36], an open-source code generator from the Eclipse Foundation based on the MOFM2T (MOF Model to Text) standard to transform the UML stereotyped models into text documents specifying the simulator configuration and the client interactions. Acceleo uses a template-based approach, in which a parameterized *template* with the model elements specifies how the text must be generated. These templates specify the text structure with position markers for including the data extracted from the models. These markers are essentially queries which select and extract model values.

In general, we complete the visual descriptions of the systems by defining certain constraints that the model elements must fulfill. These constraints can be later validated by using OCL (Object Constraint Language) [40], which is a standardized language defined by OMG that allows us to define these constraints and make queries to obtain values from the model.

3 UML Cloud Profile Framework

In this section, we present an overview of the `UML2Cloud` profile and the tool that we have implemented to create and edit cloud system models. This UML profile allows us to have an accurate representation of both the cloud infrastructure and the interactions of the clients with the cloud provider to request the services they need. A complete version of the `UML2Cloud` profile can be found in a previous work [5].

3.1 `UML2Cloud` Profile

We consider that a cloud system (see Figure 1) consists of a cloud provider, one or more data centers, and clients (also called cloud users) requesting resources

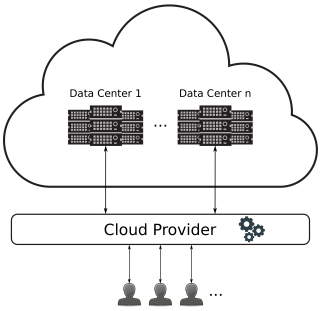


Fig. 1 Underlying architecture of a cloud system.

from the cloud. The cloud provider manages a catalog of Virtual Machines (VMs) and the hardware resources provided by the data-centers. Each data center consists of a collection of physical machines, also called nodes, which are grouped in racks. Thus, each rack contains a set of machines with the same hardware features, that is, CPU, memory and, storage. Based on this infrastructure, the cloud provider offers a catalog of VMs that clients request according to their computational requirements. Hence, the VMs required are mapped to physical machines, using a resource allocation policy, on which they are executed.

It is important to remark that our proposed profile is not focused on a specific simulation tool. On the contrary, we provide a general view of the underlying behavior of cloud infrastructures.

At present, there exists a wide variety of cloud simulators that can be used both to model and simulate cloud environments [9, 19]. Some of the current well-known cloud simulators are cloudSim [8], simGrid [12] and GreenCloud [23]. However, we have chosen Simcan2Cloud due to the following reasons: i) Simcan2Cloud has been developed using SIMCAN as a basis, which is a mature simulator that has been currently validated against real HPC architectures [35]; ii) Simcan2Cloud provides an intuitive graphical interface (GUI) that provides a high level of flexibility to model a wide spectrum of cloud scenarios. In contrast, these simulators require to manually configure the modeled scenarios, which is a complex and error-prone task; iii) an intuitive API is provided to include new functionality into Simcan2Cloud, which allows adding new user profiles to represent the behavior of different types of users and new resource allocation policies; iv) we know Simcan2Cloud in-depth and, therefore, we can include the required profiles accurately into the simulation platform.

Since many components of a data center are usually the same, and to facilitate the reuse of these components, we have defined the relationships between components as stereotype associations (see Figure 2). In this figure, we can see that the cloud infrastructure consists of a set of data centers. Each data center consists of a set of racks, which in turn consist of a set of machines (computing and storage). As we can have several data centers with the same

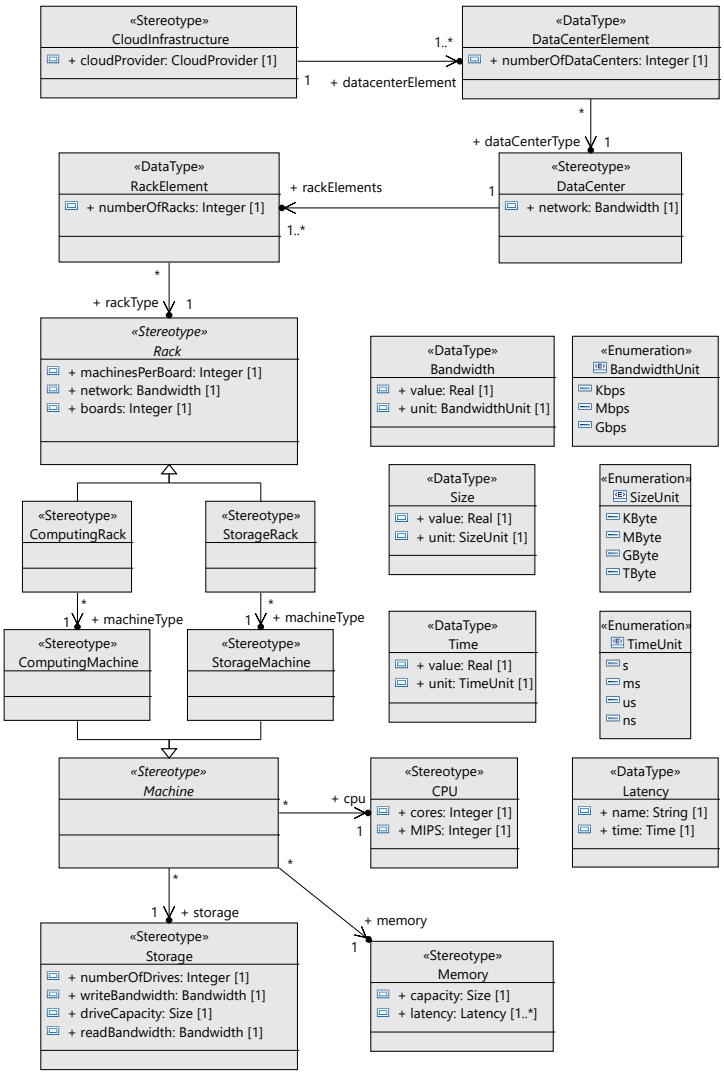


Fig. 2 `UML2Cloud` profile. Properties and associations of cloud infrastructure stereotypes.

configuration, the *DataCenterElement* data type has been defined to represent a set of data centers that have the same configuration. In the same way, the *RackElement* data type has been defined to represent a set of racks with the same configuration. As an illustration, in the rack definition, we must indicate the number of boards, the number of machines per board and the network bandwidth for the communication between machines.

The interactions between the users and the cloud provider are captured in terms of the messages they exchange. Thus, a Sequence Diagram (SD) is used to establish this relationship (see Figure 3). In this sequence diagram,

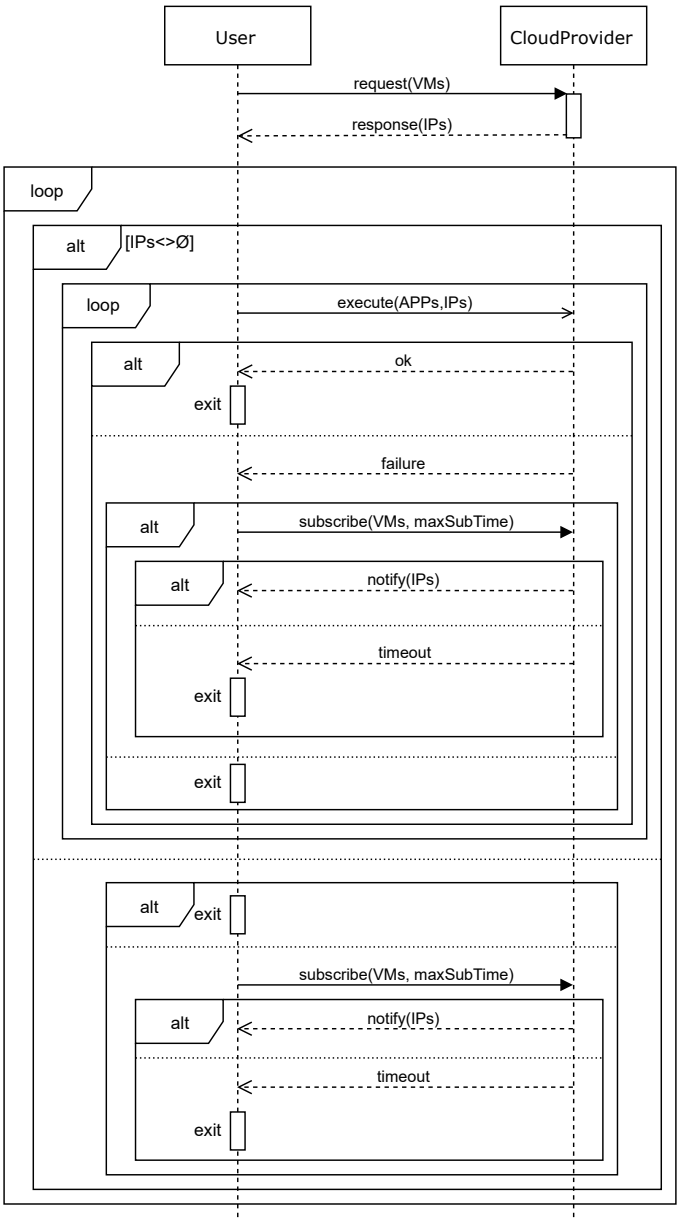


Fig. 3 Cloud provider and user interaction SD.

the user requests a set of VMs from the cloud provider, where VMs is a set of VM tuples, which are defined as follows:

$$(number\ of\ instances,\ VMtype,\ renting\ time)$$

Then, the cloud provider responds with a set of IPs. These are the IPs of the machines that can possibly attend to the user demands. However, the cloud provider can also return an empty set of IPs, which means that it cannot currently satisfy the request. In the first case, if the cloud provider returns a non-empty set of IPs, the user sends an *execute* message to the cloud provider to start the executions of the selected *Apps* on the indicated machines. Two cases can then arise:

- the execution of all these *Apps* terminates in time, so an *ok* message is returned to the user, which in turn terminates its execution;
- a *failure* message is returned from the *cloud provider*, either because the execution runs out of time *(rentingTime)*, or due to a failure.

When a *failure* message is received, the user can decide to abort, but they can also decide to submit a subscription to the cloud service to be notified when some machines are available to satisfy the request. This subscription has a timeout associated *(maxSubTime)*, which means that it is only valid for that period. Upon receiving the notification, the user sends again the *execute* message to the cloud provider; otherwise, when the subscription time-out expires, the user aborts. Finally, the initial request could be answered with an empty response, and the user can then again decide either to abort or submit a subscription (lower part of first *alt* operation in SD). Note the first *loop* in the SD to send an *execute* message in the case of being notified by the cloud provider with the IPs of the corresponding machines.

The relationships between the components are shown in Figure 4, as stereotype associations. As an illustration, a *VM* request consists of the following elements: number of cores, computing units (CUs) for the CPU cores, disk size and memory size.

In addition to the implicit constraints derived from the stereotype relationships, such as the one-to-one or one-to-many relationships, the model must meet some other constraints to ensure its correctness. Thus, we have defined and checked 34 constraints in order to validate our model. These constraints have been defined using OCL rules, see [5] for the details.

3.2 UML Cloud Profile Tool

A tool has been developed to create and edit cloud scenarios, and this tool is available on an Ubuntu virtual machine at the URL `http://antares.sip.ucm.es/cana/umlCloud/`, where instructions and a demo video have also been included.

This tool is an extension of the *Model4Cloud* tool [5]. In this new version of the tool, the main contribution is the addition of a plug-in for the transformations of the UML cloud models to configuration files of a cloud simulator. The tool has been implemented as an Eclipse feature by extending the Papyrus UML tool.

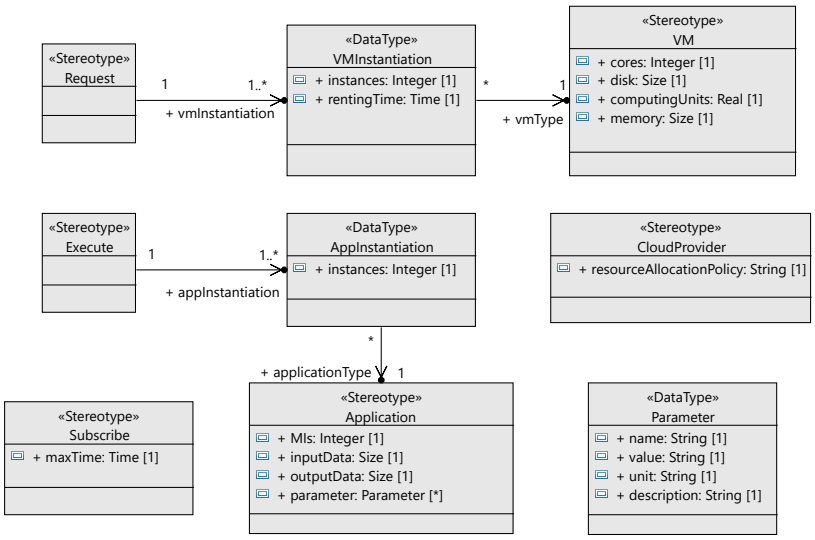


Fig. 4 `UML2Cloud` profile. Associations and properties of cloud interaction stereotypes.

Thus, we have defined and implemented the `UML2Cloud` profile, as well as the diagram styles, property views, palettes and menus that provide the graphical view of the framework (see Figure 5). The *diagram style* (see 1 in Figure 5) customizes the diagram graphical representation through Cascade Style Sheets (CSS) to show the stereotype properties directly in the diagram. The *property view* (see 2 in Figure 5) allows users a quick access to the stereotype properties to change their values. The *palette* (see 3 in Figure 5) allows users to easily create the components needed to model the cloud system, with the stereotypes already applied.

Transformations of UML cloud models to configuration files of a cloud simulator are implemented using M2T. Thus, we can automatically produce simulations of the cloud system modeled and obtain quantitative results from these simulations. In part 4 of Figure 5, we can see the *generated files* section, in which the two files that are generated by the transformation are shown, namely *omnet.ini* and *scenario.ned*.

These transformations are implemented by using several plug-ins (see Figure 6). The new plug-ins implementing these transformations are marked in red in the figure. This tool allows us to include transformation plug-ins for different cloud simulators, so <system> denotes the specific cloud environment used for the deployment. A description of all the plug-ins implemented follows:

- **es.uclm.uml2cloud.profile**: This plug-in contains the UML profile defined to model cloud systems. It also registers the profile, so the end-users will be able to use the profile easily.
- **es.uclm.uml2cloud.validation**: This plug-in contains the OCL constraints needed to validate the cloud system models.

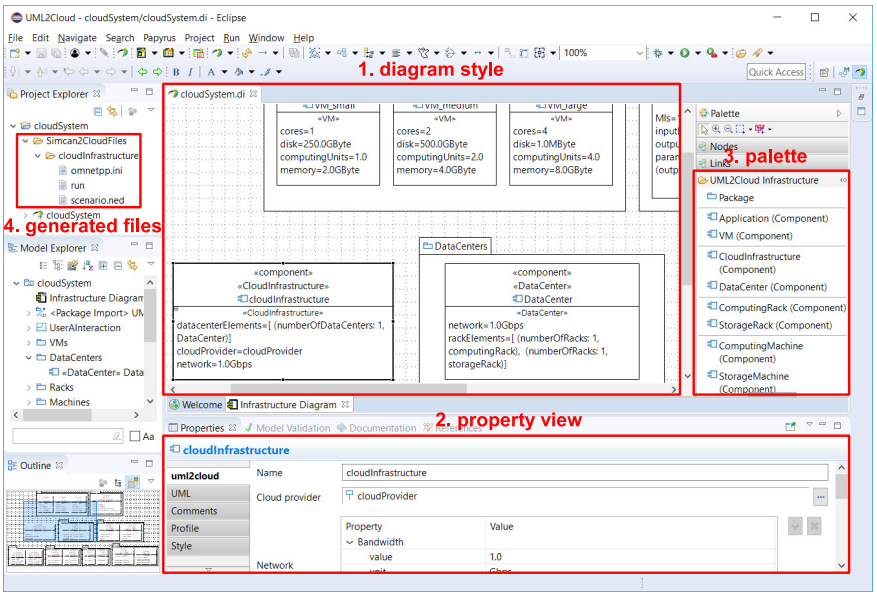


Fig. 5 Papyrus customization.

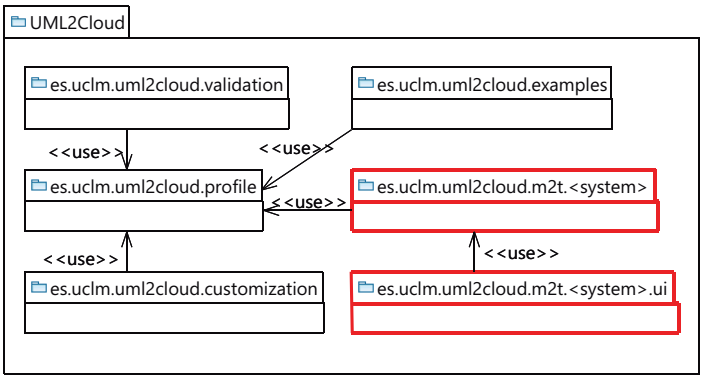


Fig. 6 Tool package diagram.

- **es.uclm.uml2cloud.customization**: This plug-in extends the Papyrus graphical editor to customize it in order to facilitate the use of the defined profile. This includes the customization of the tool palette, the properties view and the graphical appearance of the UML elements.
- **es.uclm.uml2cloud.examples**: This plug-in contains several examples of cloud systems modeled with the profile. These examples could then be used as a starting point for the modeling of a specific scenario instead of starting from scratch.
- **es.uclm.uml2cloud.m2t.<system>**: This plug-in implements the automatic transformation from cloud systems modeled with the UML cloud

profile to the configuration files of the target system. This transformation is implemented with Acceleo, which is an implementation of the MOFM2T standard. The plug-in generates the files corresponding to the modeled scenario. These files can be either the configuration files for a simulator or a real deployment environment. This transformation is generic, but the configuration files produced are specific to the cloud simulator used or to the real environment. This plug-in must, therefore, be implemented for each cloud simulator supported by our framework, so the word *system* denotes the specific simulator on which the transformation is applied. We currently provide support for the transformation to the Simcan2Cloud tool, so a plug-in called *es.uclm.uml2cloud.m2t.simcan2cloud* has been implemented.
- **es.uclm.uml2cloud.m2t.<system>.ui**: This plug-in implements the contextual menu that allows the users to easily execute the transformations. It also depends on the cloud simulator used, so it must be implemented for each cloud simulator supported by our framework.

4 The Simcan2Cloud simulator

The simulation of cloud systems, which usually involves a very large number of heterogeneous resources and millions of user requests, requires precise, efficient and flexible modeling techniques. The cloud system under study must be precisely modeled to accurately represent its behavior. It is then necessary that simulation tools provide enough flexibility to model a wide range of cloud configurations.

In order to fulfill these requirements, we have chosen Simcan2Cloud, a simulation platform based on SIMCAN [35] that is currently under development [1].

This tool allows us modeling and simulating cloud computing infrastructures, in which some parts of the simulation engine, such as the functionalities for sending messages along a path of modules and managing requests, have been inherited from SIMCAN. However, the representation of some hardware components, such as computing nodes, storage nodes, and data-centers have been adapted to be used in Simcan2Cloud, while the underlying infrastructure and functionality of the cloud have been designed and implemented from scratch.

Simcan2Cloud has been written in C++ using OMNeT++ [52], a simulation framework based on discrete event simulation (DES), which has become very popular in the research community over the last few years. Some of the most well-known simulators for modeling distributed systems, such as INET [48], OverSim [2] and RINAsim [53], to name just a few, use OMNeT++ as a basis.

The underlying infrastructure of Simcan2Cloud is depicted in Figure 7. Simcan2Cloud has been designed to provide a high level of flexibility. Thus,

[1] Simcan2Cloud is being developed by the DTRS (Design and Testing Reliable Systems) research group of the Complutense University of Madrid and supervised by Dr. Alberto Núñez

its architectural design can be divided into three main parts that can be individually modeled to generate a simulated cloud scenario. The generation of users represents the workload to be processed by the cloud, that is, the users that access the cloud in the simulated scenario. This part plays a very important role in the cloud environment [54]. In essence, each user rents one or several VMs for a specified time frame, where one or several services, previously requested by the user, are executed. The cloud provider contains data structures and algorithms for managing user requests and allocating resources. Finally, the data-centers represent the hardware part of the cloud. These contain the models of the physical machines where the VMs requested by the users are deployed.

The following sections provide a detailed description of the underlying design of Simcan2Cloud and the input required to generate a simulation scenario.

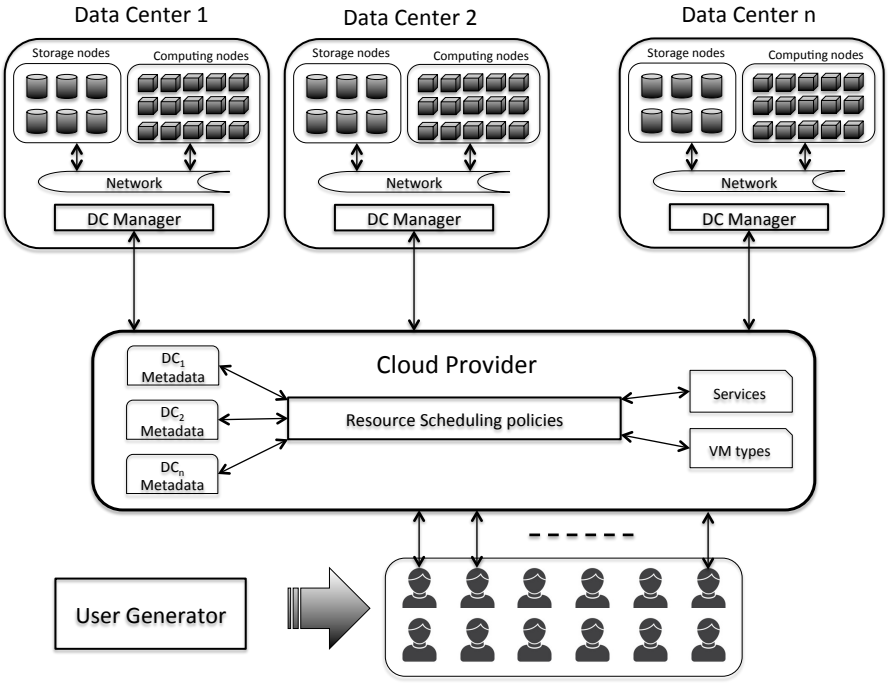


Fig. 7 General architecture of Simcan2Cloud.

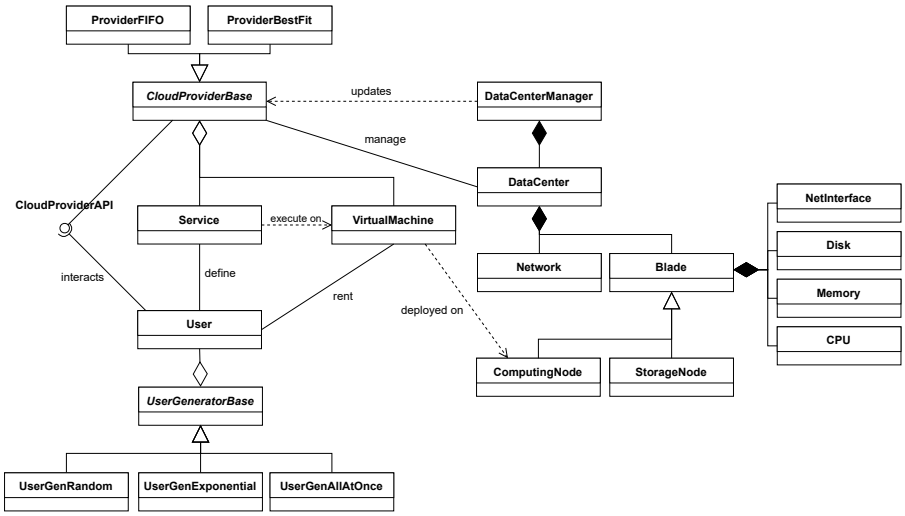


Fig. 8 Simcan2Cloud class design diagram.

4.1 Class Design Diagram of Simcan2Cloud

A simplified view of the class design diagram of Simcan2Cloud is given in Figure 8. For the sake of simplicity, only the most relevant classes are shown in this diagram.

This class diagram represents the underlying infrastructure of a complete cloud system. The main objective of this design is two-fold. Firstly, it must provide mechanisms for modeling a wide spectrum of cloud scenarios. Since each part of the cloud can be individually modeled by re-using existing models, researchers are able to combine these models from a repository, which considerably increases the total number of possible configurations to simulate a cloud scenario. For instance, it is possible to combine different algorithms to generate users with several resource allocation policies used by the cloud provider. Secondly, it must allow the inclusion of new components to increase the scalability of Simcan2Cloud, such as adding policies for managing user requests in the cloud provider and adding new hardware models representing the behavior of different devices.

To accomplish this objective, each part of the system provides basic functionality that must be used for each new component that is added to the simulator. In general, these functionalities are included in the base classes, such as *CloudProviderBase* and *UserGeneratorBase*. For instance, 3 different algorithms for generating the workload are included in different classes, namely *UserGenRandom*, *UserGenExponential*, and *UserGenAllAtOnce*, which inherit from *UserGeneratorBase*.

4.1.1 Generation of Users

In Simcan2Cloud, the workload is represented by a collection of users accessing the cloud. The main classes involved in the generation of users are as follows:

User: Each *User* object represents the behavior of a user in a real cloud. Basically, a user is modeled by defining the rented VMs and the services that are executed on them. Each *VirtualMachine* is modeled by setting its hardware features (CPU, memory, and disk). Services can be configured by setting the required parameters to model a specific behavior. The current version of Simcan2Cloud provides several services, such as web servers and data-intensive applications for processing large amounts of data, among others.

UserGeneratorBase: This abstract class contains references to the services and VMs offered by the cloud. This class also provides some facilities for creating user instances. Initially, this class parses the configuration of the workload, that is, each modeled user involved in the simulated cloud scenario and the number of user instances to be created. It is important to mention that this process is performed before the Simcan2Cloud starts to execute the simulation and, therefore, all the user instances are created when the simulation time is still zero. The algorithm that orchestrates how the users access the cloud must be located in a class that inherits from *UserGeneratorBase*. We can see in Figure 8 three classes focusing on this objective: *UserGenExponential*, which generates the workload by using an exponential distribution function, *UserGenRandom*, which randomly generates the time when each user accesses the system, and *UserGenAllAtOnce*, which makes users access the system all at once, at the beginning of the simulation.

4.1.2 Cloud Provider

In general terms, the main tasks of a cloud provider consist of attending to user requests, locating the resources requested by the user among the associated data-centers, and generating an answer for these requests. Additionally, the cloud provider creates the illusion of managing infinite resources by hiding deployment details, even if the cloud owns several data-centers that are located in different geographical locations. The classes involved in serving users are as follows:

CloudProviderBase: This abstract class contains the main functionalities for managing user requests, such as searching for the available resources in a data-center and deploying a given *Virtual Machine* on a *ComputingNode* located in a *DataCenter*. Moreover, this class accesses both the data structures containing information about each data-center and the structures containing each user request. Resource allocating policies must be implemented in a class that inherits from *CloudProviderBase*, such as *ProviderFIFO*, which attends to user requests by using a FIFO policy, and *ProviderBestFit*, which uses the "best fit" algorithm for deploying VMs on the available resources. To accurately represent the behavior of a cloud system, the cloud provider exports an API that acts as the interface between the users and the cloud.

```
1 void handleUserAppRequest(SM_UserAPP* userAPP_Rq);
2 void acceptAppRequest(SM_UserAPP* userAPP_Rq);
3 void rejectVmRequest(SM_UserVM* userVM_Rq);
4 void notifySubscription(SM_UserVM* userVM_Rq);
5 void timeoutSubscription(SM_UserVM* userVM_Rq);
6 void freeUserVms(std::string strUsername);
7 int getTotalCoresByVmType(std::string strVmType);
```

Listing 1 Excerpt from the API exported by the Cloud Provider.

CloudProviderAPI: This interface exports an API containing a list of operations that can be invoked by the users to communicate with the cloud. Listing 1 shows an excerpt from the API offered by the cloud provider. It is important to mention that this API contains a large number of methods and, for the sake of clarity, we only show the most representative ones in this listing.

4.1.3 Data Centers

Broadly speaking, a *DataCenter* represents the physical resources managed by the cloud system. The underlying infrastructure of Simcan2Cloud groups these resources into three categories: computing nodes, storage nodes, and communication networks. The main classes for managing a data-center are as follows:

DataCenterManager: This class is in charge of different tasks focused on managing the available resources, such as maintaining a list of the idle CPUs and updating the amount of free memory on each physical machine, among others.

ComputingNode: This class represents a physical machine where the VMs requested by users are deployed. A computing node can be customized by configuring the CPU system, the memory system, and the disk.

StorageNodes: This class represents a physical machine that manages large amounts of data. In general, these machines do not deploy the VMs requested by users, but store data that are accessed by them.

Network: This class represents a network for interconnecting the nodes of the data center.

4.2 Input for Simcan2Cloud to Simulate a Cloud Scenario

In order to simulate a cloud computing scenario, Simcan2Cloud requires two input files:

- *scenario.ned*, a plain-text file that configures the topology of the system. In particular, this file contains the type of a cloud provider, the type of a user generator, the models of one or several data-centers and how these are connected to the cloud provider through a communication network.
- *omnetpp.ini*, a plain-text file that contains a list of (parameter,value) pairs for configuring each module involved in the simulated scenario.

```
network CloudScenario{
        parameters:
                string serviceList;
                string vmList;
                string userList;
                string dataCentersList;
        submodules:
                dc_A:DC_4kcn_512sn;
                dc_B:DD_8kcn_1ksn;
                cloudProvider:CloudProviderFirstFit;
                userGenerator:UserGeneratorNormal;
        connections allowunconnected:
                cloudProvider.toDataCenter++ --> Eth10G -->
                     dc_A.fromCloudProvider;
                cloudProvider.fromDataCenter++ <-- Eth10G <--
                     dc_A.toCloudProvider;
                cloudProvider.toDataCenter++ --> Eth10G -->
                     dc_B.fromCloudProvider;
                cloudProvider.fromDataCenter++ <-- Eth10G <--
                     dc_B.toCloudProvider;
                cloudProvider.toUserGenerator --> Eth10G -->
                     userGenerator.fromCloudProvider;
                userGenerator.toCloudProvider --> Eth10G -->
                     cloudProvider.fromUserGenerator; }
```

Listing 2 Excerpt from the scenario.ned configuration file.

```
1 CloudScenario.vmList = "2 VM_med 15.0 2 2.0 500.0 4.0 VM_large
      23.0 4 4.0 1000.0 8.0"
2 CloudScenario.userList = "2 User_A 10000 1 AppDataIntensive 5 1
      VM_med 1 2 User_B 50000 1 AppDataIntensive 3 2 VM_large 1 3
      VM_med 1 1"
3 CloudScenario.dataCentersList = "2 dc_A 1 4 Rack_Cmp 16 64
      Node_Cmp 500 4.0 4 60000 1 2 Rack_Sto 16 16 Node_Sto 4000
      16.0 4 30000 dc_B 1 8 Rack_Cmp 16 64 Node_Cmp 500 4.0 4 60000
      1 4 Rack_Sto 16 16 Node_Sto 4000 16.0 4 30000"
4
5 CloudScenario.userGenerator.allUsersArriveAtOnce = false
6 CloudScenario.userGenerator.intervalBetweenUsers = normal(5s,0.3s)
7 CloudScenario.userGenerator.maxSubscriptionTime = 8
```

Listing 3 Excerpt from the omnetpp.ini configuration file.

Basically, the configuration of a simulated cloud scenario is divided into three groups: Workload, Data-Centers and Cloud-Provider. In order to illustrate the input for Simcan2Cloud, we present a running example. The configuration files of this example are depicted in Listing 2 and Listing 3. Since a configuration file may contain hundreds of lines, only the most representative parameters are shown in this example.

The workload represents a collection of users that access the cloud. The configuration of a workload must contain the number of instances, for each user type, to be generated in the simulated cloud environment and how these users access the system. First, the models of each VM involved must be provided. This example models two VMs (see line 1 of Listing 3). The first VM, called

VM_med, contains 2 CPU cores with a computational power of 2.0 CUs, a 500 GB disk and 4 GB of memory. However, *VM_large* provides 4 CPU cores with 4.0 CUs, a 1,000 GB disk and 8 GB of memory. Similarly, this example models 2 user types (see line 2 of Listing 3). The workload used in this example consists of 10,000 *User_A* instances, each one executing 5 instances of a service called *AppDataIntensive* on one *VM_med*, and 50,000 *User_B* instances, each executing 3 instances of the service *AppDataIntensive* on 1 *VM_large* and 1 *VM_med*. The user generator uses a normal distribution to calculate the time between two consecutive user requests (see line 6 of Listing 3). Finally, the parameter that configures the maximum subscription time is set in Listing 3 (see line 7).

The cloud in this example consists of 2 data-centers. The first data-center, called *dc_A* (see lines 8-9 of Listing 2), contains 4,096 computing nodes allocated in 4 racks, where each one houses 16 boards, each one allocating 64 blades. These blades, also called computing nodes, are provided with a 500 GB disk, 4 GB of memory and a CPU with 4 cores with a computing power of 60,000 MIPS. This data-center also contains 512 storage nodes, grouped in 2 racks. The second data-center, called *dc_B*, contains 8,192 computing nodes and 1024 storage nodes. This data-center uses the same configuration for the computing and storage nodes as the previous data-center. These parameters are configured in Listing 3 (see line 3).

The cloud provider in this example is configured in Listing 2 (see line 10). In this case, the cloud provider uses a first fit algorithm to allocate the requested VMs on the available resources.

Finally, lines 13-20 of Listing 2 contain the connections between each data-center and the cloud provider. In this case, each link is configured as a 10Gpbs Ethernet.

5 Mappings and Transformations

The transformations from a UML cloud model to the corresponding Simcan2Cloud configuration files are performed in several steps by using hierarchical subtransformations. In the following, we define the transformations performed and we describe the unit tests applied for their validation.

5.1 Transformation

The hierarchical transformations - applied to the UML cloud model - to obtain the corresponding configuration files for the Simcan2Cloud simulator are shown in Figures 9 and 10. These transformations are implemented in the plug-in *es.uclm.uml2cloud.m2t.simcan2cloud*, which has been implemented using Acceleo for the generation of the *.ned* and *.ini* files corresponding to the modeled scenario. A set of Acceleo templates has therefore been designed, with each one corresponding to a subtransformation shown in Figures 9 and 10.

Figure 9 contains the transformations implemented to produce the *.ned* configuration file (topology). The main transformation is *generateNEDFile*, which invokes the subtransformations *generateHeaderNEDFile*, *generateDataCenterDefinition* and *generateInfrstructure* to produce the header section, data center definition part and general infrastructure information, respectively. These subtransformations invoke the other subtransformations in the bottom of the figure to generate the information about the rack elements, rack connections, data center network and elements, cloud connections, users and cloud provider.

In the same way, Figure 10 contains the transformations implemented to produce the *.ini* file (cloud parameters). The main transformation is *generateINIFile*, which invokes the subtransformations *generateHeaderINIFile*, *generateCloudProviderConfiguration*, *generateDataCenterConfiguration* and *generateUserGeneratorConfiguration* to produce the header section, cloud provider configuration section, data center parameters and configuration of the user generator, respectively.

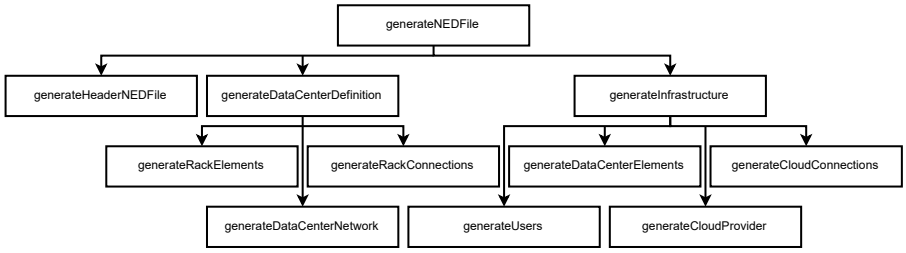


Fig. 9 Simcan2Cloud M2T *.ned* transformation scheme.

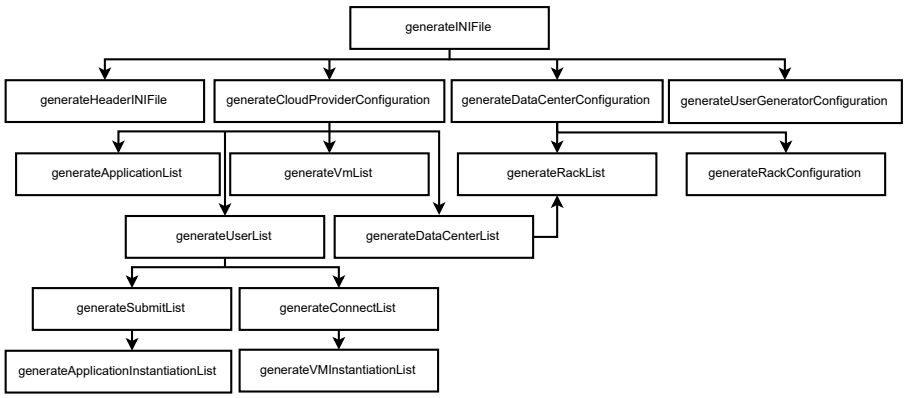


Fig. 10 Simcan2Cloud M2T *.ini* transformation scheme.

Thus, the user needs to press only the *Acceleo-Model-to-Text* transformation button in the tool (see Figure 11), which will trigger the two main trans-

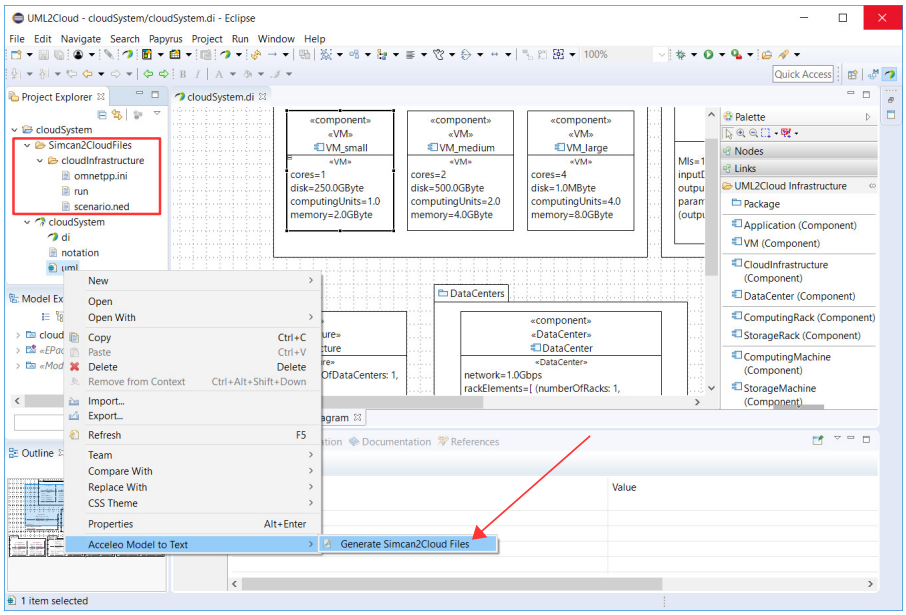


Fig. 11 A snapshot of the tool: transformation button.

formations *generateNEDFile* and *generateINIFile* mentioned above (see lines 7 and 8 in Listing 4 in Appendix A), to produce the configuration files *scenario.ned* (system topology) and *omnetpp.ini* (configuration parameters).

5.2 Transformation Validation

Proving the correctness of model transformations formally is difficult and requires formal verification techniques. An alternative approach that is widely applied in the industry is validation by testing. Validation of model transformations is important for ensuring their quality [24]. Successful validation must take into account the characteristics of model transformations and develop a suitable fault model on which test case generation can be based, by checking the correctness of the output provided by the model transformation. Thus, in this work, we use the unit testing method proposed by Tiso et al. [51] to generate a test case for each subtransformation appearing in Figures 9 and 10.

This test suite is also implemented as an Acceleo transformation, which is built by defining a template for each test case (i.e., a test template for each subtransformation). During its execution, the Acceleo transformation performs four logical steps. First, an OCL query extracts – from the input model – the specific suitable elements to be used as input for the subtransformation and executes the test case for each element obtained. The template then builds a corresponding regular expression and executes the subtransformation being tested by passing the model element as a parameter. These regular expressions

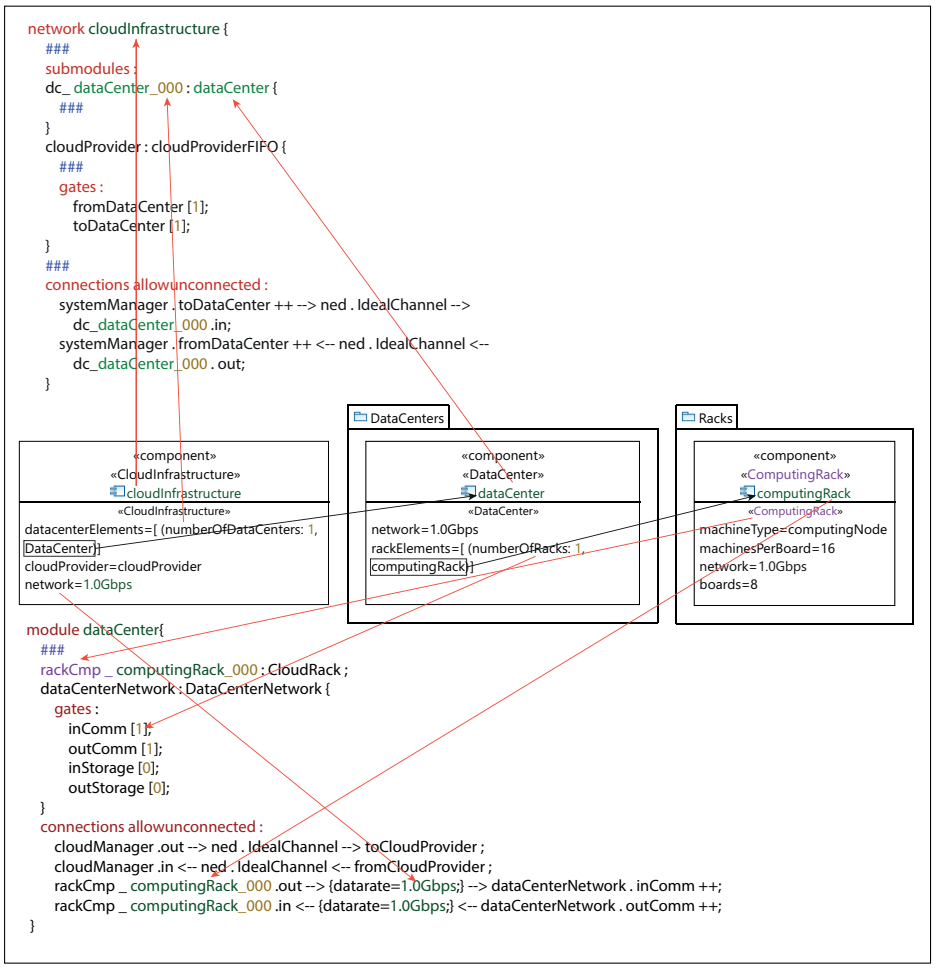


Fig. 12 Transformation example.

are defined by checking the structure of configuration files. A Java method is next invoked through a special type of Acceleo query called Java Service Wrapper[2], which matches the output of the subtransformation with the regular expression to check if it is correct. Finally, the test template report is written as an HTML report to make reading easier. Figure 13 shows a unit test report that was obtained when the subtransformation *generateHeaderNEDFile* was implemented. The result of the test for the *DataCenter1* element was *Fail*, which means that the structural definition of this element was not correct since the code produced did not match the regular expression. Hence, this subtransformation had to be modified to fix the error.

[2] Acceleo - Java Service Wrappers: `https://wiki.eclipse.org/Acceleo/Getting_Started#Java_services_wrappers`

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 { parameters: string appList; string vmList; string userList; gates: input fromCloudProvider; output toCloudProvider;

Fig. 13 Unit test report.

6 Empirical Evaluation

In this section, we carry out an experimental evaluation where different cloud architectures have been modeled and analyzed. The idea is to check how the underlying architecture of the cloud processes a given workload and, in those cases where the cloud collapses, to locate the saturation point. In essence, a workload consists of a large number of users requesting VMs to the cloud for executing applications. To generate representative workloads, we provide different types of users, which have been designed by using our proposed profile (see Figure 3), and ten different configurations of VMs where a CPU-intensive application [34] is executed.

In this study, our proposed profile has been used to model both the users accessing the cloud and the underlying cloud infrastructure (see Section 3). In the following, the term *user* refers to a client that requests resources from the cloud, *user model* denotes the behavior of a user and *workload* refers to a collection of users interacting with a cloud system.

Firstly, we present the experimental settings in Section 6.1. In Section 6.2 we investigate how the behavior of the users affects the overall cloud performance for processing a workload. In essence, this behavior is determined by the number of resources required by each user and the maximum amount of time a user is willing to wait for obtaining the requested resources, which is established via the *maxSubTime* parameter (see Figure 3). Next, we analyze in Section 6.3 how the user distribution for accessing the cloud affects overall system performance. In this case, the workloads to be processed by the cloud have been generated by using different statistical distributions. In each experiment, the scalability of the systems using different data-centers is also studied.

6.1 Experimental settings

This section describes in detail the settings used to carry out the experiments. The generated workloads have been simulated in the modeled cloud systems (see Figure 14) by using the Simcan2Cloud simulation tool.

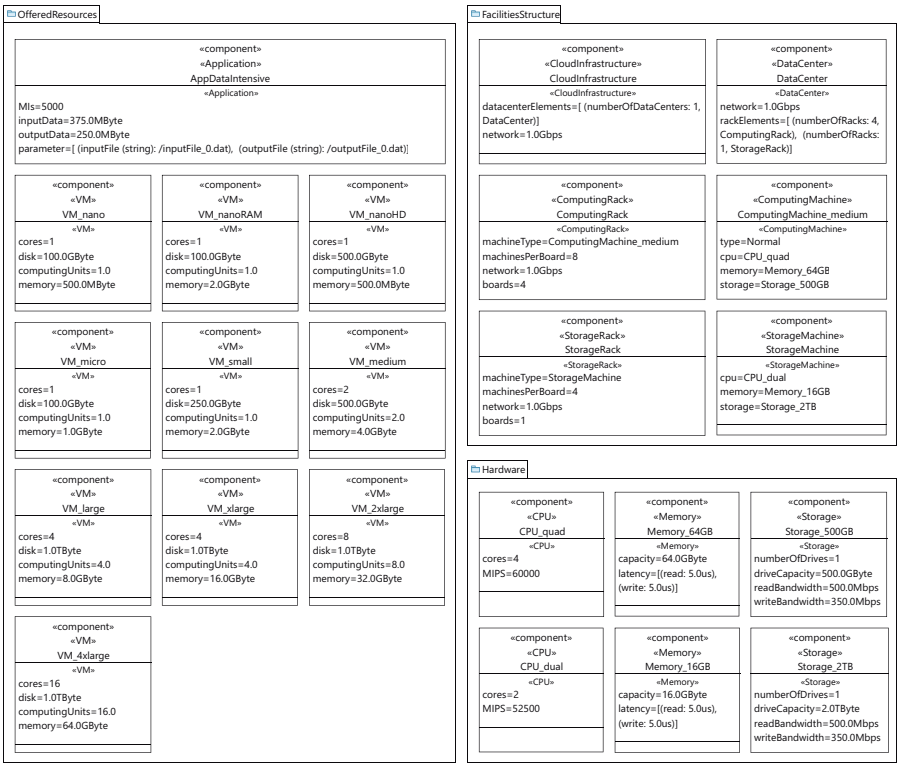


Fig. 14 Cloud system model.

Firstly, we provide 10 different models of VMs. Each cloud has been modeled as containing one data-center. All the physical machines are allocated in racks, where each rack contains a configurable number of blades that are grouped on boards. Each board contains a switch that interconnects all the blades allocated on the board. Thus, a global switch connects the switches of each board with the exterior network. All the components of the data-center are connected using a Gigabit Ethernet network.

The modeled data-centers are homogeneous in the sense that all the nodes of the same type have the same hardware features. Hence, all the computing nodes that are allocated to execute the VMs rented by the users, are equal. Similarly, the storage nodes, which are used to manage the data accessed by the computing nodes, have the same characteristics. However, the configuration of the computing nodes and the storage nodes differ. Each computing node has

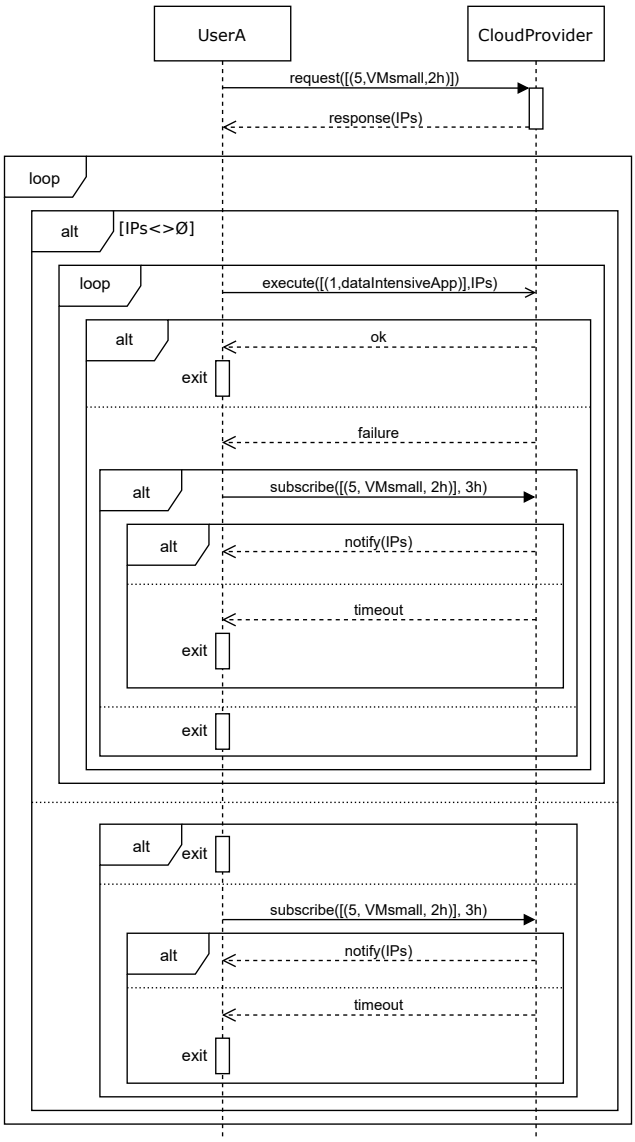


Fig. 15 Cloud interaction model for $User_A$ using $maxSubTime = 3$h.

been modeled using a 4-core CPU at 60000 MIPS, a 2 TB disk and 64 GB of RAM. Similarly, each storage node contains a 2-core CPU at 52500 MIPS, a 2000 GB disk and 16GB of RAM.

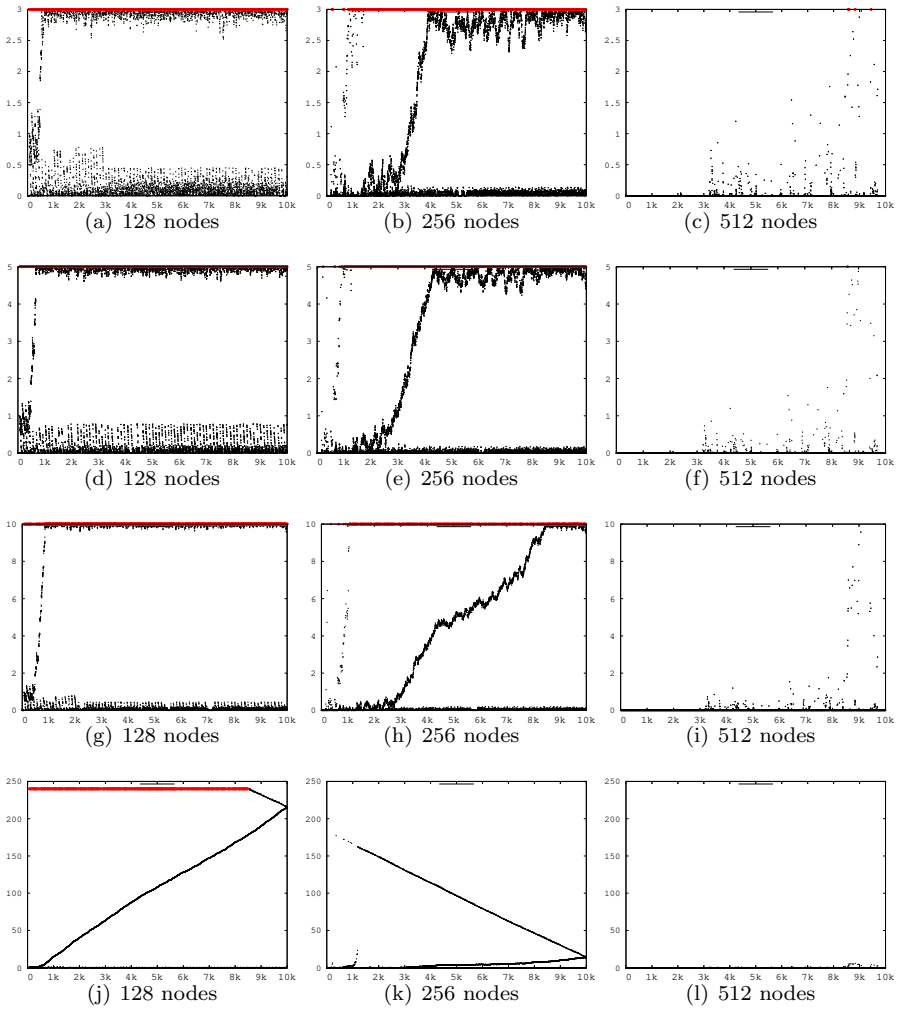
6.2 Analyzing the overall system performance

To analyze the overall system performance, we define different workloads to be processed by the cloud. Furthermore, scalability is also analyzed by using the same workload configuration, but taking different sizes for the cloud studied. Table 1 shows four different workloads that are used to analyze the behavior of the cloud. In this table, the first column refers to the user type, the second column represents the resources requested for the user, that is, the instance(s) of the virtual machine(s) and the period(s) of time that the user expects to use the VM. The following columns represent the workloads, namely ω_1, ω_2, ω_3 and ω_4, each one indicating the number of user instances generated to be processed by the cloud. As an illustration, Figure 15 contains the specific sequence diagram for the interactions of $User_A$ with the cloud provider. In this experiment, if a user requests resources from the cloud and these are not available, the user submits a subscription to the cloud to be notified when the requested resources become available. This subscription is only valid for a period, which is indicated via the *maxSubTime* parameter.

User	VMs requested by the user	ω_1	ω_2	ω_3	ω_4
$User_A$	$5 \times VM_{small}$ for 2 h.	5000	1000	500	1
$User_B$	$5 \times VM_{medium}$ for 2 h., $5 \times VM_{large}$ for 3 h.	1150	1000	1500	2490
$User_C$	$2 \times VM_{medium}$ for 2 h.	3725	1000	1000	10
$User_D$	$50 \times VM_{medium}$ for 2 h.	125	1000	2000	1500
$User_E$	$1 \times VM_{large}$ for 10 h.	0	1000	100	100
$User_F$	$2 \times VM_{micro}$ for 2 h., $2 \times VM_{small}$ for 3 h.	0	1000	900	1799
$User_G$	$2 \times VM_{micro}$ for 2 h., $2 \times VM_{small}$ for 2 h. $2 \times VM_{medium}$ for 1 h.	0	1000	500	500
$User_H$	$1 \times VM_{nanoRAM}$ for a day	0	1000	500	100
$User_I$	$1 \times VM_{nano}$ for a day	0	1000	1000	500
$User_J$	$5 \times VM_{small}$ for 2 h., $5 \times VM_{medium}$ for 3 h. $5 \times VM_{large}$ for 3 hours.	0	1000	2000	3000

Table 1 Workloads defined to analyze the overall performance of the clouds.

Figure 16 shows the results obtained from the first configuration, where the cloud processes the workload ω_1, which consists of 10000 users as it is presented in Table 1. Each row in this figure shows the simulation results obtained when using the same value for the subscription time. Thus, charts a, b and c in the first row use a subscription time of 3 hours, charts d, e and f use a subscription time of 5 hours, charts g, h and i use a subscription time of 10 hours, and finally, charts j, k and l use a subscription time of 240 hours. It is important to remark that the variable *maxSubTime* represents the maximum amount of time that a user is willing to wait until the resources are available. However, in these experiments, this time may not correspond with the time elapsed since the user requests the sources until the user has access to them. Each column of the charts in Figure 16 shows the simulation results using a data-center containing the same number of physical machines. In this



(a) 128 nodes (b) 256 nodes (c) 512 nodes

(d) 128 nodes (e) 256 nodes (f) 512 nodes

(g) 128 nodes (h) 256 nodes (i) 512 nodes

(j) 128 nodes (k) 256 nodes (l) 512 nodes

Fig. 16 Overall performance of the cloud for processing ω_1 using different *maxSubTime* values (3, 5, 10 and 240 hours).

case, 128, 256 and 512 computing nodes are used, while the number of storage nodes used is 4, 8, and 16, respectively.

In each chart, the x-axis represents the user IDs, which are numbered from 0 to 9999, while the y-axis represents the waiting time (in hours), that is, the time elapsed from the moment the user requests resources from the cloud until the user has been granted access to the requested resources. Thus, the maximum value on the y-axis corresponds to the subscription time. Those users that do not obtain the requested resources within the subscription time are represented with a red dot, which means that these users abandon the

cloud, while the other users, namely those that have been able to access and use the requested resources, are represented with a black dot.

The first row of the charts (a, b and c) represents the simulation of a cloud using different data-centers, each one containing 128, 256 and 512 nodes, and a subscription time of 3 hours. In those cases in which the data-center contains 128 and 256 nodes, it is noticeable that a high number of users, specifically 3895 and 1488, respectively, do not obtain the requested resources. Consequently, the size of these data-centers is not enough to process the workload, and therefore the cloud system collapses and is not able to provide resources to a significant number of users. Although this scenario, in which the waiting time progressively increases, is slightly alleviated when the number of physical machines is increased (see Figure 16.b), using 512 nodes renders a better scenario. In this case, only 5 users abandon the cloud, while most of the users obtain the requested resources in the first few minutes.

The second row of the charts shows the simulation results when using a subscription time of 5 hours. In this case, we obtain similar results to those in the previous cases. The number of users that abandon the system in this case is 3877, 1390 and 1 when a data-center containing 128, 256 and 512 nodes is used, respectively. Similarly, the simulations using a subscription time of 10 hours do not drastically reduce the number of users that abandon the system. In this case, the number of users that abandon the cloud is 3810, 1237 and 0 when a data-center containing 128, 256 and 512 nodes is used, respectively.

Finally, using a long subscription time renders a totally different scenario. The last row of the charts depicts the simulation results when using a subscription time of 10 days, that is, 240 hours. When considering a small data-center containing 128 nodes plots, two different situations for the users arise. On the one hand, those users requesting a small quantity of resources eventually obtain access to the system. However, the average waiting time increases as the number of users that are waiting for the resources increases (see lower black line). On the other hand, those users requesting a large amount of resources are forced to wait. Hence, these users usually abandon the system (see upper red line). When the number of users that are waiting for resources decreases, the cloud is able to provide the requested resources, and therefore, the waiting time is progressively reduced. Using a data-center containing 256 nodes renders a similar situation for the users. However, the system performance improves significantly, and even when the system is saturated all the users are able to obtain the requested resources before the subscription time expires (see Figure 16.k). Additionally, the average waiting time is considerably reduced for those users requesting a small quantity of resources (see lower black line). In these simulations, 1091 users abandon the system when the cloud contains 128 nodes, while none of the users abandon the system otherwise.

It is important to remark that when using a data-center containing 1024 nodes the system is able to provide the requested resources to all the users in the first few minutes after their arrival.

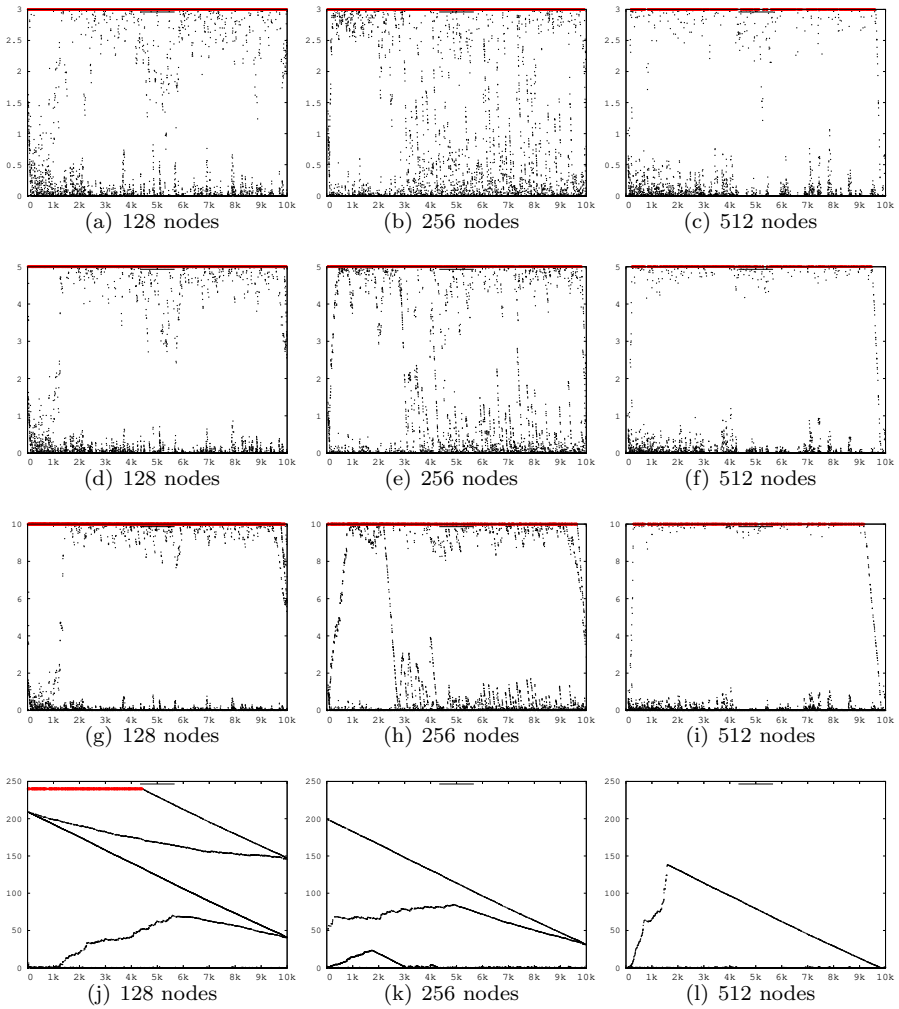


Fig. 17 Overall performance of the cloud for processing ω_2 using different *maxSubTime* values (3, 5, 10 and 240 hours).

In the next experiment the cloud processes ω_2, which consists of a total of 1000 users and contains a higher variety of users. The results of this experiment are depicted in Figure 17.

In general, the number of successfully processed users increases when the variable *maxSubTime* is also increased. This can be appreciated in the charts 17.j, 17.k and 17.l, where the maximum subscription time is set to 240 hours. In the rest of the cases, the cloud is collapsed, which is noticed by the red line – at the top of the chart – indicating that are users abandoning the system.

However, we appreciate an interesting behavior when the system scales horizontally, that is, when the cloud provides a higher number of physical

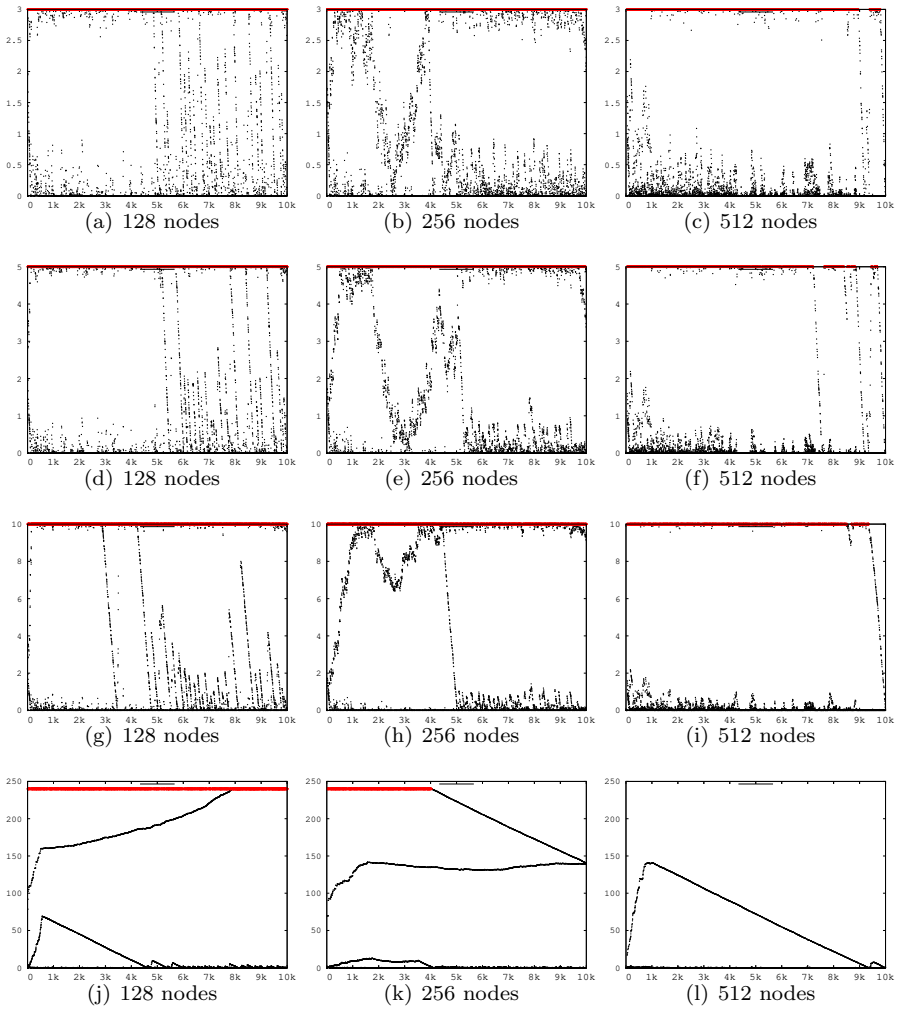


Fig. 18 Overall performance of the cloud for processing ω_3 using different *maxSubTime* values (3, 5, 10 and 240 hours).

machines. In these cases, the cloud completely processes the workload when the cloud provides 256 physical machines. Also, the waiting time is reduced from 200 to – approximately – 140 hours in the worst-case scenario (see Figure 17.k and Figure 17.l). The different lines of these charts show how the waiting time decreases for the different type of users (see Figure 17.j). Broadly speaking, the users that request a small number of resources (i.e. $User_I$) are more easily allocated than the users requesting a high number of CPU cores (i.e. $User_J$). Consequently, the latter may be waiting for longer intervals of time until the resources are available. When the system is scaled, the chart only shows one

line, which represents the same tendency for all the users that are waiting for the resources (see Figure 17.l).

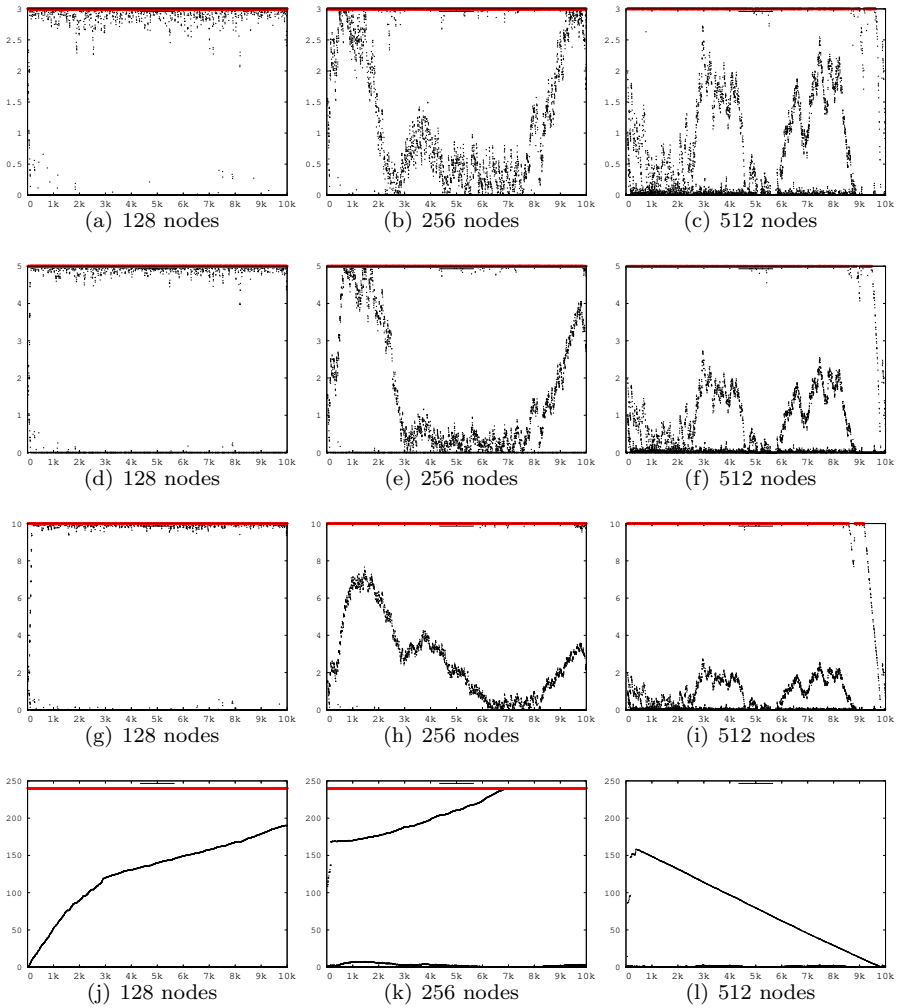


Fig. 19 Overall performance of the cloud for processing ω_4 using different *maxSubTime* values (3, 5, 10 and 240 hours).

Figure 18 and Figure 19 shows the results obtained when the cloud processes the workloads ω_3 and ω_4, respectively. Similarly, in these cases, the cloud is also collapsed when the variable *maxSubTime* is set below 240 h. On the contrary, when the users are willing to wait – as a maximum – 240 hours, the overall performance is significantly improved. Only when the cloud provides

512 physical machines all the users are successfully processed. However, we observe different behaviors from these results.

Figure 18.j shows different lines that depict how the waiting time increases and decreases at the same time. As a similar scenario occurs when the cloud process ω_2 (see Figure 17), we notice several tendencies for the different type of users requesting a different amount of resources. In this case, ω_3 contains a higher number of users requesting a greater amount of resources than ω_2 (see $User_B$, $User_D$ and $User_J$ in Table 1). These users are not able to obtain access to the requested resources and, consequently, the waiting time is increased until other users leave the system. Increasing the physical resources alleviates this issue, providing similar performance to the one obtained when the cloud process ω_2.

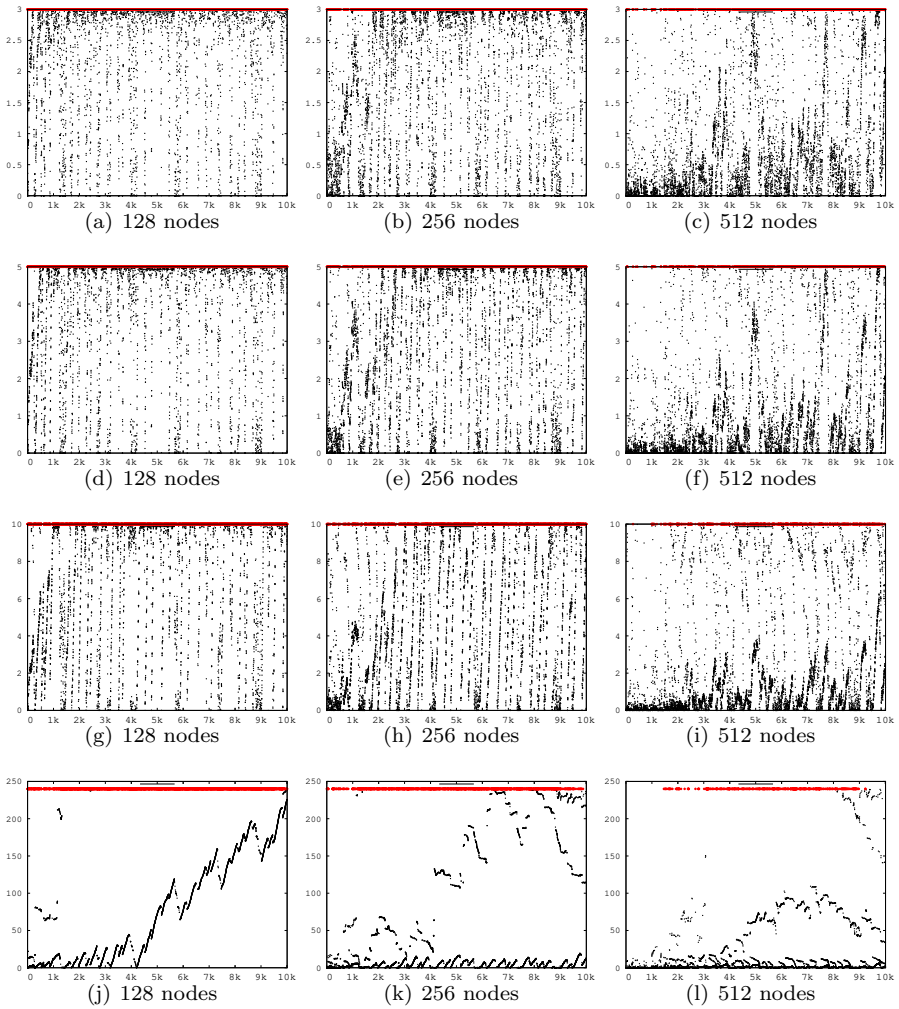
The processing of ω_4 renders a different scenario. In this case, the waiting time constantly grows when the system provides 128 physical machines. The overall performance is slightly alleviated when the system provides 256 hosts. However, the cloud is only able to fully process ω_4 when the cloud provides 512 nodes. In this case, the waiting time reaches 150 hours in the worst-case scenario (see Figure 19.l).

Additionally, we have carried out an experiment where a real trace has been adapted to be executed in the Simcan2Cloud simulator. In particular, this trace contains two years worth of accounting records produced by the DJM software running on the 1024-node CM-5 at Los Alamos National Lab (in short, LANL)[3].

This trace contains up to 122000 users and has been gathered from a system containing 1024 physical machines. However, we need the information that specifies the resources required by each user and the time-stamp representing the exact moment when each user access to the system. Specifically, we use the following information from the trace: Job Number, Submit Time, Run Time, Requested Number of CPU cores (VMs) and Requested time.

Figure 20 shows the results obtained by the cloud for executing the LANL CM-5 trace using different configurations. In this case, we appreciate that the system is not able to completely process all the user requests. However, these results show that increasing the number of physical machines and the value of the variable *maxSubTime* significantly improves the overall system performance. In those cases where the cloud provides 512 physical machines, several peaks are depicted (see charts 20.c, 20.f, 20.i and 20.l) which show how the system fluctuates when the users that are waiting to obtain the requested resources, finally get access to them. In those cases when the *maxSubTime* is set to 240 h., these charts show the tendency of the waiting time, which is more noticeable when a small data-center is used.

[3] Trace available at: `http://www.cs.huji.ac.il/labs/parallel/workload/logs.html`



(a) 128 nodes (b) 256 nodes (c) 512 nodes

(d) 128 nodes (e) 256 nodes (f) 512 nodes

(g) 128 nodes (h) 256 nodes (i) 512 nodes

(j) 128 nodes (k) 256 nodes (l) 512 nodes

Fig. 20 Overall performance of the cloud for processing the trace LANL CM-5 using different *maxSubTime* values (3, 5, 10 and 240 hours).

6.3 Analizing the impact of the user distribution

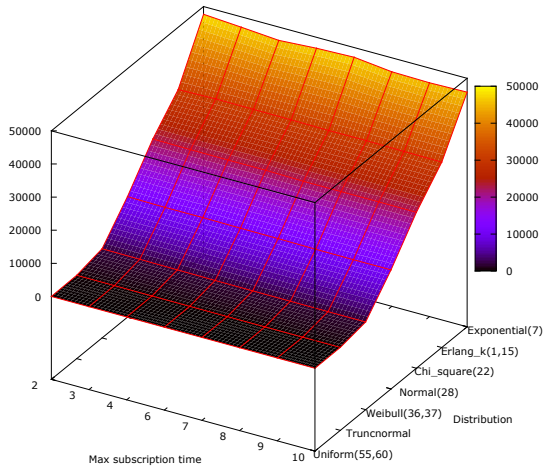
The second experiment has been carried out to analyze how the user distribution affects overall system performance. In this case, different workloads consisting of 50000 users have been created by using several statistical distributions, which establish a time interval between two consecutive users accessing the cloud (see Table 2). Thus, we use the following distributions to generate the workloads: Exponential, Erlang, Chi-square, Normal, Weibull, Truncnormal and Uniform.

Figure 21 shows the results obtained when the cloud processes the workload ω_5. Specifically, Figure 21.a shows the simulation results using a data-center of 512 nodes and 21.b shows the simulation results using a data-center containing 1024 nodes. In these charts, the x-axis represents the subscription time, the y-axis represents the distribution used to generate the workload, where the values of each distribution are indicated in seconds, and the z-axis refers to the number of users that abandon the cloud, that is, they do not obtain the requested resources within their subscription times. The generated workloads are sorted by using a sorting criteria based on the density of users. Hence, the exponential distribution generates a workload in which users arrive at the cloud system within a 200 hour time-frame, while the uniform distribution generates a workload in which users arrive at the cloud within a 1600 hour time-frame.

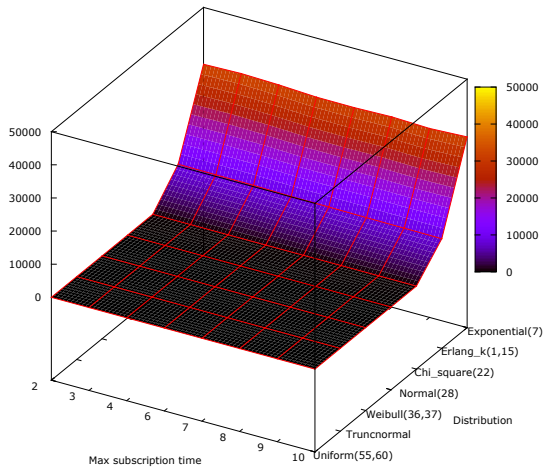
User	VMs requested by the user	ω_5	ω_6
$User_A$	$5 \times VM_{small}$ for 2 h.	47500	2000
$User_B$	$5 \times VM_{medium}$ for 2 h., $5 \times VM_{large}$ for 3 h.	5000	2000
$User_C$	$2 \times VM_{medium}$ for 2 h.	47000	2000
$User_D$	$50 \times VM_{medium}$ for 2 h.	500	2000
$User_E$	$1 \times VM_{nano}$ for 5 h.	0	2000
$User_F$	$1 \times VM_{micro}$ for 5 h.	0	2000
$User_G$	$1 \times VM_{small}$ for 5 h.	0	2000
$User_H$	$1 \times VM_{medium}$ for 5 h.	0	2000
$User_I$	$1 \times VM_{large}$ for 5 h.	0	2000
$User_J$	$1 \times VM_{xlarge}$ for 5 h.	0	2000
$User_K$	$1 \times VM_{2xlarge}$ for 5 h.	0	2000
$User_L$	$1 \times VM_{4xlarge}$ for 5 h.	0	2000
$User_M$	$5 \times VM_{nano}$ for 2 h., $5 \times VM_{micro}$ for 3 h.	0	2000
$User_N$	$5 \times VM_{small}$ for 2 h., $5 \times VM_{medium}$ for 3 h.	0	2000
$User_O$	$5 \times VM_{medium}$ for 2 h., $5 \times VM_{large}$ for 3 h.	0	2000
$User_P$	$5 \times VM_{large}$ for 2 h., $5 \times VM_{xlarge}$ for 3 h.	0	2000
$User_Q$	$5 \times VM_{xlarge}$ for 2 h., $5 \times VM_{2xlarge}$ for 3 h.	0	2000
$User_R$	$5 \times VM_{micro}$ for 2 h., $5 \times VM_{small}$ for 3 h.	0	2000
$User_S$	$5 \times VM_{small}$ for 2 h., $5 \times VM_{medium}$ for 3 h., $5 \times VM_{large}$ for 3 h.	0	2000
$User_T$	$10 \times VM_{micro}$ for 2 h., $10 \times VM_{medium}$ for 3 h., $10 \times VM_{4xlarge}$ for 3 h.	0	2000
$User_U$	$20 \times VM_{large}$ for 2 h., $20 \times VM_{xlarge}$ for 3 h., $20 \times VM_{2xlarge}$ for 3 h.	0	2000
$User_V$	$5 \times VM_{nano}$ for 2 h., $5 \times VM_{micro}$ for 3 h., $5 \times VM_{xlarge}$ for 3 h.	0	2000
$User_W$	$10 \times VM_{nano}$ for 2 h., $10 \times VM_{micro}$ for 3 h., $10 \times VM_{2xlarge}$ for 3 h.	0	2000
$User_X$	$10 \times VM_{small}$ for 2 h., $10 \times VM_{medium}$ for 3 h., $10 \times VM_{xlarge}$ for 3 h.	0	2000
$User_Y$	$1 \times VM_{nano\_hd}$ for 12 h.	0	1500
$User_Z$	$1 \times VM_{nano}$ for a day	0	500

Table 2 Definition of several workloads to analyze the impact of the user distribution on the overall system performance.

These charts show that the cloud is able to properly process workloads containing a low density of user requests, that is, workloads generated by using both a Uniform and a Trucnormal distribution, which generate user requests within a 1600 and 1400 hour time-frame, respectively. In this case, only 145 users abandon the system when a subscription time of 2 hours is considered.



(a) 512 nodes



(b) 1024 nodes

Fig. 21 Performance of the cloud for processing ω_5

However, as the density of users increases, the number of users that abandon the cloud increases as well. Although this effect can be appreciated in both clouds, the system using a data-center containing 512 nodes easily collapses. In this case, when using a Weibull distribution, in which the workload generates users accessing the cloud within a 960 hour time-frame, the cloud is not able to entirely attend to all the user requests and, consequently, around 2000 users abandon the cloud.

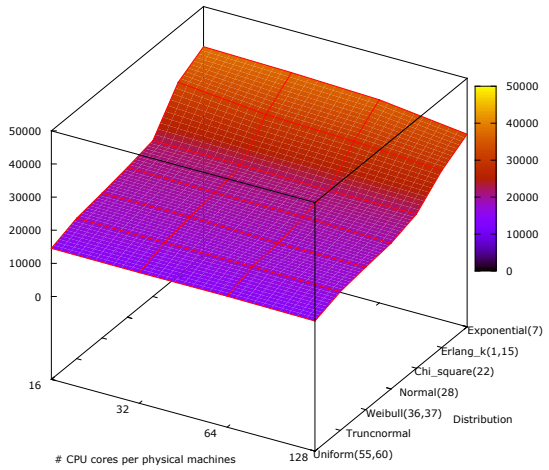
The same effects can be observed when using the other distributions. The worst-case scenario consists in processing a workload generated with an exponential distribution, where around 47000 users abandon the system. However, using a larger data-center partially alleviates this scenario, allowing the cloud to properly process most of the workload, thus providing the requested resources to all users. Only when the workload is generated by using both the Erlang and exponential distributions the cloud is not able to process the workload entirely, and consequently, around 8000 and 32000 users abandon the cloud, respectively.

As we noticed the variable *maxSubTime* has a slight impact on this experiment, we have performed a simulation by scaling – vertically – the system to process the workload ω_6. For this, we have configured each physical machine in the cloud to provide 16, 32, 64 and 128 CPU cores. In this case, the variable *maxSubTime* is set to 5 hours.

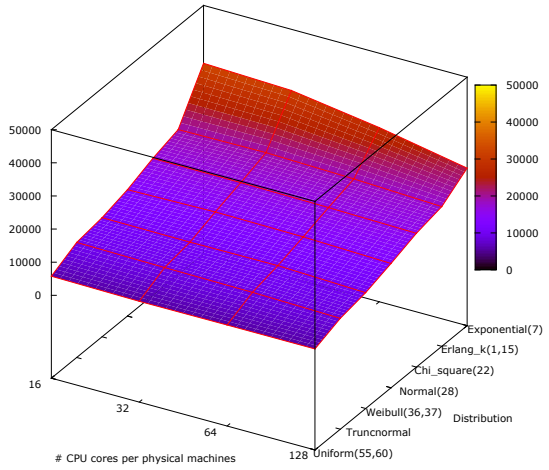
When the cloud provides 512 physical machines (see Figure 22.a), the number of users that abandon the system in the best-case scenario ranges from 14520 to 14244, that is, when the workload is generated using a uniform distribution. In the worst-case scenario, that is, using the exponential distribution, the number of users that cannot reach the requested resources ranges from 37889 to 33128. These results show that the number of CPU cores has a greater impact on the overall system performance when the number of users that concurrently access the system increases. This fact is depicted when the workload is generated using an exponential distribution. In this case, the time-frame between two users is considerably smaller than the space between two users generated using a uniform distribution.

However, we obtain better results when the cloud provides 1024 hosts. In this case, the number of users that abandon the cloud ranges from 5768 to 5473 when the users are generated using the uniform distribution. When we use an exponential distribution to generate the workload, the number of users that are forced to leave the system ranges from 32536 to 22547. Consequently, in this case, increasing the number of CPU allows the cloud to provide resources to 9989 users, which – approximately – represent the 20% of the total number of users in the trace.

Broadly speaking, the user distribution has a direct impact on the overall system performance. For this, it is crucial to customize the system using the most appropriate configuration, adjusting the number of physical machines and the number of CPU cores per machine.



(a) 512 nodes



(b) 1024 nodes

Fig. 22 Performance of the cloud for processing ω_6

7 Related Work

In the last years, optimizing, modeling and investigating the cloud have become a focus of much interest [15, 20, 27, 30]. However, these are very challenging tasks due to the following reasons. First, providing an accurate and formal

model of the cloud is complex. There are a large number of variables (i.e. the behavior of users and workload) and components (CPUs, memories, networks, virtual machines) that must be formally modeled to accurately represent the behavior of the system. Second, since there are a large number of variables that cannot be controlled, such as network traffic, it is very difficult to repeat a test case.

At present, some well-known cloud providers and third-party cloud services provide tools for calculating the financial cost of resource and service utilization [1, 29, 44, 45]. Unfortunately, these tools calculate only the static cost, that is, the cost associated with the requested resources for a time-frame. Hence, data such as performance, average response time and percentage of resource utilization are not provided by these tools.

A promising alternative way to investigate and model the cloud consists in using simulation tools, as these offer several advantages: i) Simulation experiments are cheap and repeatable, and can be launched in parallel in order to reduce the execution time. ii) Simulation is more flexible than performing experiments on a real system. In a simulation experiment the entire model can be easily modified by changing configuration parameters. On the contrary, real systems require making changes to the hardware, which is usually more expensive and time-consuming. iii) Simulation does not require specific hardware to launch experiments. iv) Simulators can be easily shared with other researchers. However, simulation also entails certain problems. For instance, results obtained from simulation do not provide 100% accuracy and, in some cases, modeling the required system under test with enough detail may require a significant amount of effort and time.

The high diversity and continuously growing scale of simulation and modeling tools makes the selection of an appropriate tool to perform the required research very difficult [19]. Over the last few years, a wide spectrum of simulation tools have been proposed to study different aspects of cloud systems [19], such as energy consumption[13, 23], throughput and utilization of resources [14, 20], and VM scheduling strategies [31].

Taking into account the modeling challenge, and considering that the current simulation tools use different languages to represent the cloud infrastructure, it is a good approach to use a common graphical modeling language. Over the last few years, diverse cloud modeling approaches have emerged. Considering the report made by Bergmayr et al. [4] and focusing mainly on the graphical modeling languages, we can find the following approaches: CAML [3], TOSCA [6], MULTICLAPP [21] and MOCCA [25]. In Table 3 a comparison of our framework with these graphical modeling languages is presented.

CAML [3] makes it possible to represent cloud-based deployment topologies in UML and refine them with cloud offerings captured by dedicated UML profiles. It is based on a model library and profiles. These profiles allow you to model cloud offerings from both technical and non-technical perspectives, for example, performance and pricing features, respectively. As CAML is based on UML, it can be directly applied to UML models, which is especially useful

		CAML	TOSCA	MULTICLAPP	MOCCA	`UML2Cloud`
Scope		Description of cloud-based deployment topologies and refinement of them with cloud offerings captured by dedicated UML profiles.	Description of portable composite cloud applications for their automated provisioning and lifecycle management.	Cloud applications modeling from a cloud- provider independent perspective.	Migration of existing software to a cloud environment.	Cloud datacenter and cloud provider infrastructure modeling and simulation. Code generation for target cloud environments from component and interaction configurations.
Target		IaaS PaaS SaaS	IaaS PaaS SaaS	IaaS PaaS SaaS	IaaS PaaS SaaS	IaaS PaaS SaaS
Syntax	Abstract	UML	XML Schema	UML	MOF	UML
	Concrete	GUI	Text & GUI	GUI	GUI	GUI
	Serial.	XMI	XML & YAML	XMI	XMI	XMI
Application Structure		Class Component Deployment	Component Deployment	Component Deployment	Component Deployment	Interaction (Seq.) Component Deployment Simulation
Modeling		Arbitrary UML model editor	GUI model editor	Arbitrary UML model editor	GUI model editor	GUI model editor
Refinement		UML profiles	Ontological types Implementation artifacts	UML profiles	N/A	UML profiles
Generation		m2t:UML-Java t2m:Java-UML m2m:CAML-TOSCA	m2m: Deployment plan	m2t:Service adapter m2m:Deployment plan	m2m:Deployment plan	m2t: UML2Cloud-Simcan2Cloud

Table 3 A comparison of graphical cloud modeling languages with `UML2Cloud`.

when UML models are previously obtained and one wishes to adapt them for a cloud environment.

TOSCA [6] provides a way to enable portable automated deployment and management of composite applications. It describes the structure of composite applications in terms of so-called service templates. An extended version of TOSCA, called ToscaMart [47], enables developers to reuse existing components and fragments of applications. TOSCA is based on XML, whereas Vino4TOSCA [7] provides a graphical notation for service templates. The application model can be completed with management plans that are included by using existing workflow languages, such as BPMN [38], and can be seen as the mapping between the cloud applications and cloud environments.

MULTICLAPP [21] supports the modeling of cloud applications from a cloud-provider independent perspective. This approach models the application components in UML and refines them with a dedicated profile. Multicloud applications are modeled as a composition of software artifacts, where each one can be assigned to a different platform. Additionally, stereotypes are provided that allow components to be annotated with QoS parameters, such as their response time.

MOCCA [25] is a method for migrating existing software to a cloud environment. The MOCCA metamodel represents the architecture and the deployment of the existing software. The deployment in a cloud environment can be expressed in terms of a clustering of architectural elements and specific implementation units that are assigned to the virtual resources of a cloud environment. The virtual resources are described in Open Virtualization Format (OVF) [17], which is a textual modeling language.

Most cloud modeling languages deal with the description of cloud deployment configurations for one or several target cloud environments. They allow the modeling of the services and resources offered by the cloud provider but without paying attention to the underlying hardware. Thus, there is a gap in cloud data center infrastructure modeling, which some researchers might wish to model and simulate in order to obtain certain data from the data centers. The profile proposed in this work fills this gap. It allows researchers to model the cloud provider infrastructure, making it possible to simulate it.

The above languages exploit model transformation techniques to generate code or configuration and deployment scripts. However, even if we have these codes or scripts from the models, it is still necessary to experiment in a real cloud. Caglar et al. [10] proposes the use of MDE and simulation to estimate the performance and cost of hosting the services in the cloud. It provides the generation of simulation code for CloudSim [8, 11] from a model.

Other works have used MDE and simulation techniques successfully in other areas [16, 50]. Syntony [16] is an Eclipse-based framework for automated and tool-assisted development and analysis of network protocols. It uses UML diagrams, and MARTE and UTP profiles, to describe complex protocols. Then, it transforms the models into simulation code for OMNET++ [52].

8 Conclusions and Future Work

In this paper we have presented a framework for modeling cloud systems, called `UML2Cloud`, capturing the most relevant components of the cloud, such as its underlying architecture, virtualization, users and workloads.

Thus, one of the main contributions of this paper is the implementation of a complete framework supporting a UML profile for modeling the cloud architecture, as well as the simulated user-cloud interactions. In this way, a graphical representation of the cloud system is given, which allows us to easily understand and configure the system infrastructure and interactions. With the proposed framework a specific cloud scenario can be created by using the editor, after which it is validated and automatically translated into the cloud system configuration files to simulate the cloud system. `UML2Cloud` aids users in the sense that it hides the low-level details related to the simulation tool, that is, among other aspects, the syntax of the configuration files and scripts to run the experiments. Hence, users do not need to know the simulation platform used to simulate the modeled cloud environments.

From the results obtained by the simulator, we can draw conclusions about how the user's behavior affects the overall cloud performance. In these experiments, a large number of users and VMs have been modeled, and several workloads have been synthetically generated using a wide spectrum of user distributions. Moreover, a real trace has been adapted to be executed in Simcan2Cloud, allowing to compare the results obtained from the executions of the different workloads. As a result, the provided data renders interesting conclusions that allow configuring the cloud to improve the overall system performance depending on the workload to be executed.

In conclusion, using simulation allows us to check, without making an initial investment, important factors of the cloud, such as its scalability and performance. Hence, cloud providers can estimate the amount of resources needed to satisfy the demand for a specific workload of users. Moreover, this framework is very useful for analyzing both the horizontal and vertical scalability in those cases in which the cloud provider decides to increase the size of a cloud.

An interesting line for future research is to complete the UML cloud profile to model user decisions, cloud costs, and different allocation algorithms to obtain quantitative results and thus draw conclusions from them.

Acknowledgments

This work was partially supported by the Spanish Ministry of Science and Innovation (cofinanced by European Union FEDER funds) projects "DArDOS (Formal development and analysis of complex systems in distributed contexts: foundations, tools and applications)", reference TIN2015-65845-C3, subprojects 1-R and 2-R, and TIN2015-66972-C5 subproject 2-R, the Comunidad de Madrid project SICOMORo-CM under Grant S2013/ICE-3006,

and the Junta de Comunidades de Castilla-La Mancha (Spain) project SBPLY/17/180501/000276 (cofunded with FEDER funds, EU).

References

1. Amazon Elastic Compute Cloud (2019) Web page at http://aws.amazon.com/ec2/. Date of last access: 30th January, 2019
2. Baumgart I, Heep B, Krause S (2007) Oversim: A flexible overlay network simulation framework. In: 2007 IEEE Global Internet Symposium, pp 79–84
3. Bergmayr A, Troya J, Neubauer P, Wimmer M, Kappel G (2014) UML-based Cloud Application Modeling with Libraries, Profiles, and Templates. In: CloudMDE@ MoDELS, pp 56–65
4. Bergmayr A, Wimmer M, Kappel G, Grossniklaus M (2014) Cloud Modeling Languages by Example. In: 2014 IEEE 7th International Conference on Service-Oriented Computing and Applications, IEEE, pp 137–146, DOI 10.1109/SOCA.2014.56, URL http://ieeexplore.ieee.org/document/6978602/
5. Bernal A, Cambronero ME, Valero V, Núñez A, Cañizares PC (2019) A framework for modeling cloud infrastructures and user interactions. IEEE Access 7:43269–43285, DOI 10.1109/ACCESS.2019.2907180
6. Binz T, Breitenbücher U, Kopp O, Leymann F (2014) TOSCA: Portable Automated Deployment and Management of Cloud Applications. In: Bouguettaya A, Sheng QZ, Daniel F (eds) Advanced Web Services, Springer New York, pp 527–549, DOI 10.1007/978-1-4614-7535-4_22, URL https://doi.org/10.1007/978-1-4614-7535-4_22
7. Breitenbücher U, Binz T, Kopp O, Leymann F, Schumm D (2012) Vino4tosca: A Visual Notation for Application Topologies Based on TOSCA. In: Meersman R, Panetto H, Dillon T, Rinderle-Ma S, Dadam P, Zhou X, Pearson S, Ferscha A, Bergamaschi S, Cruz IF (eds) On the Move to Meaningful Internet Systems: OTM 2012, Springer Berlin Heidelberg, pp 416–424, DOI 10.1007/978-3-642-33606-5_25, URL https://doi.org/10.1007/978-3-642-33606-5_25
8. Buyya R, Ranjan R, Calheiros RN (2009) Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: Challenges and opportunities. In: 7th High Performance Computing and Simulation Conference (HPCS'09), IEEE Computer Society, pp 1–11
9. Byrne J, Svorobej S, Giannoutakis K, Tzovaras D, Byrne P, Östberg P, Gourinovitch A, Lynn T (2017) A review of cloud computing simulation platforms and related environments. In: 7th International Conference on Cloud Computing and Services Science (CLOSER'17), pp 651–663, DOI 10.5220/0006373006790691
10. Caglar F, An K, Shekhar S, Gokhale A (2013) Model-driven performance estimation, deployment, and resource management for cloud-hosted ser-

vices. In: Proceedings of the 2013 ACM workshop on Domain-specific modeling, ACM, pp 21–26
11. Calheiros RN, Ranjan R, Beloglazov A, Rose CAFD, Buyya R (2011) Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software - Practice and Experience 41(1):23–50
12. Casanova H, Legrand A, Quinson M (2008) SimGrid: A generic framework for large-scale distributed experiments. In: 10th Int. Conf. on Computer Modeling and Simulation, UKSIM' 08, pp 126–131
13. Castañé G, Núñez A, Llopis P, Carretero J (2013) E-mc^2: A formal framework for energy modelling in cloud computing. Simulation Modelling Practice and Theory 39:56–75
14. Cañizares PC, Núñez A, Merayo MG (2018) Mutomvo: Mutation testing framework for simulated cloud and hpc environments. Journal of Systems and Software 143:187 – 207, DOI https://doi.org/10.1016/j.jss.2018.05.010, URL `http://www.sciencedirect.com/science/article/pii/S0164121218300931`
15. Das AK, Hong J, Goswami S, Platania R, Lee K, Chang W, Park SJ, Liu L (2017) Augmenting amdahl's second law: A theoretical model to build cost-effective balanced hpc infrastructure for data-driven science. In: 2017 IEEE 10th International Conference on Cloud Computing (CLOUD'17), pp 147–154
16. Dietrich I, Dressler F, Schmitt V, German R (2007) SYNTONY: network protocol simulation based on standard-conform UML 2 models. In: Proceedings of the 2nd international conference on Performance evaluation methodologies and tools, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), p 21
17. DMTF (2013) *Open Virtualization Format (OVF), v2.0.0*. Web page at `https://www.dmtf.org/standards/ovf`. Date of last access: 30th January, 2019
18. Eurostat Statistics Explained (2018) Cloud computing - statistics on the use by enterprises. Web page at `http://ec.europa.eu/eurostat/statistics-explained/index.php/Cloud_computing_-_statistics_on_the_use_by_enterprises`. Date of last access: 30th January, 2019
19. Fakhfakh F, Kacem HH, Kacem AH (2017) Simulation tools for cloud computing: A survey and comparative study. In: IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS'17), pp 221–226
20. Filelis-Papadopoulos CK, Gravvanis GA, Kyziropoulos PE (2018) A framework for simulating large scale cloud infrastructures. Future Generation Computer Systems 79:703–714, DOI https://doi.org/10.1016/j.future.2017.06.017, URL `http://www.sciencedirect.com/science/article/pii/S0167739X17303230`
21. Guillén J, Miranda J, Murillo JM, Canal C (2013) A UML Profile for modeling multicloud applications. In: European Conference on Service-Oriented and Cloud Computing, Springer, pp 180–187

22. Kleppe AG, Warmer J, Bast W (2003) MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley Longman Publishing Co., Inc.
23. Kliazovich D, Bouvry P, Khan SU (2012) GreenCloud: A packet-level simulator of energy-aware cloud computing data centers. The Journal of Supercomputing 62(3):1263–1283
24. Küster JM, Abd-El-Razik M (2007) Validation of model transformations – first experiences using a white box approach. In: Kühne T (ed) Models in Software Engineering, Springer Berlin Heidelberg, pp 193–204
25. Leymann F, Fehling C, Mietzner R, Nowak A, Dustdar S (2011) Moving applications to the cloud: an approach based on application model enrichment. International Journal of Cooperative Information Systems 20(03):307–356
26. Mellor SJ, Clark T, Futagami T (2003) Model-Driven Development: Guest editor's introduction. IEEE software 20(5):14–18
27. Meng FJ, Zhang X, Chen P, Xu JM (2017) Driftinsight: Detecting anomalous behaviors in large-scale cloud platform. In: 2017 IEEE 10th International Conference on Cloud Computing (CLOUD'17), pp 230–237
28. Mens T, Van Gorp P (2006) A taxonomy of model transformation. Electronic Notes in Theoretical Computer Science 152:125–142
29. Microsoft Azure (2019) Web page at `http://azure.microsoft.com`. Date of last access: 30th January, 2019
30. Mishra SK, Puthal D, Sahoo B, Jena SK, Obaidat MS (2018) An adaptive task allocation technique for green cloud computing. The Journal of Supercomputing 74(1):370–385, DOI 10.1007/s11227-017-2133-4
31. Mohan N, Varma K, Choi E (2016) Study and Comparison of Virtual Machine Scheduling Algorithms in Open Source Clouds, In: Advanced Multimedia and Ubiquitous Engineering: FutureTech & MUE, Springer, pp 349–355
32. Nashaat H, Ashry N, Rizk R (2019) Smart elastic scheduling algorithm for virtual machine migration in cloud computing. The Journal of Supercomputing DOI 10.1007/s11227-019-02748-2
33. Núñez A, Hierons RM (2015) A methodology for validating cloud models using metamorphic testing. Annales of Telecommunications 70(3-4):127–135
34. Núñez A, Fernández J, García JD, García F, Carretero J (2010) New techniques for simulating high performance MPI applications on large storage networks. The Journal of Supercomputing 51(1):40–57
35. Núñez A, Fernández J, Filgueira R, García F, Carretero J (2012) SIMCAN: A flexible, scalable and expandable simulation platform for modelling and simulating distributed architectures and applications. Simulation Modelling Practice and Theory 20(1):12–32
36. Obeo (2007) Acceleo Generator. Web page at `http://www.acceleo.org/`. Date of last access: 30th January, 2019
37. OMG (2008) MOF Model To Text Transformation Language (MOFM2T) v1.0. Web page at `http://www.omg.org/spec/MOFM2T/1.0/`. Date of last

access: 30th January, 2019
38. OMG (2011) Business Process Model & Notation (BPMN). Web page at http://www.omg.org/spec/BPMN/2.0. Date of last access: 30th January, 2019
39. OMG (2014) MDA Guide revision 2.0. Web page at http://www.omg.org/cgi-bin/doc?ormsc/14-06-01.pdf. Date of last access: 30th January, 2019
40. OMG (2014) Object Constraint Language (OCL) v2.4. Web page at http://www.omg.org/spec/OCL/2.4. Date of last access: 30th January, 2019, accessed: 2016-11-13
41. OMG (2014) UML Profile Specifications. Web page at http://www.omg.org/spec/#Profile. Date of last access: 30th January, 2019
42. OMG (2015) UML specification version 2.5. Web page at http://www.omg.org/spec/UML/2.5. Date of last access: 30th January, 2019
43. OMG (2016) MOF Query/View/Transformation v1.3. Web page at http://www.omg.org/spec/QVT/1.3/. Date of last access: 30th January, 2019
44. RightScale (2017) Web page at http://www.rightscale.com. Date of last access: 24th November, 2018
45. SCALR (2017) Web page at http://www.scalr.com. Date of last access: 24th November, 2018
46. Schmidt DC (2006) Guest Editor's Introduction: Model-Driven Engineering. Computer 39(2):25–31, DOI 10.1109/MC.2006.58, URL http://dx.doi.org/10.1109/MC.2006.58, accesed: 2017-07-04
47. Soldani J, Binz T, Breitenbücher U, Leymann F, Brogi A (2016) Toscamart: A method for adapting and reusing cloud applications. Journal of Systems and Software 113:395 – 406, DOI https://doi.org/10.1016/j.jss.2015.12.025, URL http://www.sciencedirect.com/science/article/pii/S0164121215002903
48. Steinbach T, Kenfack HD, Korf F, Schmidt TC (2011) An Extension of the OMNeT++ INET Framework for Simulating Real-time Ethernet with High Accuracy. In: Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques (SIMUTools '11), pp 375–382
49. Tan Y, Wu F, Wu Q, Liao X (2019) Resource stealing: a resource multiplexing method for mix workloads in cloud system. The Journal of Supercomputing 75(1):33–49
50. Teixeira S, Agrizzi BA, Filho JGP, Rossetto S, de Lima Baldam R (2017) Modeling and automatic code generation for wireless sensor network applications using model-driven or business process approaches: A systematic mapping study. Journal of Systems and Software 132:50 – 71, DOI https://doi.org/10.1016/j.jss.2017.06.024, URL http://www.sciencedirect.com/science/article/pii/S0164121217301255
51. Tiso A, Reggio G, Leotta M (2014) Unit testing of model to text transformations. In: Proceedings of the Workshop on Analysis of Model Transformations co-located with ACM/IEEE 17th International Conference on Model Driven Engineering Languages & Systems (MoDELS 2014), Valencia, Spain, September 29, 2014., pp 14–23, URL http://ceur-ws.org/

Vol-1277/2.pdf
52. Varga A, Hornig R (2008) An Overview of the OMNeT++ Simulation Environment. In: Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops (Simutools '08), pp 1–10
53. Veselý V, Marek M, Hykel T, Rysavý O (2015) Skip this paper - rinasim: Your recursive internetwork architecture simulator. CoRR
54. Yigitbasi N, Iosup A, Epema D, Ostermann S (2009) C-Meter: A Framework for Performance Analysis of Computing Clouds. In: 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, pp 472–477

A Transformation templates and generated files

In this appendix we show how a piece of a cloud model is transformed into several lines of text that belong to different parts of the *scenario.ned* configuration file. Let us consider the excerpt of a cloud model shown in Figure 23. Component and stereotype property values are extracted through the implemented Acceleo templates, and then these are written in the *scenario.ned* file. To start with, the main template *generateSimcan2CloudFiles* (see Listing 4) extracts the information from the main component, that is *CloudInfrastructure*, and calls the *generateNEDFile* and *generateINIFile* templates. The *generateNEDFile* (see Listing 5) template creates the *scenario.ned* file. This template extracts the data centers referenced by the cloud infrastructure and calls the *generateScenario* and *generateDataCenter* templates to fill in the file.

The templates shown in Listings 4,5,6,7,8 and 9 write the part of the *scenario.ned* configuration file shown in Listing 10. In the same way, the templates shown in Listings 11,12,13 and 14 write the part of the *scenario.ned* configuration file shown in Listing 15.

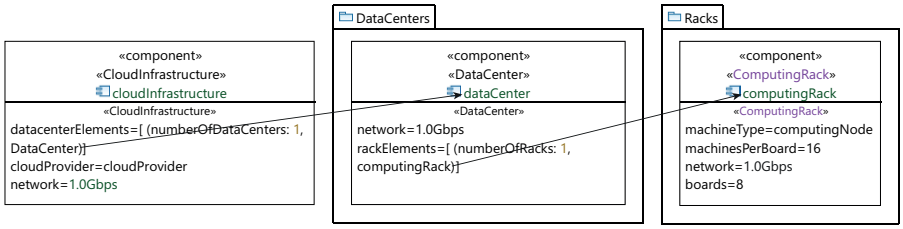


Fig. 23 A piece of a cloud system model.

Listing 4 Main Acceleo template.

```
[comment encoding = UTF-8 /]
[module
    generateSimcan2CloudFiles('http://www.uclm.es/UML/profiles/UML2Cloud/1',
    'http://www.eclipse.org/uml2/5.0.0/UML')]
[import es::uclm::uml2cloud::m2t::simcan::files::generateNEDFile /]
[import es::uclm::uml2cloud::m2t::simcan::files::generateINIFile /]

[template public generateSIMCANFiles(aCloudInfrastructure :
    CloudInfrastructure)]
[comment @main /]
        [aCloudInfrastructure.generateNEDFile()/]
        [aCloudInfrastructure.generateINIFile()/]
[/template]
```

Listing 5 generateNEDFile Acceleo template.

```
[template public generateNEDFile(aCloudInfrastructure :
    CloudInfrastructure)]

[file (aCloudInfrastructure.getFolder()+'scenario.ned', false, 'UTF-8')]
###
[aCloudInfrastructure.generateScenario()/]
[aCloudInfrastructure.datacenterElements.generateDataCenter()/]
[/file]
[/template]
```

Listing 6 generateScenario Acceleo template.

```
[template public generateScenario(aCloudInfrastructure :
    CloudInfrastructure)]
network [aCloudInfrastructure.base_Component.name/]{
  ###
  submodules:
 [aCloudInfrastructure.datacenterElements.generateScenarioDataCenter()/]
    [aCloudInfrastructure.generateScenarioCloudProvider()/]
    [aCloudInfrastructure.generateScenarioUserGenerator()/]
    [aCloudInfrastructure.generateScenarioConnections()/]
}
[/template]
```

Listing 7 generateScenarioDataCenter Acceleo template.

```
[template public generateScenarioDataCenter(aDataCenterElement :
    DataCenterElement)]
[for (index : Integer |
    Sequence(Integer){1..aDataCenterElement.numberOfDataCenters})]
dc_[aDataCenterElement.dataCenterType.base_Component.name/]_[index/]:
    [aDataCenterElement.dataCenterType.base_Component.name/]{
    ###
}
[/for]
[/template]
```

Listing 8 generateScenarioCloudProvider Acceleo template.

```
[template public generateScenarioCloudProvider(aCloudInfrastructure :
    CloudInfrastructure)]
cloudProvider:CloudProvider[aCloudInfrastructure.cloudProvider
    .resourceAllocationPolicy/]{
  ###
    gates:
      fromDataCenter['['+aCloudInfrastructure.datacenterElements->
          collectNested(numberOfDataCenters)->sum()+']'/]
      toDataCenter['['+aCloudInfrastructure.datacenterElements->
          collectNested(numberOfDataCenters)->sum()+']'/];
}
[/template]
```

Listing 9 generateScenarioConnections Acceleo template.

```
[template public generateScenarioConnections(aCloudInfrastructure :
    CloudInfrastructure)]
connections allowunconnected:
[for (aDataCenterElement : DataCenterElement |
    aCloudInfrastructure.datacenterElements)]
  [for (index : Integer |
      Sequence(Integer){1..aDataCenterElement.numberOfDataCenters})]
    cloudProvider.toDataCenter++ -->
        {datarate=[aCloudInfrastructure.network.getBandwidth()/];} -->
        dc_[aDataCenterElement.dataCenterType.base_Component.name/]
        _[index/].in;
    cloudProvider.fromDataCenter++ <--
        {datarate=[aCloudInfrastructure.network.getBandwidth()/];} <--
        dc_[aDataCenterElement.dataCenterType.base_Component.name/]
        _[index/].out;
  [/for]
[/for]

  cloudProvider.toUserGenerator --> ned.IdealChannel -->
      userGenerator.fromCloudProvider;
  userGenerator.toCloudProvider --> ned.IdealChannel -->
      cloudProvider.fromUserGenerator;
[/template]
```

Listing 10 Excerpt 1 from the generated *.ned* file.

```
network cloudInfrastructure {
  ###
  submodules:
  dc_dataCenter_000:dataCenter {
    ###
  }
  cloudProvider:cloudProviderFIFO {
    ###
    gates:
      fromDataCenter[1];
      toDataCenter[1];
  }
  ###
  connections allowunconnected:
    systemManager.toDataCenter++ --> ned.IdealChannel -->
        dc_dataCenter_000.in;
    systemManager.fromDataCenter++ <-- ned.IdealChannel <--
        dc_dataCenter_000.out;
}
```

Listing 11 generateDataCenter Acceleo template.

```
[template public generateDataCenter(aDataCenterElement :
    DataCenterElement) ? (aDataCenterElement.numberOfDataCenters>0)]
[aDataCenterElement.dataCenterType.base_Component.name/]
module [aDataCenterElement.dataCenterType.base_Component.name/]{
  ###
  [aDataCenterElement.dataCenterType.rackElements->select(
      rackType.oclIsTypeOf(ComputingRack)).generateRackElement()/]
  [aDataCenterElement.dataCenterType.rackElements->select(
      rackType.oclIsTypeOf(StorageRack)).generateRackElement()/]
  [aDataCenterElement.dataCenterType.generateDataCenterNetwork()/]
  connections allowunconnected:
    dcManager.out --> ned.IdealChannel --> toCloudProvider;
    dcManager.in <-- ned.IdealChannel <-- fromCloudProvider;
    [aDataCenterElement.dataCenterType.rackElements->select(
        rackType.oclIsTypeOf(ComputingRack)).generateRackConnection(
        aDataCenterElement.dataCenterType.network)/]
    [aDataCenterElement.dataCenterType.rackElements->select(
        rackType.oclIsTypeOf(StorageRack)).generateRackConnection(
        aDataCenterElement.dataCenterType.network)/]
}
[/template]
```

Listing 12 generateRackElement Acceleo template.

```
[template public generateRackElement(aRackElement : RackElement) ?
    (aRackElement.numberOfRacks>0)]
[for (index : Integer | Sequence(Integer){1..aRackElement.numberOfRacks})]
[aRackElement.rackType.eClass().name.toLowerFirst()/]_[aRackElement
    .rackType.base_Component.name/]_[index/]:CloudRack;
[/for]
[/template]
```

Listing 13 generateDataCenterNetwork Acceleo template.

```
[template public generateDataCenterNetwork(aDataCenter : DataCenter)]
dataCenterNetwork:DataCenterNetwork{
  gates:
    inComm['['/][aDataCenter.rackElements->select(rackType.oclIsKindOf(
        ComputingRack))->collectNested(numberOfRacks)->sum()/][']'/];
    outComm['['+aDataCenter.rackElements->select(rackType.oclIsKindOf(
        ComputingRack))->collectNested(numberOfRacks)->sum()+']'/];
    inStorage['['+aDataCenter.rackElements->select(rackType.oclIsKindOf(
        StorageRack))->collectNested(numberOfRacks)->sum()+']'/];
    outStorage['['+aDataCenter.rackElements->select(rackType.oclIsKindOf(
        StorageRack))->collectNested(numberOfRacks)->sum()+']'/];
}
[/template]
```

Listing 14 generateRackConnection Acceleo template.

```
[template public generateRackConnection(aRackElement : RackElement,
    network : Bandwidth)  ? (aRackElement.numberOfRacks>0)]
[for (index : Integer | Sequence(Integer){1..aRackElement.numberOfRacks})]
[aRackElement.rackType.eClass().name.toLowerFirst()/]_[aRackElement
    .rackType.base_Component.name/]_[index/].out -->
    {datarate=[network.getBandwidth()/];} --> dataCenterNetwork.inComm++;
[aRackElement.rackType.eClass().name.toLowerFirst()/]_[aRackElement
    .rackType.base_Component.name/]_[index/].in <--
    {datarate=[network.getBandwidth()/];} <-- dataCenterNetwork.outComm++;
[/for]
[/template]
```

Listing 15 Excerpt 2 of the generated *.ned* file.

```
module dataCenter {
  ###
  rackCmp_computingRack_000:CloudRack;
  dataCenterNetwork:DataCenterNetwork {
    gates:
      inComm[1];
      outComm[1];
      inStorage[0];
      outStorage[0];
  }
  connections allowunconnected:
    cloudManager.out --> ned.IdealChannel --> toCloudProvider;
    cloudManager.in <-- ned.IdealChannel <-- fromCloudProvider;
    rackCmp_computingRack_000.out --> {datarate=1.0Gbps} -->
        dataCenterNetwork.inComm++;
    rackCmp_computingRack_000.in <-- {datarate=1.0Gbps} <--
        dataCenterNetwork.outComm++;
}
```