Model driven simulation of elastic OCCI cloud resources

Mehdi Ahmed-Nacer, Slim Kallel, Faiez Zalila, Philippe Merle, Walid Gaaloul

Model Driven Simulation of Elastic OCCI Cloud Resources

MEHDI AHMED-NACER[1], SLIM KALLEL[2], FAIEZ ZALILA[3], PHILIPPE MERLE[3] AND WALID GAALOUL[4]

[1]*University of Sciences and Technology Houari Boumediene, Algiers, Algeria*
[2]*ReDCAD, University of Sfax, Sfax, Tunisia*
[3]*Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 - CRIStAL, F-59000 Lille, France*
[4]*Telecom SudParis, Samovar, IP Paris, France*
Email: m.ahmed.nacer@usthb.dz

Deploying a cloud configuration in a real cloud platform is mostly cost- and time- consuming, as large number of cloud resources have to be rent for the time needed to run the configuration. Thereafter, cloud simulation tools are used as a cheap alternative to test Cloud configuration. However, most of existing cloud simulation tools require extensive technical skills and does not support simulation of any kind of cloud resources. In this context, using a model-driven approach can be helpful as it allows developers to efficiently describe their needs at a high level of abstraction. To do, we propose, in this article, a Model-Driven Engineering (MDE) approach based on the OCCI (Open Cloud Computing Interface) standard metamodel and CloudSim toolkit. We firstly extend OCCI metamodel for supporting simulation of any kind of cloud resources. Afterward, to illustrate the extensibility of our approach, we enrich the proposed metamodel by new simulation capabilities. As proof of concept, we study the elasticity and pricing strategies of Amazon Web Services (AWS). This article benefits from *OCCIware Studio* to design an OCCI simulation extension and to provide a simulation designer for designing cloud configurations to be simulated. We detail the approach process from defining an OCCI simulation extension until the generation and the simulation of the OCCI cloud configurations. Finally, we validate the proposed approach by providing a realistic experimentation to study its usability, the resources coverage rate and the cost. The results is compared with the ones computed from AWS.

Keywords: Amazon Web Services; Cloud Computing; CloudSim; Elasticity; OCCI; OCCIware; Pricing; Simulation

Received 24 Sep 2019; revised 02 Mar 2020

1. INTRODUCTION

Cloud computing has been adopted as the dominant rent model for computing resources [1]. This model delivers Infrastructure-, Platform- and Software-as-a-Service (resp. IaaS, PaaS and SaaS) over the Internet on an easy pay-per-use business model [2]. Even if the model becomes increasingly popular, it remains not reachable for every developer or researcher. Indeed, rent cloud resources over a real platform often comes at a high cost quotient. One suitable alternative is the use of cloud simulation tools. They are particularly helpful to setup the parameters of the cloud resources (such as CPU, RAM, etc.) before deploying it on a real cloud platform and to conduct the necessary experimentation in repeatable and controllable environment free of cost. In this respect, a number of simulation tools are developed for cloud computing environments. However, using the traditional simulation tools lead to several challenges:

1. None of the existing cloud simulation tools are based on a generic model that allows a simulation of any kind of cloud resources.
2. Cloud architects are forced to be aware of the programming language and command line syntax to implement the simulation of their cloud architecture instead of focusing on their cloud concerns and results.
3. Cloud architects have to understand the source code of the cloud simulation tool and how it processes to enrich it with new functionalities. This task is complex, requires an effort of an expert developer and time consuming. While, an effective solution should be modular, extensible and ease the description of the simulation needs regardless the technical specifications of any simulation tool.

For these purposes, cloud architects are looking for software engineering tools that allow simulation, designing, editing, validating, generating, and managing any kinds of cloud resources, with minimum effort and time consuming. The Cloud computing standard Open Cloud Computing Interface (OCCI) [3] and its associated tooling OCCIware [4] provide a generic and a precise resource-oriented metamodel, along with an enhanced tooling

environment called OCCIware Studio [5]. This later is a model-driven tool chain for designing, managing and analyzing any kind of cloud computing resources. It takes the advantages of Model-Driven Engineering (MDE) approach for cloud computing [6] to let both cloud architects and users to efficiently describe their needs at a high level of abstraction. Through OCCIware Studio, the cloud architect can define a cloud domain called *OCCIware extension* while users are able to graphically design a running cloud system called *OCCIware configuration* instance of the defined extension. This configuration is composed of OCCIware resources and links between them.

In this context, extending OCCIware metamodel for supporting cloud simulation can lead the users to design an OCCIware configuration, managing any kind of cloud resources and simulate them with minimum effort.

To do so, we propose to reuse an existing simulation tool, which should be compatible with the proposed approach. The suitability is made in accordance with a set of requirements related to the extensibility, compatibility with OCCIware metamodel and ease to understand its technical skills. A comparison, detailed in the following, of various cloud simulation tools concludes that CloudSim is the most suitable cloud simulation tool for our study.

Thus, to tackle the problems addressed above, the main contributions of this article can be summarized as:

1. OCCIwareSim: The first contribution consists in proposing a model driven simulation approach based on OCCIware metamodel. This contribution extends OCCI concepts and illustrates the advantages of the proposed MDE approach by showing the ease of extending the metamodel to support the simulation of any kind of cloud resources at a high level of abstraction.

2. Simulation Designer: The second contribution consists in providing a framework where the users can graphically design a cloud configuration and run the simulation. This designer gives facilities for designing, editing, validating, generating, and managing the cloud resources conform to the extension concepts.

3. EPriceCloudSim: The third contribution consists in extending CloudSim tool for supporting new features. We propose, as proof of concepts, to extend CloudSim by elasticity and pricing strategies concepts. These two features are important in cloud computing environment [7][8].

4. Evaluation: To demonstrate the efficiency of our approach, we evaluate the principle of interoperability through showing various use cases being supported by our tool. Then, we measure the easiness and efficiency of our tool based on the time taken to complete the design and modify a cloud configuration. We also estimate the cost of the infrastructure and compare the result with the ones computed from Amazon Web Services (AWS) provider.

In our previous work [9, 10], we have proposed an OCCIware simulation extension. However, we have not presented a model driven engineering approach for the simulation with its enhanced tools and the benefits of its use. In addition, since the previous work, the OCCIware metamodel has evolved and a new version has been proposed [5]. In this article, the proposed extension is compatible with the latest version of OCCIware metamodel. Moreover, the simulation tool used, in this article, is improved by two new functionalities. In fact, we include the ability to simulate elastic cloud resources and the price of the consumed resources according to specific pricing strategies. The results will be compared with a real deployment.

The article is structured as follows: Section 2 explains the motivations behind our contribution. Section 3 gives a background on OCCI standard and OCCIware approach. Afterward, in Section 4, we give an overview of the proposed approach. In Section 5, we conduct the first contribution of this paper which consists in proposing an extension of OCCIware metamodel for simulation. We detail, in Section 6 the simulation designer for designing an OCCIware simulation configuration. Afterward, in Section 7, we present the CloudSim extension for supporting the proof of concept capabilities. In Section 8, we validate our proposed approach. We position our work with related approaches in Section 9. Finally, Section 10 concludes on future work and perspectives.

2. MOTIVATION

Consider the scenario where cloud architects have to test and evaluate the performance of an application in a real cloud computing platform. Analysing the performance of this application with various scheduling and allocation policies is extremely challenging and costly. Thus it is more suitable to use cloud simulation tools.

Even though simulation requires a smaller hardware platform for testing and enables easier evaluation of different scenarios, its utilization is difficult for many reasons:

1. Require the developers to have a correct model of the application behavior. This requires extensive technical skills, such as a proprietary configuration script and know-how tooling functionalities. Hence, the users will focus more on the technical skills than their cloud concerns and results.

2. Lack of the simulation tool that allows simulation of any kind of cloud resources (compute, network, storage, cloud federation, etc) with all functionalities. If the architect has a specific configuration, it requires for him an effort and time to extend the source code of the simulation tool.

3. Test the application over the simulation requires the ability to control the whole parameters of the application, with less effort and time during the modification. If the application is not properly modeled, results obtained during simulation may not be achieved once the application is deployed.

4. Need to provide various forms of documentation of their cloud simulation configurations. However, this task is complex and usually made in an ad-hoc manner with the effort of a human developer, which is error-prone and amplifies both development and time costs.

Motivated by these considerations, we believe that there is a need for a **model-driven simulation approach based on cloud standard metamodel** in order to:

1. Allow cloud architects to focus more on their cloud concerns rather than implementation details, without the risk of misunderstandings for the concepts and the behavior that rely under cloud simulation APIs.

2. Simulate any kind of cloud resources as well as in a cloud federation environment. If the cloud architects want to simulate a new resource or a new functionality which are not supported by the cloud simulation tool, the cloud architects have just to extend the metamodel by describing their needs at a high level of abstraction.

3. Enable cloud architects to model, edit and generate their cloud applications to be simulated with minimum effort and time consuming.

4. Generate automatically textual documentation to assist cloud architects, to understand the concepts and the behavior of the cloud simulation extension.

Users of our solution are cloud architects, which are able, with the help of OCCIwareSim, to answer questions such as (i) *"How many resources type can I simulate ?"* (ii) *"How many time and effort can I save using OCCIwareSim against traditional tools ?"*; and (iii) *"How can I easily extend OCCIwareSim metamodel for other capabilities ?"*

3. BACKGROUND

In this section, we provide basic information about the OCCI standard and OCCIware approach, the adopted metamodel to implement our cloud simulation approach.

3.1. OCCI Standard

Open Cloud Computing Interface (OCCI) is an open cloud standard [11] specified by Open Grid Forum (OGF). OCCI defines a RESTful protocol and provides APIs for all kinds of management tasks in cloud environments, including Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). In order to be modular and extensible, OCCI specifications are divided into four categories [12]: (i) OCCI Core [3] , (ii) OCCI Protocols [13], (iii) OCCI Renderings [14] and (iv) OCCI Extensions [12].

Among these OCCI specifications, we are mainly interested in this article in the OCCI Core Model [3] and OCCI extension [12]. These specifications are sufficient to define the basic entities of the simulation tool whether in IaaS, PaaS or SaaS resources. The rest of the specifications are detailed in the Section Appendix A.

OCCI Core: it is an informal definition of the OCCI Core Model [3] proposed as a RESTful-oriented model. This model is an abstraction of real-world resources, including the means to identify, classify, associate and extend those resources. The gray-colored classes in Figure 1 shows the different concepts of OCCI Core Model which describes cloud resources as instances of *Resource* or a sub-type thereof. A *Resource* can be connected to another one using a *Link*. *Resource* and *Link* are sub-types of *Entity*. Each *Entity* instance is typed by a *Kind*. *Kind* is the notion of class/type within OCCI, e.g., Network, Compute, Storage, etc. while *Mixin* is used to associate additional features, e.g., IP address, price, location, to resource/link instances. *Action* represents business specific behaviors, such as start/stop a virtual machine, and up/down a network, etc. *Category* is an abstract base class inherited by *Kind*, *Mixin* and *Action*. Each instance of kind, mixin or action owns a set of *attributes*. *Attribute* represents the definition of a customer-visible property, *e.g.*, the hostname of a machine, the IP address of a network, or a parameter of an action. A fundamental advantage of the OCCI Core Model is its extensibility. Extending OCCI Core Model is possible by defining sub-kinds of the existing ones or by declaring new *Mixins* and applying them on the appropriate kind.

OCCI Extensions: these specifications consist of multiple documents. Each one describes a particular extension of the OCCI Core Model. Various extensions have been proposed such as Infrastructure extension [12], Platform extension [15], etc. In this article, we are mainly interested in the OCCI *Infrastructure* extension. OCCI Infrastructure [12] is an extension of the Core Model to represent the cloud infrastructure layer. It defines compute, storage and network resource types and their associated *Link* types. For more details, see Section Appendix A.

Even if OCCI standard is widely accepted in the cloud computing community, as an open generic standard to manage Everything as a Service (XaaS), it lacks a precise definition of its core concepts [16]. To overcome this issue, OCCIware project [16] was introduced.

3.2. OCCIware

In the OCCIware project[¶] [16], a formal model-driven toolchain to manage everything as a service with OCCI has been provided. Thereby, a new metamodel for OCCI, named OCCIware Metamodel [4] [5] have been proposed. This metamodel provides a precise framework to design correct and well-formed OCCI artifacts. In addition, a model-driven toolchain, named OCCIware Studio [5], built on the top of OCCIware metamodel has been provided to design, verify, and manage Everything as a Service (XaaS) with OCCI.

OCCIware metamodel [4] [5] is a precise metamodel for OCCI. Figure 1 presents the diagram of OCCIware

[¶] http://www.occiware.org/

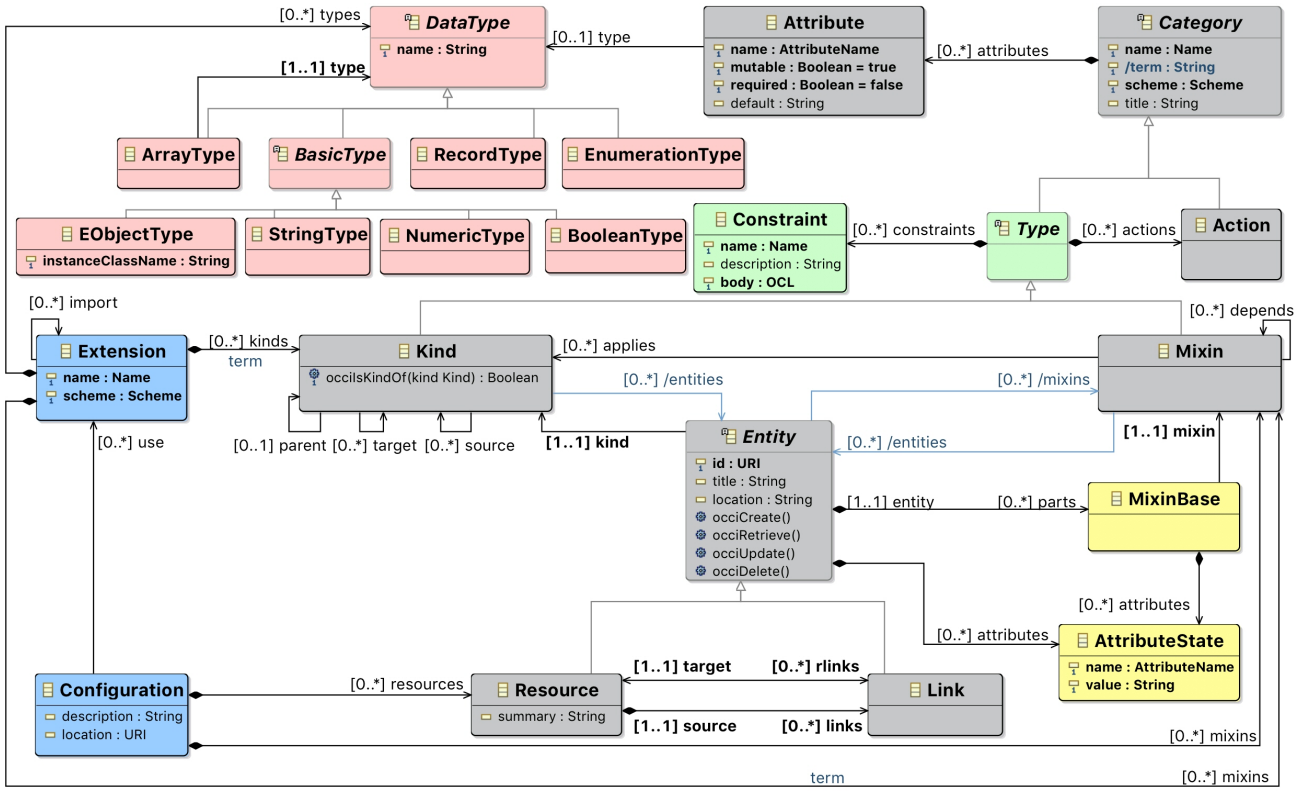


FIGURE 1: Diagram of OCCIware Metamodel [5]

metamodel. This metamodel rigorously defines the static semantics of the OCCI core concepts, which comprises a precise type classification system, an extensible data type system, and both extension and configuration concepts. This metamodel is implemented with Eclipse Modeling Framework (EMF). Beyond the eight concepts defined by the OCCI standard, OCCIware metamodel introduces a set of concepts required to design correct OCCI artifacts. For example, it provides the notion of *Extension* and *Configuration*.

- *Extension* represents an OCCI extension, *e.g.*, infrastructure extension [12], platform extension [15], application extension [15], etc. `Extension` has a `name`, has a `scheme`, has a `description`, has a `specification`, owns zero or more `kinds`, owns zero or more `mixins`, owns zero or more data `types`, and can `import` zero or more extensions.

- *Configuration* is an abstraction of an OCCI-based running system, and is composed of resource and link instances. A configuration must explicitly state which extensions it uses. Modeling a configuration offline could allow designers to think about and analyze their cloud systems without requiring to deploy them actually in the clouds.

Based on OCCIware metamodel, a toolchain named OCCIware Studio [5] has been implemented. It is a set of tools designed to ease both the development and deployment of OCCI-based solutions. This studio is a set of plugins for the Eclipse integrated development environment, which provides various frameworks for simplifying the development of OCCIware studio tools. Through this Studio the users create graphically their extensions and configurations.

In the remainder of this article we use the terms `OCCIware Extension` and `OCCIware Configuration` instead of OCCI extension and OCCI Configuration to refer to the extension and the configuration based on OCCIware metamodel.

4. OVERVIEW OF THE APPROACH

To achieve the predefined aims, we proposed three-phases approach for simulating cloud resource allocations based on model-driven engineering (Figure 2).

In the first phase, we define the extension of OCCIware metamodel for the simulation. Mainly, we perform this phase in two steps:

1. In the first step, we propose the extension of OCCIware metamodel for the simulation by using CloudSim tool. In other words, we give the representation of the CloudSim resources in the OCCIware metamodel. This step allows any cloud architect to enrich our proposed extension by other functionalities at a high level of abstraction. As proof of concepts, in this paper, we

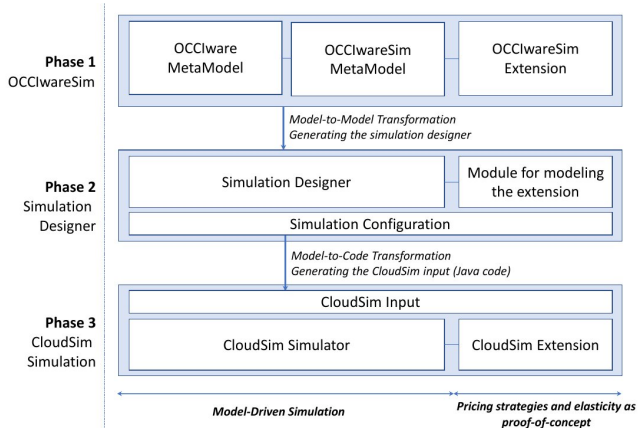


FIGURE 2: Overview of the approach

enrich the proposed extension by the elasticity and pricing strategies concepts.

2. In the second step, we extend OCCIwareSim by the elasticity and pricing strategies concepts. Through this study, we illustrate the ease of extending the OCCIwareSim metamodel, by using the proposed approach, for any kind of cloud resources and for any functionality. By this way, the cloud architect does not require any technical skills since this is done through modeling operations. Thus, he focuses more on the cloud concerns rather than the programming issues.

In the second phase, we generate a simulation designer, with its enhanced MDE toolchain, conform to the proposed extension. This simulation designer is generated based on a set of defined model-to-model transformation rules. Through the designer, the cloud architect can:

- Create, model and edit a cloud configuration and verify its validity. This verification is done in accordance to the predefined simulation extension. For instance, we cannot link a predefined resource with another resource if this is not specified in the simulation extension.

- Generate automatically a textual documentation to assist the cloud architects to understand the concepts and the behavior of the simulation configuration.

Once the simulation configuration is designed and verified, we apply a model to code transformation to generate the source code corresponding to the created configuration.

The third phase is done by the *cloud developer*. In fact, if the proposed extension (in phase 1) is not supported by the simulation tool, the cloud developer needs to implement the functions that support the proposed extension. However, if the cloud architect made an extension that was supported by the simulation tool, the cloud developer will not make any effort since the code source of the configuration is generated automatically. For our study, the elasticity and pricing strategies are not supported by CloudSim tool. To do so, we develop EPriceCloudSim, an extension of CloudSim for supporting the elasticity and the pricing strategies.

Requirement	Reason
Extensibility and Open source	- Let us to add new simulation capabilities
Java Language	- Written in the same language as OCCIware metamodel. - Saves us from adapting the source code of other cloud simulation tools writing in other language, to OCCIware metamodel.
Layer Abstraction and Documentation	- Make it easier for us to understand the source code in order to extend it.
Federation	- Allows the coordination between different cloud providers.

TABLE 1: cloud simulation tool suitability

In the following, we detail each phase described above.

5. OCCIWARESIM

This section proposes the main contribution of this article, OCCIwareSim: a model driven simulation approach based on OCCIware metamodel. We first extend the OCCIware metamodel for simulation. Afterward, we study its extensiblity by enriching it with new simulation concepts.

5.1. OCCIwareSim Metamodel

Before proposing the OCCIwareSim metamodel, we need to conduct a study for choosing the most appropriate simulation tool for our approach. To do so, we have define six requirements summarized in table 1.

After a comprehensive study of almost all cloud computing simulation tools, we conduct a comparison of different cloud simulator tools (See Section 9). We have concluded that CloudSim [17] is the most appropriate simulation tool to achieve our goal.

CloudSim provides four main types of components, which are: *Datacenter*, *Host*, *Virtual Machine* (VM) and *Cloudlet*. Figure 3 presents the OCCIware simulation metamodel. It gives the representation of the CloudSim resources in OCCIware metamodel.

In the following, we present the relationships between the CloudSim entities and the main OCCI resource types.

The *Datacenter* resource type in Figure 3 represents the cloud provider in CloudSim. It is the main hardware infrastructure that provides services to deal with user requests. Datacenter manages a set of hosts (i.e., physical machines). These hosts are interconnected and managed by a set of management policies. Datacenter is represented as a kind extending the *Compute* kind defined in OCCI Infrastructure extension.

Host executes actions related to the management of virtual machines (e.g., creation and destruction). A host has a defined policy for provisioning memory and bandwidth, as well as an allocation policy for processing elements to virtual machines. A host is associated to a datacenter. It can host virtual machines. Host is represented as a kind extending the *Compute* kind defined in OCCI Infrastructure extension.

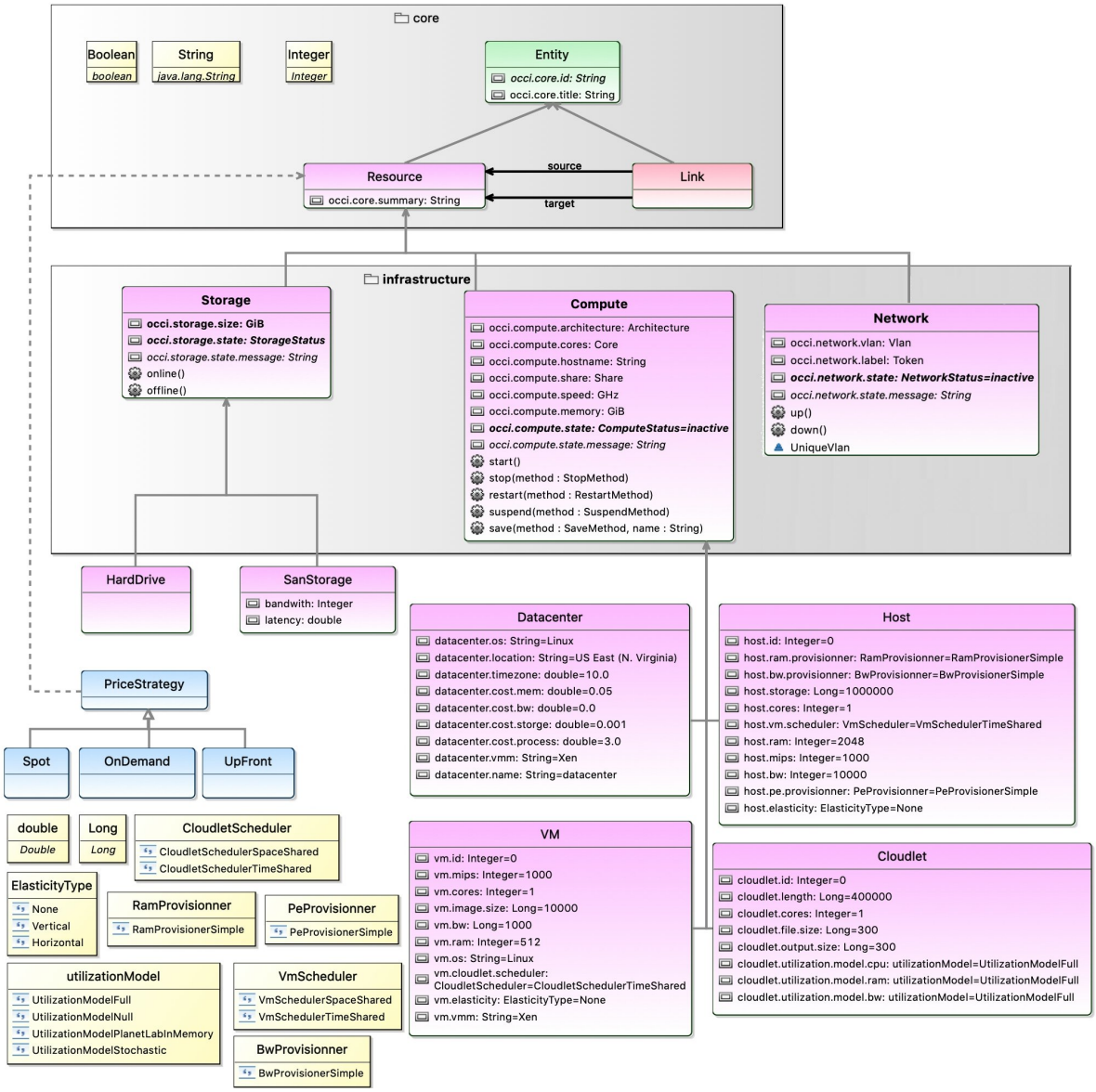


FIGURE 3: OCCIwareSim: Metamodel for simulating any kind of cloud resources

VM runs inside a Host, sharing *hostList* with other VMs. It processes *cloudlets*. This processing happens according to a policy, defined by the *CloudletScheduler*. Each VM has an owner, which can submit cloudlets to the VM to be executed. VM is represented as a kind extending the *Compute* kind defined in OCCI Infrastructure extension.

Cloudlet models the cloud-based application services or the CloudSim task running in VMs. Each cloudlet is defined by the number of CPU operations it requires. Cloudlet is represented as a kind extending the *Compute* kind defined in OCCI Infrastructure extension.

HardDrive represents the storage system and the behaviour of a typical hard drive storage. *HardDriveStorage* is represented as a kind extending the *Storage* kind defined in OCCI Infrastructure extension.

SanStorage represents a storage area network composed of a set of hard disks connected in a LAN. *SanStorage* is represented as a kind extending the *Storage* kind defined in OCCI Infrastructure extension.

In addition to that, we define a set of *DataType*s required to create correct OCCIware artifacts. They allow to precisely instantiate the appropriate values for the attributes of CloudSim entities:

- *CloudletScheduler*: represents the policy of scheduling performed by a VM.
- *VmScheduler*: represents the policy used by a *VM* to share processing power among *VMs* running in a host.

For scheduling, CloudSim offers two types of policies: space-shared and time-shared. In space-shared policy, only one VM (or Cloudlet) can run at a given instance of time. While, in time-shared policy multiple Cloudlets (VMs) can simultaneously share the processing power within VMs (Hosts).

- *RamProvisioner*: represents the provisioning policy of memory to virtual machines inside a host.
- *BwProvisioner*: represents the provisioning policy of bandwidth to virtual machines inside a host.
- *PeProvisioner*: defines native types of processing elements provisioner.

For provisioning, CloudSim provides in its original version a simple best effort allocation policy: if there is an available requested resource (ram, bw, cpu), it allocates. Otherwise, it fails.

- *UtilizationModel*: provides a fine-grained control over resource usage by a Cloudlet. For instance, using all the available CPU capacity.

At this stage, the OCCIwareSim metamodel allows a simulation of all resources supported by CloudSim tool. However, the proposed approach provides a generic metamodel. Thus, the users can add new extensions by giving a representation of their needs in the OCCIwareSim metamodel. In this context, we extend, in the next Section, the OCCIwareSim metamodel for elasticity and pricing strategies.

5.2. OCCIwareSim Extension: Proof of concept

Many studies exist that extend CloudSim for other capabilities such as extensions for elastic media resources [18], for debt-aware learning [18], and by providing a graphical interface [19]. However, these extensions are made in a hard coded manner. It will be difficult to understand the source code, to execute it and to extend it with other features. To illustrate the ease of extending the proposed metamodel, we enrich the OCCIwareSim metamodel by two new extensions: elasticity and pricing strategies. These extensions are considered as proof of concepts. Many studies demonstrate that elasticity and pricing strategies are closely tied and they can provide a significant discount [7][8].

In the following, we first discuss the elasticity concept in cloud computing, and we detail the pricing strategies according to Amazon Web Service (AWS). Afterward, we position these two concepts against OCCIwareSim metamodel.

5.2.1. Elasticity

Elasticity in cloud computing [20, 21] is the degree to which a system is able to adapt to workload changes by provisioning resources in an autonomic manner, such that a cloud service continues running smoothly even when the number or quantity of its utilization scales up or down, thereby avoiding under-utilization and over-utilization of resources. Provisioning of resources can be made using vertical or horizontal elasticity. In horizontal elasticity, when the number of user demands for the resource increases, the provider has to replicate as many copies of resource instances as this number. Similarly, when the number of user demands decreases, the provider has to delete the unused resource instances. In contrast, vertical elasticity consists of changing the resources assigned to an already running VM, for example, increasing (or reducing) the allocated CPU power or the memory.

5.2.2. Pricing strategy

One of the most important challenges of cloud providers is to supply users with good computed services at reasonable prices. However, each cloud provider has its own strategy of pricing. According to this pricing strategy, customers pay in function of time, quantity, or resource's instance type they consume. In this article, we focus on the pricing strategies for VM instances. Networking and database costs are out of the scope of this work. Let us note that we focus on AWS, because it continues to lead the market in terms of maturity and offers the widest range of functionalities, VMs and pricing strategies. AWS instances can be purchased using three main types of pricing strategies:

- On-demand instances: they let the customer pay for compute capacity per-hour or per-second (depending on which instances customers run) with no long-term commitments or upfront payments. The customer increases or decreases the compute capacity depending on the demands of its application and only pays the specified per hourly rates for the instance used.
- Reserved instances: this instance type offers consumers the option to make an one-time reservation for 1 or 3 year terms with in advance payment for an instance. In turn, the customer receives a significant discount for each hour running that instance (up to 75%) compared to on-demand instance pricing.
- Spot instances: Amazon EC2 Spot Instances let customers take advantage of unused EC2 capacity in the AWS cloud. It allows the consumers to bid on spare Amazon EC2 computing capacity. To get the spot instance, customer specifies the maximum price they are able to pay per instance hour based on the spot price history available via the Amazon EC2 API and the AWS Management Console. Spot instances offer a significant discount (until 90%) compared to on-demand pricing. Therefore, consumers can significantly reduce the cost of running their applications and grow the compute capacity of their applications.

5.2.3. Extension's entities in OCCIwareSim Metamodel

One key challenge is how to properly deal with elasticity and pricing strategies in OCCIwareSim metamodel. To resolve that, we have a set of elements to add in the OCCIwareSim metamodel.

For elasticity, we add two attributes in VM and Host to indicate whether these resources are elastic or not and which kind of elasticity are if they are elastic. These attributes are supported by a new provisioning function presented in Algorithm 1.

To deal with pricing strategies in our approach, as shown in Figure 3, the *Price* is defined as a *Mixin* that can be applied to every OCCIware Resource. This choice allows potential users to only define a *Resource* sub-kind to simulate and thus the *Price* mixin will be applied without the need to create a new mixin or edit the OCCIwareSim metamodel. In this way, the storage, the network and the compute resources can be simulated on a specific strategy.

Based on the OCCIwareSim extension designed and validated by OCCIware Studio, we have proposed a model-driven framework called Simulation Designer. This later allows users to graphically design a cloud configuration to be simulated, called also a simulation configuration.

6. SIMULATION DESIGNER

The main goal of the simulation designer consists in designing a correct simulation configuration conforms to OCCIwareSim metamodel. Later, this simulation configuration will be simulated by EPriceCloudSim.

In this section, we firstly detail the transformation rules applied to generate the simulation designer. Afterward, we give an overview of the simulation designer platform.

6.1. Generation of Simulation Designer

Conforming to the OCCIwareSim metamodel, the cloud architect can create simulation configurations. To facilitate this task, it is mandatory to provide him the required tooling to design, check, and simulation its configuration. To do that, we have defined a Simulation Designer, a graphical modeler to create, modify and visualize simulation configurations. It was developed on top of Eclipse Sirius[‖].

The mapping from the OCCIwareSim metamodel to the Sirius tool is straightforward: each OCCI kind in the OCCIwareSim metamodel (`Datacenter`, for example), is mapped into a Sirius container node representing an OCCI resource. Inside, each OCCI attribute is instantiated and can be initialized and edited. In addition, each mixin is mapped into a Sirius container inside the applied OCCI resource. So, the cloud architect can dynamically drag and drop a mixin inside the OCCI resource and remove it. Finally, the different OCCI resources are linked between them using Sirius edges.

6.2. Simulation Designer Platform

The OCCIware simulation designer, generated based on the OCCIwareSim metamodel, allows the creation of simulation configurations and runs the simulation. A screenshot of the simulation designer is presented in Figure A.4.

The cloud architect in front of the simulation designer has the ability to:

- Navigate through the simulation project containing the simulation configurations;
- Provide a graphical representation of simulation configuration. From this panel, the cloud architect can model a simulation configuration by using drag-and-drop operation.
- Provide the Eclipse properties editor for viewing and modifying attributes of a selected modeling element;
- Import an existing configuration or generate a new simulation configuration:
  - A new simulation configuration: this configuration contains instances of the *Compute* or *Storage* sub-kinds defined in the simulation extension, i.e., *Datacenter*, *Host*, *VM* , *HardDriveStorage*, *Cloudlet*, etc. It consists to drag and drop the simulation extension elements into the simulation configuration.
  - From an existing configuration: the user can import an existing generic OCCI configuration in order to be simulated. In this case, the user intervention is needed. Indeed, this OCCI configuration does not contain enough information that allows mapping the existing configuration to a simulation one. In addition, it is difficult to understand the intention of the user about which resource is to be simulated (e.g., a compute resource can be a datacenter, a host, a VM, etc). Thus, we have extended the simulation designer, to allow users to map the different existing resources into simulation ones.

Once the simulation configuration is designed, the simulation designer take the advantage of the MDE approach to:

1. Perform several verification to guarantee the right form of the simulation configuration. These activities ensure that (1) all resources needed by EPriceCloudSim are available in the simulation configuration, (2) each resource imported from an existing configuration is mapped to only one simulation resource (e.g. a compute cannot be a Datacenter and Host at the same time. However, a compute can be mapped to a VM and tagged by OnDemand mixin), and (3) a correct match between the mapping and resources (e.g. a storage cannot be mapped to a Datacenter or a Host).
2. Produce automatically the Java source code related to the entire simulation configuration including all resources, links and attributes.
3. Generate automatically a textual documentation to assist cloud architects to understand the concepts of the simulation extension and configuration.

When the simulation configuration is designed, verified and the source code is generated, the cloud architect has to execute the simulation by selecting *Run Simulation* from the contextual menu. This will call the simulation tool to start the simulation.

[‖] `https://www.eclipse.org/sirius/`

We can consider that the simulation configuration is completely created. Later, it must be completed by the software developers to implement concretely how the generated cloud configuration must be executed on the cloud simulator.

7. EPRICECLOUDSIM: CLOUDSIM EXTENSION

In this section, we demonstrate how we extended CloudSim to support the elasticity and pricing strategies extensions.

7.1. Elasticity

CloudSim provides in its original version a simple best effort allocation policy. For example, allocating primary memory (RAM) to the VMs on host is feasible only if the host has the required amount of free memory. If the request is beyond the available memory capacity, then it is simply rejected. This is similar for other resources such as bandwidth and CPU.

To support the elasticity, we extend CloudSim by a new *Provisioning class*. During the simulation, a function, in the provisioning class, retrieves the elasticity parameters from the simulation configuration and processes in two separate steps: first, it determines which resource exceeds the capacity. In this case, the function determines the elasticity type. If the elasticity is horizontal, a new resource is provisioned. While, in case of vertical elasticity, the capacity of the resource is resized according to the workload demand.

Algorithm 1 presents the function, inside the provisioning class, that allocates the VM to a Host. We present only the verification of the ram capacity. The other capacities (such as storage and bandwidth) are allocated in the same manner. During the allocation, the Host verifies if it has the capacity in RAM to allocate correctly the requested VM. If the verification returns false (in line 2). The algorithm verifies if the Host is elastic and which the elasticity type is. In case the elasticity is horizontal (line 3), the provisioning algorithm calls the procedure `VmCreate` to allocate a new VM on this Host. In case the elasticity is vertical (in line 7), the provisioning algorithm resizes the Host by the RAM requested. Otherwise, the Host fails the allocation of the VM (line 10).

Algorithm 1 VM Allocation in Host

```
1: function VMCREATE(Vm vm): Boolean
2:     if (getRamProvis().allocateRamVm(vm, vm.getRequestedRam() = false ) then
3:         if (vm.getElasticity().equals("Horizontal")) then
4:         CreateVm(this, vm);
5:         return true;
6:         else
7:             if (vm.getElasticity().equals("Vertical")) then
8:            resize(vm.getCurrentRequestedRam());
9:            return true;
10:             else
11:            Log.printLine("Allocation of VM failed by RAM");
12:            CloudSim.error=true;
13:            return false;
14:             end if
15:         end if
16:     else vm.setHost(this); return true;
17:     end if
18: end function
```

Thereby, where the simulation should fail in the basic CloudSim tool due to the lack of elasticity, EPriceCloudSim re-executes the simulation with the new configuration (since a new resource was added or resized).

Once the entire application is constructed and the execution time is computed, we can estimate the real cost of this application under AWS. This leads us to the second extension.

7.2. Pricing Strategies

Basically, CloudSim lets the user to point out the price of each resource. However, the cost estimated is independent of the region, of the simulation moment and of the strategies. Then, the cost estimated cannot be real.

To obtain the real cost according to AWS pricing strategies, we have developed a RESTful web service that retrieves the price of the instances and the data transfer. This service uses in Algorithm 2 the GET requests to retrieve all information about the available resources that matches with the requested parameters. These parameters are a combination of characteristics that can influence the price of the instances such as the region, the operating system, the CPU size, the memory size and the duration. The later parameter should be estimated by EPriceCloudSim in the first step. The rest of the parameters are extracted from the simulation configuration.

If the resource is found, the response is a JSON file that contains the details of the available resources at the moment of the simulation. These details include the instance name (e.g., m3.medium), capacity of the instance in memory and virtual CPU cores, instance id, etc. And the price of each resource under different strategy:

1. The function *OnDemandInstancePrice* uses the instance Id retrieved and returns the price corresponding to the On-Demand strategy,

2. *getInstancePriceReserved* returns a list of prices corresponding to the commitment (1yr or 3 yrs) of the user,

3. *RunSpotPrice* function uses only the instance name and returns a set of information in JSON format stored in the string. These informations detail the cost, the availability instance region and the description of the instance.

Algorithm 2 Cost function of strategies

```
1: function GETPRICE(location, os, cpu, memory, hour): List
2:     price, list: List<Object>;
3:     instanceId, reserved, instanceName, spot: String;
4:     ondemand: Float;
5:     list← getInstancePrice(location, os, cpu, memory, hour);
6:     instanceId ← list.get(3);
7:     ondemand ← OnDemandTreatment.OnDemandInstancePrice(instanceId);
8:     reserved ← ReservedTreatment.getInstancePriceReserved(instanceId);
9:     instanceName ←list.get(0).toString();
10:    spot←SpotHistory.RunSpotPrice(instanceName);
11:    price.add(ondemand, reserved, spot);
12:    return price;
13: end function
```

In case the parameters specified in the GET request do not correspond to any instance, we treat three cases:

1. If the memory size exists but the number of the given *cpu* is not found, the application provides the user with an instance whose *cpu* is just higher than the requested one.

2. If the number of the given *cpu* exists but the memory size is not found, in this case the application provides the user with an instance whose memory size is just greater than the requested one.

3. In case the two previous features do not exist, the application makes available to the user the closest instances in term of number of *cpu* and memory size.

Finally, the total price is computed from the cost of the resource type that matches with the simulation configuration parameters, the strategy and the execution time estimated before.

This application requires a pre-configuration and installation such as the station work configuration to support AWS Command Line Interface (*CLI*)** and an AWS account with its public and private access Key ID.

7.3. Custom Broker

In order to let the users to map from the OCCIware resources to CloudSim resources (in other words, make a translation from OCCIware defined resources to CloudSim resources), we need to extend CloudSim for that. Basically, CloudSim lacks the ability to link (bind) the specified resources defined by the user. It assumes that the physical infrastructure is abstracted from Cloud users/brokers. For instance, it is not possible to create two VMs in two different hosts if only one host can host them. However, in our metamodel we need to give the users the ability to choose where the cloud resources should be placed on. To this end, we need to develop a custom broker and override several methods. Thereby, we extend these two main classes:

1. DatacenterBroker: modifying the way VM provisioning requests are submitted to the hosts in a specific datacenter and the way cloudlets are submitted and assigned to VMs.

2. VmAllocationPolicy: we extend this abstract class to implement our own algorithms for deciding which host a new VM should be placed on.

7.4. Simulation

EPriceCloudSim receives the OCCIware simulation configuration that contains the EPriceCloudSim resources (datacenter, Host, VM and Cloudlets) and links between them (in *Map<Resource, List<Resource>>* format). Each resource is defined by a list of attributes defined in the extension such as the instance capacity (in storage, cpu, bandwidth), the instance type, the desired region, etc. An illustrated example is presented in section Appendix B (Frame (B) of the Figure A.4).

Firstly, EPriceCloudSim constructs the infrastructure of the application from the simulation configuration by using the source code which is generated automatically by the simulation designer. Please note that no line code is written to extract the resources. Then, EPriceCloudSim simulates the execution of the application over the infrastructure constructed. As output, EPriceCloudSim estimates the execution time and the resources needed on the infrastructure. However, the infrastructure can slightly differ from what the users introduce in OCCIware configuration. Indeed, depending on the resource attributes in the OCCIware configuration, the resource can be elastic (vertical or horizontal). Then, some resources can be added to the infrastructure during the simulation. Afterward, EPriceCloudSim calls the REST service to retrieve the real cost, in different strategies, of the infrastructure generated on AWS.

Finally, the simulation results are stored in a log file. This log contains the informations about the execution time of the application and the real deployment cost in different strategies.

The source code of this implementation is available at `https://github.com/mehdiAhmed/OCCIwareSim` and a demonstration video showing the execution of the different steps is available at `https://youtu.be/t0l689YbW_o`.

8. EVALUATION

The experimentation results are based on a series of experiments performed in four ways: resources coverage rate (Section 8.1), capacity of deployment evaluation (Section 8.2), cost evaluation (Section 8.3) and usability evaluation (Section 8.4).

8.1. Resources Coverage Rate

To illustrate the efficiency of the proposed approach on interoperability issue, we perform a quantitative evaluation. In this experiment, we compute the resource coverage rate by our proposed approach of different cloud resource standards : TOSCA, CIMI, and OCCI.

To do so, we acquire 10 configurations retrieved from three different cloud standard specifications : 4 configurations from TOSCA, 2 configurations from CIMI and 4 configurations from OCCI ††. Afterward, we use our simulation designer to construct these configurations and we compute the number of entities that can be represented by our approach, the coverage percentage and the number of code lines that can be reduced.

The results, shown in Figure 4, demonstrate that most of the configurations are fully covered by our simulation designer. Basically, we found that all use cases from

**https://aws.amazon.com/cli/

††available at `http://www-inf.it-sudparis.eu/SIMBAD/tools/linked-cr/usecases`

OCCI and TOSCA are fully generated for simulation by taking into account all the described features. In CIMI, only the user profile is not supported by the simulation designer, which has a coverage of 80%. Therefore, we can conclude, by taking into consideration these experiments, that our proposed approach ensures good standard coverage interoperability by handling representative use cases coming from three of the most used Cloud standards.

Moreover, our experiments prove the advantage of our approach by reducing drastically the user efforts, in terms of code lines. Concretely, without our simulation designer the users may have to write 3900 code lines in total for 15 cloud resources in order to simulate them in CloudSim.

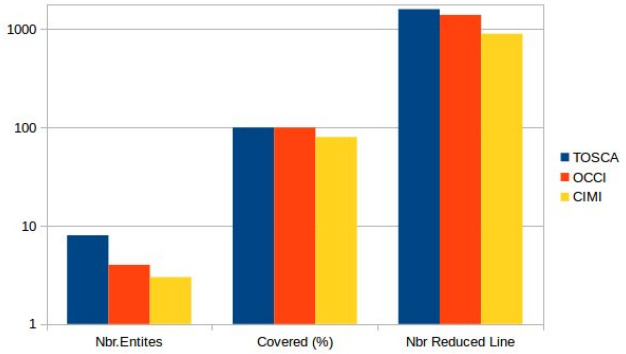


FIGURE 4: Resource coverage rate

8.2. Deployment Capacity

In this evaluation, we try to illustrate, through certain cases, the limits of AWS and the CloudSim tool, and how OCCIwareSim surpasses these limits.

Figure 5 presents the number of accepted and rejected requests of VMs for different configurations, when these instances are deployed in a real AWS platform, in CloudSim tool and in OCCIwareSim. As expected, AWS has a limit in the number of instances deployed by each account in the same region. Indeed, by default, AWS reaches a limit of 20 instances per region for each account. However, the user can request a higher limit by providing information about the new limit and regions where it should be applied. While, using OCCIwareSim or CloudSim the infrastructure is immediately deployed and the application can be simulated.

In addition, requesting 80 *medium* VMs in two regions exceed the limit of the user which is 40VMs in two regions. After requesting AWS for increasing the limit, they increased the limit for 60 VMs instead of 40 VMs. Also, CloudSim fails the simulation since it exceeds the limit of the hosts in the datacenters. This is because CloudSim lacks the elasticity concept.

Moreover, sometimes (depending on the period) AWS does not have sufficient instance type capacity as requested. In this case, AWS fails to launch instances and the user needs to wait a few moments for re-launching the instances. This situation was happened when we have deployed an infrastructure with *r4.16xlarge* instances.

In CloudSim, the initial configuration of the host in the datacenters does not have sufficient capacity to host the requested demand. With a configuration of 20 *r4.16xlarge* instances in one datacenter and with 100 *r4.16xlarge* instances in two datacenter, CloudSim satisfies respectively 14 and 35 instances. Then, the simulation with CloudSim failed.

While, using OCCIwareSim the requested instances are immediately available without waiting period.

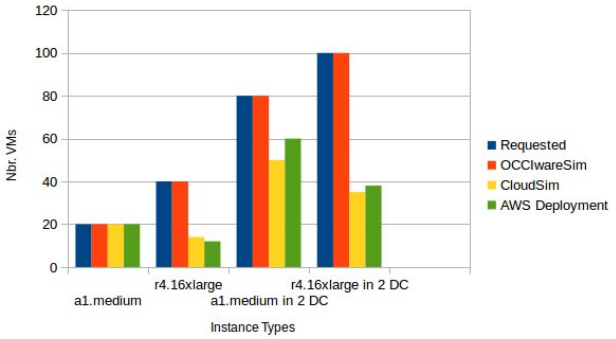


FIGURE 5: Number of acceptance and rejection of VMs in AWS CloudSim and OCCIwareSim

8.3. Cost Evaluation

In this section, we present the evaluation of our proposed approach against the price calculator of Amazon[‡‡]. The purpose of this evaluation is to verify if the cost obtained after simulation is close to the cost proposed by Amazon.

Amazon offers the customers a useful tool called AWS Calculator to model their applications, estimate their costs before the deployment and determine the best and worst case scenarios. This tool incorporates the latest pricing changes across all Amazon services in all regions. However, the infrastructure in AWS Calculator is abstracted to the end users. While, the simulation designer tool allows the binding between resources. Thus, the end users have more detailed access to the infrastructure. In addition, AWS Calculator returns the cost according to only one specific strategy. While, OCCIware simulation designer allows a cost comparison of different strategies in only one running simulation.

Table 2 presents the cost of a real application deployment by using OCCIware simulation designer and AWS Calculator. According to Section 5.2.2 and to AWS pricing strategies, we compute for each OCCIware configuration, three payment strategies: i) On-demand, ii) Reserved Instances and iii) Spot Instances. For On-demand strategy, we compute the pricing per hour. For Reserved Instances three possibilities are given:

1. All Upfront: The customers pay for the entire Reserved Instance term with one upfront payment.

[‡‡]http://aws.amazon.com/calculator

2. Partial Upfront: The customers pay for a portion of the Reserved Instance upfront, and then pay for the remainder over the course of one year term. This option balances the RI payments between upfront and hourly.

3. No Upfront: The customers pay nothing upfront but commit to pay for the Reserved Instance over the course of the Reserved Instance term.

In AWS platform, the customers have to choose the term of one or three years. For Spot Instance strategy, we present the saving cost over On-Demand strategy given by the Spot instance advisor analysis [§§]. In order to compare the different strategies and validate our tool, we present in Table 2 the total cost (per 1 year) of various configurations estimated by OCCIware simulation designer and AWS Calculator on different strategies. To obtain the presented results, we simulate the same configuration over the same regions by using OCCIware simulation designer and AWS calculator. We performed our configuration on *m3.medium* machines, that has installed GNU/Linux in the US East (N. Virginia and Ohio).

In Table 2, the cost estimated in both tools are quite similar for all configurations. For On-Demand strategy, the result obtained is the same, since both tools use the same API pricing and it rarely changes. The customers choosing On-Demand pricing pay a low cost per hour without any up-front payment. But using On-Demand for any sustained periods generates unexpected AWS costs at the end of the year. For Upfront pricing, OCCIware simulation designer also obtains the same results as AWS calculator. However, the OCCIware simulation designer gives more details on Partial-Upfront. Indeed, OCCIware simulation designer specifies what the customers pay at the up-front and how they pay by hours. While, AWS Calculator gives only the up-front price. By using up-front strategy, the customers get the best effective hourly price compared to On-Demand.

For Spot instance strategy, OCCIware simulation designer retrieves the saving cost against On-Demand strategy from AWS API. Compared to the result obtained by Spot instance advisor, there is a slight difference between them. This is due to the period of the experiment and the frequency of the interruption at this period. However, OCCIware simulation designer gives more informations (in Figure B.1) then Spot instance advisor. If the customers need an EC2 instance which is not available at the requested region, OCCIware simulation designer suggests another region and gives some information about this instance.

In addition, when the customers ask specific EC2 machines in different datacenters located in different regions, the instances may not exist in these regions. The customers need to look for another alternative such as searching for other type of machines close to the previous one. This is what we observe in Table 2 with two datacenters (Virginia and Ohio). In fact, the customers need *m3.medium* instances. This type of instance is not available in Ohio. Then, the experiment fails until the customers find another type of instance or look for another region. However, OCCIware simulation designer automatically proposes to the user the closest instance than the instance requested. In our experiment, OCCIware simulation designer proposes for the customers *r3.large* in Ohio region instead of *m3.medium*.

From these results, we can conclude that using OCCIware simulation designer, the users have:

- The possibility to estimate the real cost before deploying the application on the real cloud. In this way, it will reduce the deployment costs and help in avoiding failures.
- The detailed access to the infrastructure by choosing the resource placement and by binding between these resources. While, AWS does not provide this ability.
- The detailed comparison of the application cost with different strategies. While, in AWS calculator the same result are obtained after several experiments since the results are returned by strategy.
- The result is returned and never fails. If the users need for a resource which is not available at the requested region, OCCIware simulation designer suggests another resource close to the requested one. While, in AWS the deployment fails.

8.4. Usability Evaluation

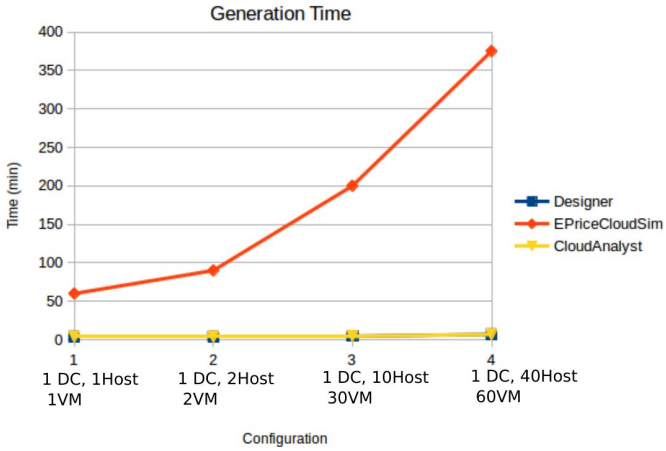
The purpose of this evaluation is to study the usability of OCCIware simulation designer approach against EPrice-CloudSim tool and CloudAnalyst [19] (CloudSim extension with a graphical user interface). To do so, we ask five master students that have knowledge in CloudSim simulation tool and in Java programming to produce some configurations by using EPriceCloudSim, OCCIware simulation designer and CloudAnalyst. Once they finish the configuration, we ask them to change some parameters such as the regions of VMs and the size of attributes, add some abilities (add a new pricing strategy) and re-execute the simulation. After that, we compute the average of the time spent to produce the configuration, the number of code lines in Java and the average time spent to make changes in the configurations. The results are presented respectively in Figures 6a, 6b and 6c.

OCCIware simulation designer and CloudAnalyst offer to the end user a graphical interface where from they generate the configurations and run the simulation. The users do not need to be good Java programmers or have knowledge in CloudSim. The users have just to slide the resource proposed on the panel and the code is generated automatically. While, without this interface the users lose time for generating the configuration, need to write a lot of Java code lines and need to have knowledge in CloudSim. Figures 6a and 6b present the time spent and the number of lines written in Java to generate the configuration described in x-axis of each figure. Using an OCCIware simulation designer the user can save up to six hours of time and more than 900 of code lines. To add new functionalities, the users have to write some code lines in CloudAnalyst to add this functionality on the graphical

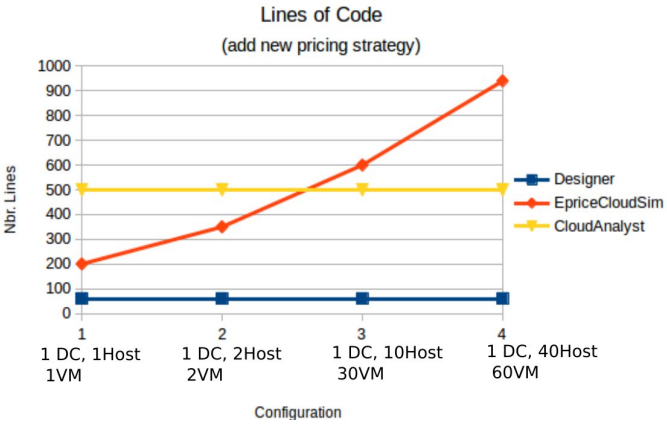
[§§] https://aws.amazon.com/ec2/spot/instance-advisor

			OCCIwareSim		AWS Calculator	
Configuration	Billing		Price ($)	Total/1 Yr	Price ($)	Total /1 Yr
1DC, 1 Host 1VM Virginia	on-demand/price per hour		0.067/h	588.60	0.067/h	588.60
	Upfront 1 Yr	No	420.48	420.48	420.48	420.48
		Partial	211 upfront and 0.017/h	359.92	359.92	359.92
		All	353.00	353.00	353.00	353.00
	Spot instance (Saving over On-Demand)		reduce 82%	105.948	reduce 85%	88.29
1DC, 2 Host 4VM Virginia	on-demand/price per hour		0.268/h	2354.40	0.268/h	2354.40
	Upfront	No	1681.92	1681.92	1681.92	1681.92
		Partial	844.00 upfront, and 0.068/h	1439.68	1439.68	1439.68
		All	1412.00	1412.00	1412.00	1412.00
	Spot instance (Saving over On-Demand)		reduce 81%	447.33	reduce 85%	353.16
1DC, 10 Host 30 VM Virginia	on-demand/price per hour		2.01/h	17658.00	2.01/h	17658.00
	Upfront	No	12614.40	12614.40	12614.40	12614.40
		Partial	6330.00 upfront and 0.51/h	10797.60	10797.60	10797.60
		All	10590.00	10590.00	10590.00	10590.00
	Spot instance (Saving over On-Demand)		reduce 79%	3708.18	reduce 85%	2648.70
2DC, 40 Host 60VM Virginia and Ohio	on-demand/price per hour		9.96/h	87249.60	-	-
	Upfront	No	55188.00	55188.00	-	-
		Partial	30840.00 upfront and 5.40/h	47133.60	-	-
		All	46320.00	46320.00	-	-
	Spot instance (Saving over On-Demand)		reduce 88%	10469.95	-	-

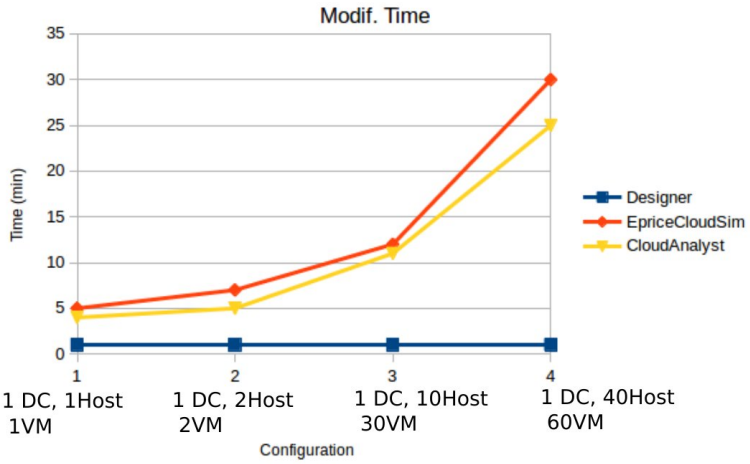
TABLE 2: Cost Evaluation



(a) Generation Time



(b) Lines of Code



(c) Modification Time

FIGURE 6: Usability Evaluation

interface. Since OCCIware simulation designer is based on the MDE approach, extending a cloud simulation with new functionalities is done directly from the designer. Thereby, the users save on the code line.

Figure 6c presents the time spent to modify a configuration. The modifications are related to the region of Vms or capacity of the resources. Using EPriceCloudSim, the users need to verify the links between resources and ensure that the datacenters and Host can host correctly the VM instances. Otherwise, many modifications are made. Also, CloudAnalyst does not support the modification of all resources except datacenters (for instance, the Host parameters, the VMs parameters, etc). Then, using CloudAnalyst requires effort from the cloud users. However, using OCCIware simulation designer, the users do not have lines to change or add, and no time spent for modifications. Indeed, the modifications are made through a graphical interface with a simple click and the code is automatically generated. This gain is due to the utilization of a model-driven engineering approach based on a cloud standard (OCCIware) metamodel.

9. RELATED WORK

In this section, we study the previous works related to the proposed approach. To do so, we firstly compare the existing cloud simulation tool against a set of requirements presented in Section 7. This study conducted us to choose CloudSim tool for the proposed metamodel. In the second time, we position our approach against other solutions that have been proposed.

Cloud simulation tools

Table 3 presents the details of comparative study of different cloud simulators based on the evaluation criteria mentioned in Section 7.

SimGrid [22] is a generic framework for simulation of distributed applications in Grid platforms. It is mostly written in C and has dependencies with Python and Perl. SimGrid is open source and can be extensible. From early 2009 up until today, SimGrid has been extended to P2P, HPC and to support Cloud federation [22].

GreenCloud [23] is a packet-level simulator that uses the Network Simulator 2 (NS2) libraries for energy-aware data centers. It is written in C++ and OTcl, two different languages must be used to implement one single experiment. Since GreenCloud is used quite less as compared to other cloud simulators, there is no extension of GreenCloud to our best knowledge and it does not support a cloud federation.

iCanCloud [24] is a software simulation framework for large storage networks. iCanCloud has been developed in C++ on the top of OMNeT++ and INET frameworks. The tool is open source and an extensible simulator. New components can be added to the repository of iCanCloud to increase the functionality of the simulation platform. The disadvantages of iCanCloud are firstly, only Cost per Performance (C/P) modeling of cloud computing environments is simulated or validated and secondly, it models and simulates only EC2 (Elastic Compute Cloud) environments. In addition, iCanCloud does not support the simulation in a cloud federation environment.

GridSim [25] provides facilities for the modeling and simulation of resources and network connectivity with different capabilities, configurations, and domains. GridSim is open source, implemented in Java [26], supports the simulation of multiple cloud providers and is extensible. Many extensions of GridSim were developed such as [27, 28, 29]. Even if GridSim is a powerful simulator, it does not explicitly define any specific application model. In addition, the tool is developed initially for Grid computing systems.

Although the aforementioned toolkits are capable of modeling and simulating cloud application management behaviors, none of them are able to clearly isolate the multi-layer service abstractions (SaaS, PaaS, and IaaS) differentiation required by Cloud computing environments.

CloudSim [17] is an open source simulation application which enables seamless modeling, simulation, and experimentation of cloud computing and application services. It is the most popular simulation tool available for cloud computing environment. CloudSim is implemented at the next level, by programmatically extending the core functionalities exposed by the GridSim layer [30]. CloudSim is extensible and many features can easily be added enabling the modeling of new types of applications, not supported by CloudSim.

We have concluded that CloudSim [17] is the most suitable and has the richest cloud libraries functionalities. It is open source and has been developed in the java programming language which makes it compatible with OCCIware tooling. In addition, CloudSim is based on a modular design that can easily be extended. Moreover, CloudSim supports the Cloud computing federation and it is widely used in cloud research. This has led to the availability of documentation that guide us to understand and implement policies in CloudSim straight away. For these reasons, we have chosen the CloudSim tool to achieve our goal.

MDE approaches

In the second study, we looked for simulators employing the MDE approaches and based on a cloud standard metamodel. Since we have conducted a case study on elasticity and pricing strategies, we also investigated the simulators that treat the elasticity and pricing strategies. A synthesis of the works that we have analyzed are described in Table 4.

Simulation Program for Elastic Cloud Infrastructures (SPECI) [31] is a simulation tool that enables exploration of scaling properties of large data centers. This simulation tool is composed of two packages, one refers to the data center layout and topology, and the other one consists of the components for experiment execution and measuring. The authors study how the overall DC can perform with increasing numbers of components. According to the described requirements, the authors introduce only the aspect of elasticity.

In the same context, PERFSCALE [32] simulation tool allows simulating the behavior of an elastic application

	Extensibility	Open source	Language	Layer Abstraction	Federation	Documentation
SimGrid	Yes	Yes	C	No	Yes	Limited
GreenCloud	Limited	Yes	C++/OTcl	No	No	Limited
iCanCloud	Limited	Yes	C++	No	No	Limited
GridSim	Yes	Yes	Java	No	Yes	Limited
CloudSim	Yes	Yes	Java	Yes	Yes	Yes

TABLE 3: Comparison of cloud simulators

executing in the cloud, under the identified phenomena and variable workload. The authors study the elasticity and the cost of running applications under concrete elasticity. However, PERFSCALE tool does not provide the pricing policies. In addition, their implementation neither employs the MDE solutions nor based on metamodel cloud standard.

Moreover, a few works have been done [33, 18, 34] on extending CloudSim for elasticity. In [33] the authors propose a technical debt-aware learning approach for autonomous elasticity management based on a reinforcement learning of elasticity debts in resource provisioning. The authors evaluate their approach by extending CloudSim. Moreover, CloudSim was extended for the elasticity of the media services in [18]. The authors investigated the elastic media distribution and extended CloudSim for the validation of their algorithm. In [34] the authors proposed a generic simulation framework based on the CloudSim toolkit for the validation and evaluation of a Cloud service broker deployed on an Intercloud environment. The authors implemented, based on the open source Java implementation for OCCI called OCCI4Java [35], an OCCI frontend for CloudSim. In this way, the entire communication between broker and providers is forwarded to the native CloudSim DatacenterBroker class through standard OCCI-interfaces. However, the OCCI was just used as an abstract Cloud API that allows the broker to act as OCCI client against the Intercloud Gateway, which runs as OCCI-server on the provider side.

CloudAnalyst [19] extends also CloudSim simulation tool by providing a GUI where from the users can design a cloud configuration. However, CloudAnalyst is not based on the MDE approach and cloud standard metamodel. Thereby, it does not take the advantages of MDE approach for cloud to let both cloud architects and users to efficiently describe their needs at a high level of abstraction.

Although these works extend CloudSim tool but they do not employ the MDE approach and they are not based on any cloud standard metamodel. In addition, they do not deal with the pricing strategies.

The only work published for modeling a cloud simulation configuration by using MDE approach, for the best of our knowledge, was in [36]. The authors have proposed CloudMF tool for facilitating the simulation and the deployment of multi-cloud configuration. The tool was developed upon MODACloudML metamodel [45]. Thus, CloudMF meets only requirement related to solutions based on the MDE approach.

Concerning the simulation of the price of cloud resources, several works have been done for pricing models in cloud computing environment [46, 47, 48]. However, in the analyzed papers, we did not find any works that provide a tool that allows the users to analyze the prices of the resources and services dynamically. Only recently, Alves et al. [37] have proposed an extension of CloudSim (called CMCloudSim) features to model and simulate the cost of cloud resources from Amazon, Google and Microsoft Azure. The proposed extension retrieves the price from the website of these providers. This extension does not support the different pricing strategies of Amazon (i.e., On-Demand, Reserved Instances, and Spot Instances). It focuses only on On-demand strategy. In addition, this extension cannot estimate the price of vertical or horizontal elasticity action. Moreover, the extension does not employ the MDE approach and it does not extend any cloud standard metamodel.

Also, we did not find any works based on MDE approach for the simulation of elasticity and pricing strategies based on other standard' resources such as TOSCA [49] or CIMI [50]. The few works that exists [40, 41, 42] propose a private simulation tool for evaluating their approaches on topologies, power consumption, latency and so on. In other papers [43, 44] the authors treat also the simulation by using TOSCA standard. But, the proposed solutions simulate the management operations in TOSCA services templates and they do not evaluate the elasticity or the cost of the infrastructure.

In the other hand, some works were made in the same area (study the prince of the cloud resources and their elasticity) such as SimGrid [38], B. Javadi et al.[39] and EMDCloudSim [18]. However, none of these tools are based on the MDE approach and cloud standard which are the basic idea of this article.

To the best of our knowledge, there is currently no previous attempt to extend OCCI, TOSCA or CIMI metamodel for simulation. In our previous work [9, 10], we have proposed an extension of OCCIware metamodel for the simulation. However, we have not presented the MDE approach with its enhanced tools and the benefits of its use. In addition, the OCCIware metamodel has evolved and a new version has been proposed [5]. While, in this article, the proposed extension is compatible with the latest version of OCCIware metamodel. Moreover, the simulation tool used, in this article, is improved by supporting the elasticity and the pricing strategies. All this has not been studied in our previous work.

The proposed MDE approach for cloud simulation based on OCCIware metamodel in this article is an initial step in this direction.

Studied Solutions \ Criteria	MDE-based	Cloud Standard	Elasticity Strategies	Price strategies
SPECI [31] No	No	Yes	No	
PERFSCALE [32]	No	No	Yes	No
CloudSim extension [33, 18, 34]	No	No	Yes	No
CloudMF [36]	Yes	No	No	No
CMCloudSim [37]	No	No	No	Yes
SimGrid [38], B. Javadi et al[39]	No	No	No	Limited
EMDCloudSim [18]	No	No	Yes	Limited
OtherSim[40, 41, 42, 43, 44]	No	Yes	No	No

TABLE 4: Cloud simulation solution against described capabilities

10. CONCLUSION

Nowaday, many simulation tools were proposed for cloud computing resources simulation. However, none of them, to our knowledge, propose simultaneously a model-based and standard-based approach to simulate any kind of cloud resources. OCCIware proposes a generic model and an API for managing any kind of cloud computing resources. It has been applied in several domains such as infrastructure, platform, robotic and so on. Unfortunately, no OCCIware-compliant implementation has been proposed to deal with the simulation in the cloud. On the other side, various simulation tools were proposed in cloud computing environments. However, they are not based on a generic model that allows a simulation of any kind of cloud resources and their extensibility require a huge technical effort.

In this article, we have proposed a model-driven engineering approach based on the OCCIware metamodel and CloudSim toolkit. This approach takes the advantage of the MDE approach by allowing the simulation of any kind of cloud computing resources and by enabling the users to extend the simulation tool for other capabilities in a high level of abstraction and to model, edit and generate their application with minimum effort and time consuming.

To achieve our goal, we have conducted our study through three steps. In the first step, we have proposed an extension of OCCIware metamodel for the simulation. Do to that, we have conducted a deep study for selecting the most suitable cloud simulation tool for OCCIware metamodel. We found that CloudSim tool is the most suitable since it meets our proposed requirements. Then, we have proposed OCCIwareSim metamodel: the extension of OCCIware metamodel for the simulation by using CloudSim tool. After that, to illustrate the extensibility of our proposed approach, we have presented a proof of concept study. This later consists in extending OCCIwareSim metamodel for the elasticity and pricing strategies.

In the second step, we have proposed a model-driven framework called simulation designer, with its enhanced MDE toolchain, conform to the proposed extension. Through the simulation designer, the cloud architects can graphically design, modify and visualize a simulation configuration. Once the simulation configuration is completed, the simulation designer takes the advantage of the MDE approach to perform several verification and to generate its source code. This later is the input of EPriceCloudSim tool.

In the third step, we have proposed EPriceCloudSim. This simulation tool enriches the basic CloudSim tool by the elasticity and by providing a RESTful API to estimate a real cost from AWS provider.

We have validated our proposed approach by a set of evaluations. The first evaluation demonstrates the efficiency of the proposed approach on interoperability issues. It computes the resource coverage rate of different cloud standard configurations by OCCIwareSim. The second evaluation illustrates, through certain cases, the limits of AWS and the CloudSim tool and how the proposed approach surpasses these limits. The third evaluation demonstrates that the cost estimated by the proposed approach is almost the same as the real price from AWS platform, as well as it has more advantage. The last evaluation studies the usability of our proposed approach. It found that the user requires less effort and time with a little programming and deployment effort to model and simulate their cloud applications.

The next research steps will be to introduce resource failures and temporal constraints for reserving the instances. This leads to study the allocation optimization and to implement a scheduling algorithm to find the best among all possible allocations that respect a set of constraints (RAM, CPU and time). We will also enrich OCCIwareSim by including network topologies and other pricing strategies from other providers.

ACKNOWLEDGEMENTS

This work is partially supported by OCCIware, a research project funded by French FSN (Fonds national pour la Société Numérique) program.

11. AVAILABILITY

Readers can find the open source code of OCCIware simulation tool on `https://github.com/occiware/OCCIwareSim`

REFERENCES

[1] Armbrust, M. et al. (2010) A view of cloud computing. *Communications of the ACM*, **53**, 50–58.

[2] Mell, P. and Grance, T. (2009) The nist definition of cloud computing. national institute of standards and technology. *Information Technology Laboratory, Version*, **15**, 2009.

[3] Metsch, T., Edmonds, A., Nyren, R., and Papaspyrou, A. (2010) Open cloud computing interface–core. Technical

report. In The Open Grid Forum Document Series.

[4] Merle, P., Barais, O., Parpaillon, J., Plouzeau, N., and Tata, S. (2015) A precise metamodel for open cloud computing interface. *Proceedings of the 8th IEEE International Conference on Cloud Computing*, New York City, NY, USA, June, pp. 852–859. IEEE.

[5] Zalila, F., Challita, S., and Merle, P. (2017) A model-driven tool chain for occi. *Proceedings of the On the Move to Meaningful Internet Systems. OTM 2017 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE*, Rhodes, Greece, October, pp. 389–409. Springer.

[6] Bruneliere, H., Cabot, J., and Jouault, F. (2010) Combining model-driven engineering and cloud computing. *Proceedings of the 4th Workshops on Modeling, Design, and Analysis for the Service Cloud co-located with the 6th European Conference on Modelling Foundations and Applications*, Paris, France, June, pp. 1–8. Citeseer.

[7] Yi, S., Kondo, D., and Andrzejak, A. (2010) Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud. *Proceedings of the IEEE International Conference on Cloud Computing*, Miami, FL, USA, July, pp. 236–243. IEEE Computer Society.

[8] Genaud, S. and Gossa, J. (2011) Cost-wait trade-offs in client-side resource provisioning with elastic clouds. *Proceedings of the IEEE International Conference on Cloud Computing*, Washington, DC, USA, July, pp. 1–8. IEEE Computer Society.

[9] Ahmed-Nacer, M., Gaaloul, W., and Tata, S. (2017) Occi-compliant cloud configuration simulation. *Proceedings of the IEEE International Conference on Edge Computing*, Honolulu, HI, USA, June, pp. 73–81. IEEE Computer Society.

[10] Ahmed-Nacer, M. and Tata, S. (2016) Simulation extension for cloud standard occiware. *Proceedings of the 25th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Paris, France, June, pp. 263–264. IEEE Computer Society.

[11] Edmonds, A. and Metsch, T. (2011) Open cloud computing interface-restful http rendering. Technical report. Platform Computing and Intel.

[12] Metsch, T., Edmonds, A., and Parák, B. (2010) Open cloud computing interface-infrastructure. Technical report. In The Open Grid Forum Document Series.

[13] Edmonds, O.-W. A., ICCLab, Z., Thijs Metsch, I., and Boris Parák, C. (2013) Open cloud computing interface-http protocol. Technical report. In The Open Grid Forum Document Series.

[14] Nyrén, R., Feldhaus, F., Parák, B., and Sustr, Z. (2016) Open cloud computing interface–json rendering. Technical report. In The Open Grid Forum Document Series.

[15] Yangui, S. and Tata, S. (2016) An occi compliant model for paas resources description and provisioning. *The Computer Journal*, **59**, 308–324.

[16] Parpaillon, J., Merle, P., Barais, O., Dutoo, M., and Paraiso, F. (2015) Occiware - A formal and tooled framework for managing everything as a service. *Proceedings of the Projects Showcase, part of the Software Technologies: Applications and Foundations 2015 federation of conferences*, L'Aquila, Italy, July, pp. 18–26. CEUR-WS.org.

[17] Buyya, R., Ranjan, R., and Calheiros, R. N. (2009) Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities. *Proceedings of the 2009 International Conference on High Performance Computing & Simulation*, Leipzig, Germany, June, pp. 1–11. IEEE.

[18] Xavier, R., Moens, H., Volckaert, B., and Turck, F. D. (2015) Design and evaluation of elastic media resource allocation algorithms using cloudsim extensions. *Proceedings of the 11th International Conference on Network and Service Management*, Barcelona, Spain, November, pp. 318–326. IEEE Computer Society.

[19] Wickremasinghe, B., Calheiros, R. N., and Buyya, R. (2010) Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications. *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications*, Perth, Australia, April, pp. 446–452. IEEE.

[20] Herbst, N. R., Kounev, S., and Reussner, R. (2013) Elasticity in cloud computing: What it is, and what it is not. *Proceedings of the 10th International Conference on Autonomic Computing*, San Jose, CA, June, pp. 23–27. USENIX Association.

[21] Al-Dhuraibi, Y., Paraiso, F., Djarallah, N., and Merle, P. (2018) Elasticity in cloud computing: State of the art and research challenges. *IEEE Trans. Services Computing*, **11**, 430–447.

[22] Quinson, M. (2009) Simgrid: a generic framework for large-scale distributed experiments. *Proceedings of the 9th International Conference on Peer-to-Peer Computing*, Seattle, Washington, USA, September, pp. 95–96. IEEE.

[23] Kliazovich, D., Bouvry, P., and Khan, S. U. (2012) Greencloud: a packet-level simulator of energy-aware cloud computing data centers. *The Journal of Supercomputing*, **62**, 1263–1283.

[24] Núñez, A., Vázquez-Poletti, J. L., Caminero, A. C., Castañé, G. G., Carretero, J., and Llorente, I. M. (2012) icancloud: A flexible and scalable cloud infrastructure simulator. *Journal of Grid Computing*, **10**, 185–209.

[25] Buyya, R. and Murshed, M. (2002) Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and computation: practice and experience*, **14**, 1175–1220.

[26] Howell, F. and McNab, R. (1998) Simjava: A discrete event simulation library for java. *Simulation Series*, **30**, 51–56.

[27] Albodour, R., James, A., and Yaacob, N. (2010) An extension of gridsim for quality of service. *Proceedings of the 2010 14th International Conference on Computer Supported Cooperative Work in Design*, Shanghai, China, April, pp. 361–366. IEEE.

[28] Caminero, A., Sulistio, A., Caminero, B., Carrión, C., and Buyya, R. (2007) Extending gridsim with an architecture for failure detection. *Proceedings of the 13th International Conference on Parallel and Distributed Systems*, Hsinchu, Taiwan, December, pp. 1–8. IEEE.

[29] Albın, J. L., Lorenzo, J. A., Cabaleiro, J. C., Pena, T. F., and Rivera, F. F. (2007) Simulation of parallel applications in gridsim. *Proceedings of the Iberian Grid Infrastructure Conference*, Santiago de Compostela, Spain, May, pp. 208–219. IBERGRID.

[30] Rodrigo, N., Rajiv, R., César, R., and Rajkumar, B. (2009) Cloudsim: a novel framework for modeling and simulation of cloud computing infrastructures and services. *CloudSim ICCP*, **1**, 1–9.

[31] Sriram, I. (2009) Speci, a simulation tool exploring cloud-scale data centres. *Proceedings of the First International Conference on Cloud Computing*, Beijing, China, December, pp. 381–392. Springer.

[32] Perez-Palacin, D., Mirandola, R., and Scoppetta, M. (2016) Simulation of techniques to improve the utilization of cloud elasticity in workload-aware adaptive software. *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, Delft, The Netherlands, March, pp. 51–56. ACM.

[33] Mera-Gómez, C. J., Ramírez, F., Bahsoon, R., and Buyya, R. (2017) A debt-aware learning approach for resource adaptations in cloud elasticity management. *Proceedings of the 15th International Conference on Service-Oriented Computing*, Malaga, Spain, November, pp. 367–382. Springer.

[34] Jrad, F., Tao, J., and Streit, A. (2012) Simulation-based evaluation of an intercloud service broker. *Proceedings of the 3rd International Conference on Cloud Computing*, Nice, France, July, pp. 140–145. Citeseer.

[35] (2012). Occi4java java-based occi implementation. `http://occi-wg.org/tag/java/`.

[36] Ferry, N., Chauvel, F., Song, H., Rossini, A., Lushpenko, M., and Solberg, A. (2018) Cloudmf: Model-driven management of multi-cloud applications. *ACM Trans. Internet Techn.*, **18**, 16:1–16:24.

[37] Alves, D. C., Batista, B. G., Filho, D. M. L., Peixoto, M. L. M., Reiff-Marganiec, S., and Kuehne, B. T. (2016) CM cloud simulator: A cost model simulator module for cloudsim. *Proceedings of the IEEE World Congress on Services*, San Francisco, CA, USA, June-July, pp. 99–102. IEEE Computer Society.

[38] Desprez, F. and Rouzaud-Cornabas, J. (2013) Simgrid cloud broker: Simulating the amazon aws cloud. Technical report. INRIA, Research Centre Grenoble, France.

[39] Javadi, B., Thulasiram, R. K., and Buyya, R. (2011) Statistical modeling of spot instance prices in public cloud environments. *Proceedings of the IEEE 4th International Conference on Utility and Cloud Computing*, Melbourne, Australia, December, pp. 219–228. IEEE.

[40] Sampaio, A., Rolim, T., Mendonça, N. C., and Cunha, M. (2016) An approach for evaluating cloud application topologies based on TOSCA. *Proceedings of the 9th IEEE International Conference on Cloud Computing*, San Francisco, CA, USA, June, pp. 407–414. IEEE Computer Society.

[41] Cunha, M., das Chagas Mendonça, N., and Sampaio, A. (2017) *Cloud Crawler*: a declarative performance evaluation environment for infrastructure-as-a-service clouds. *Concurrency and Computation: Practice and Experience*, **29**.

[42] Alshara, Z., Alvares, F., Brunelière, H., Lejeune, J., Prud'homme, C., and Ledoux, T. (2018) Come4acloud: An end-to-end framework for autonomic cloud systems. *Future Generation Comp. Syst.*, **86**, 339–354.

[43] Bonchi, F., Brogi, A., Canciani, A., and Soldani, J. (2018) Simulation-based matching of cloud applications. *Sci. Comput. Program.*, **162**, 110–131.

[44] Dehghanipour, M. (2015) Design and implementation of tosca service templates for provisioning and executing bone simulation in cloud environments. Master's thesis. Universität Stuttgart.

[45] Ardagna, D. et al. (2012) Modaclouds: a model-driven approach for the design and execution of applications on multiple clouds. *Proceedings of the 4th International Workshop on Modeling in Software Engineering*, Zurich, Switzerland, June, pp. 50–56. IEEE Computer Society.

[46] Basu, S., Chakraborty, S., and Sharma, M. (2015) Pricing cloud services—the impact of broadband quality. *Omega*, **50**, 96–114.

[47] Kar, A. K. and Rakshit, A. (2015) Flexible pricing models for cloud computing based on group decision making under consensus. *Global Journal of Flexible Systems Management*, **16**, 191–204.

[48] Arshad, S., Ullah, S., Khan, S. A., Awan, M. D., and Khayal, M. S. H. (2015) A survey of cloud computing variable pricing models. *Proceedings of the 10th International Conference on Evaluation of Novel Approaches to Software Engineering*, Barcelona, Spain, April, pp. 27–32. SciTePress.

[49] Lauwers, C., Lipton, P., and Tamburri, D. (2013) Topology and Orchestration Specification for Cloud Applications. Technical report. oasis-open.org, OASIS.

[50] Durand, J., Andreou, M., Davis, D., and Pilz, G. (2013) Cloud infrastructure management interface (cimi) model and restful http-based protocol. Technical report. DMTF.

[51] Hwang, R.-H., Lee, C.-N., Chen, Y.-R., and Zhang-Jian, D.-J. (2013) Cost optimization of elasticity cloud resource subscription policy. *IEEE Transactions on Services Computing*, **7**, 561–574.

[52] Edmonds, A., Metsch, T., Papaspyrou, A., and Richardson, A. (2012) Toward an open cloud standard. *IEEE Internet Computing*, **16**, 15–25.

[53] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999). Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard). Updated by RFCs 2817, 5785.

[54] Hong, H., Chen, D., Huang, C., Chen, K., and Hsu, C. (2015) Placing virtual machines to optimize cloud gaming experience. *IEEE Trans. Cloud Computing*, **3**, 42–53.

[55] Katsaros, G. (2016) Open cloud computing interface–service level agreements. Technical report. In The Open Grid Forum Document Series.

[56] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999) Hypertext transfer protocol–http/1.1. Technical report. In The Open Grid Forum Document Series.

[57] Edmonds, A. and Metsch, T. (2016) Open cloud computing interface–text rendering. Technical report. In The Open Grid Forum Document Series.

[58] Ciuffoletti, A. (2014) A simple and generic interface for a cloud monitoring service. *Proceedings of the 4th International Conference on Cloud Computing and Services Science*, Barcelona, Spain, April, pp. 143–150. SciTePress.

[59] Samimi, P. and Patel, A. (2011) Review of pricing models for grid & cloud computing. *Proceedings of the IEEE Symposium on Computers & Informatics*, Kuala Lumpur, Malaysia, March, pp. 634–639. IEEE.

[60] Geelan, J. et al. (2009) Twenty-one experts define cloud computing. *Cloud Computing Journal*, **4**, 1–5.

APPENDIX A. OPEN CLOUD COMPUTING INTERFACE

In this appendix, we will briefly introduce the Open Cloud Computing Interface (OCCI) standard and its specification.

The Open Cloud Computing Interface (OCCI) is a RESTful Protocol and API for all kinds of management tasks. OCCI was originally initiated to create a remote management API for IaaS model-based services, allowing for the development of interoperable tools for common tasks including deployment, autonomic scaling and monitoring. It has since evolved into a flexible API with a strong focus on interoperability while still offering a high degree of extensibility. The current release of the Open Cloud Computing Interface is suitable to serve many other models in addition to IaaS, including PaaS and SaaS. In order to be modular and extensible the current OCCI specification is released as a suite of complementary documents, which together form the complete specification. The documents are divided into four categories following categories as illustrated in Figure A.1:

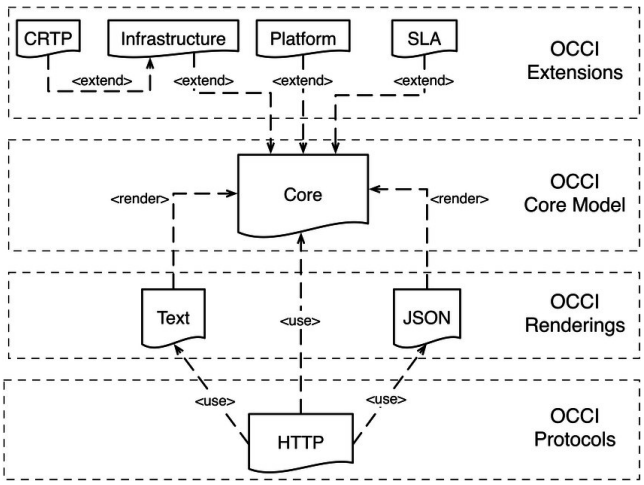


FIGURE A.1: OCCI Specification

Appendix A.1. OCCI Core Specification

The OCCI Core specification [3] consists of a document defining the OCCI Core Model. The OCCI Core Model can be interacted with renderings (including associated behaviours) and expanded through extensions. Figure A.2 illustrates the OCCI Core Model. Table A.1 defines the OCCI core concepts. For more details, refer to [3].

Appendix A.2. OCCI Protocol Specification

The OCCI Protocol specifications consist of multiple documents, each describing how the model can be interacted with over a particular protocol (e.g. HTTP, AMQP, etc.). Multiple protocols can interact with the same instance of the OCCI Core Model. Currently, only OCCI HTTP Protocol, a RESTful protocol for communication between OCCI server and OCCI client has been implemented.

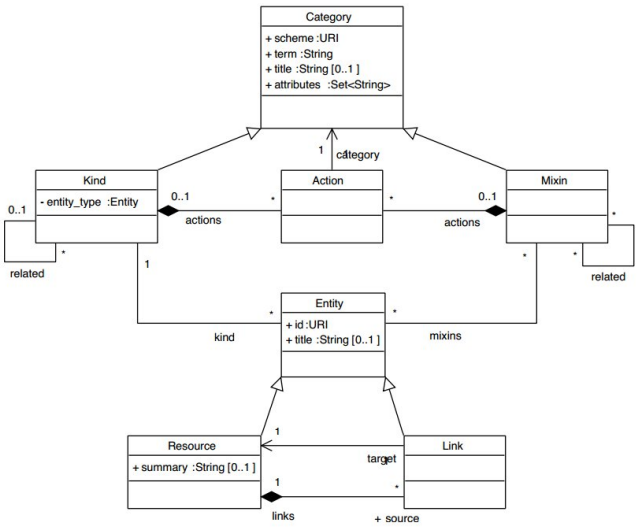


FIGURE A.2: OCCI Core Model

Concepts	Definition
Entity	is the abstract base class of all resources and links. types.
Resource	represents any cloud computing resource, e.g., a virtual machine, a network, an application. Resource owns a set of links.
Link	is a relation between two Resource instances, e.g., a computer connected to a network, an application hosted by a container.
Category	is the abstract base class inherited by Kind, Mixin, and Action.
Kind	is the notion of class/type within OCCI, e.g., Compute, Network, Container, Application.
Mixin	is used to associate additional features, e.g., location, price, user preference, ranking, to resource/link instances.
Action	represents an action that can be executed on entities, e.g., start a virtual machine, stop an application container, restart an application, resize a storage.
Attribute	represents the definition of a client visible property, e.g., the hostname of a machine, the IP address of a network, or a parameter of an action.

TABLE A.1: OCCI core concepts and their definitions

The OCCI HTTP Protocol [13] maps the OCCI Core model into the URL hierarchy by binding *Kind* and *Mixin* instances to unique URL paths. Such a URL path is called the *location* of the *Kind* or *Mixin*. A provider is free to choose the location as long as it is unique within the service provider's URL namespace. For example, the *K*ind instance for the *C*ompute type may be bound to /my/occi/api/compute/. A *K*ind instance whose associated type cannot be instantiated MUST NOT be bound to an URL path. This applies to the Kind instance for OCCI Entity which, according to OCCI Core, cannot be instantiated.

Appendix A.3. OCCI Rendering Specification

The OCCI Rendering specifications consist of multiple documents, each describing a particular rendering of the OCCI Core Model. Multiple renderings can interact with the same instance of the OCCI Core Model and will automatically support any addition to the model which follows the extension rules defined in OCCI Core. Currently, both OCCI Text [57] and JSON [14] renderings have been defined for the OCCI standard.

Appendix A.3.1. Text Rendering

The OCCI Text rendering [57] specifies a rendering of OCCI instance types in a simple text format. The rendering can be used to render OCCI instances independently of the protocol being used. Thus messages can be delivered by, e.g., the HTTP protocol. The following example show an *E*ntity instance rendering using the text plain rendering.

```
< Category: compute; \
< scheme="http://schemas.ogf.org/occi/infrastructure#" \
< class="kind";
< Link: </users/foo/compute/b9ff813e-fee5-4a9d-b839-673
    f39746096?action=start>; \
< rel="http://schemas.ogf.org/occi/infrastructure/compute
    /action#start"
< X-OCCI-Attribute: occi.core.id="urn:uuid:b9ff813e-fee5
    -4a9d-b839-673f39746096"
< X-OCCI-Attribute: occi.core.title="My Dummy VM"
< X-OCCI-Attribute: occi.compute.architecture="x86"
< X-OCCI-Attribute: occi.compute.state="inactive"
< X-OCCI-Attribute: occi.compute.speed=1.33
< X-OCCI-Attribute: occi.compute.memory=2.0
< X-OCCI-Attribute: occi.compute.cores=2
< X-OCCI-Attribute: occi.compute.hostname="dummy"
```

Listing 1: OCCI Text rendering

Appendix A.3.2. JSON Rendering

The OCCI JSON Rendering [14] specifies a rendering of OCCI instance types in the JSON data interchange format. The Rendering can be used to render OCCI instances independently of the transport mechanism being used. Thus messages can be delivered by e.g. the HTTP protocol or by using text files with the .json file extension.

```
{
"resources": [
{
"kind": "...",
"mixins": [ "...", "..." ],
"attributes": { },
"actions": [ { } ],
"links": [ { }, { } ]
}
]
}
```

Listing 2: OCCI JSON rendering

Appendix A.4. OCCI Extension Specification

The OCCI Extension specifications consist of multiple documents, each describing a particular extension of the OCCI Core Model. The extension documents describe additions to the OCCI Core Model defined within the OCCI specification suite. In papaer, we are interested in infrastructure extension [12]. However, many others extensions have been proposed such as Platform extension [15] and SLA extension [55].

Appendix A.4.1. OCCI Infrastructure extension

The OCCI Infrastructure extension [12] details how an OCCI implementation can model and implement an Infrastructure as a Service API offering by utilizing the OCCI Core Model. This API allows for the creation and management of typical resources associated with an IaaS service, for example, creating a Compute instance and Storage instance and then linking them with StorageLink. The main infrastructure types defined within OCCI Infrastructure are illustrated in Figure A.3:

- Compute: Information processing resources
- Network: Interconnection resource that represents an L2 networking resource. This is complemented by the *IPNetwork Mixin*.
- Storage: Information recording resources.
- NetworkInterface: connects a *Compute* instance to a Network instance. This is complemented by an *IPNet* workInterface *Mixin*
- StorageLink: connects a *Compute* instance to a *Storage* instance

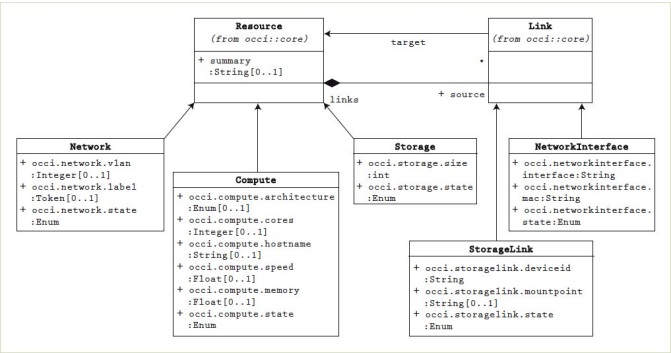


FIGURE A.3: OCCI Infrastructure Model

These infrastructure types inherit the OCCI Core Model Resource base type and all its attributes. The HTTP Protocol and Text Rendering documents define how to serialize and interact with these types using RESTful communication.

To summarize, Table A.2 captures the different OCCIware specifications and lists the OCCI concepts defined in the corresponding document.

APPENDIX B. SIMULATION DESIGNER AND CONFIGURATION

In this section, we will give some technical details about the simulation designer and configuration.

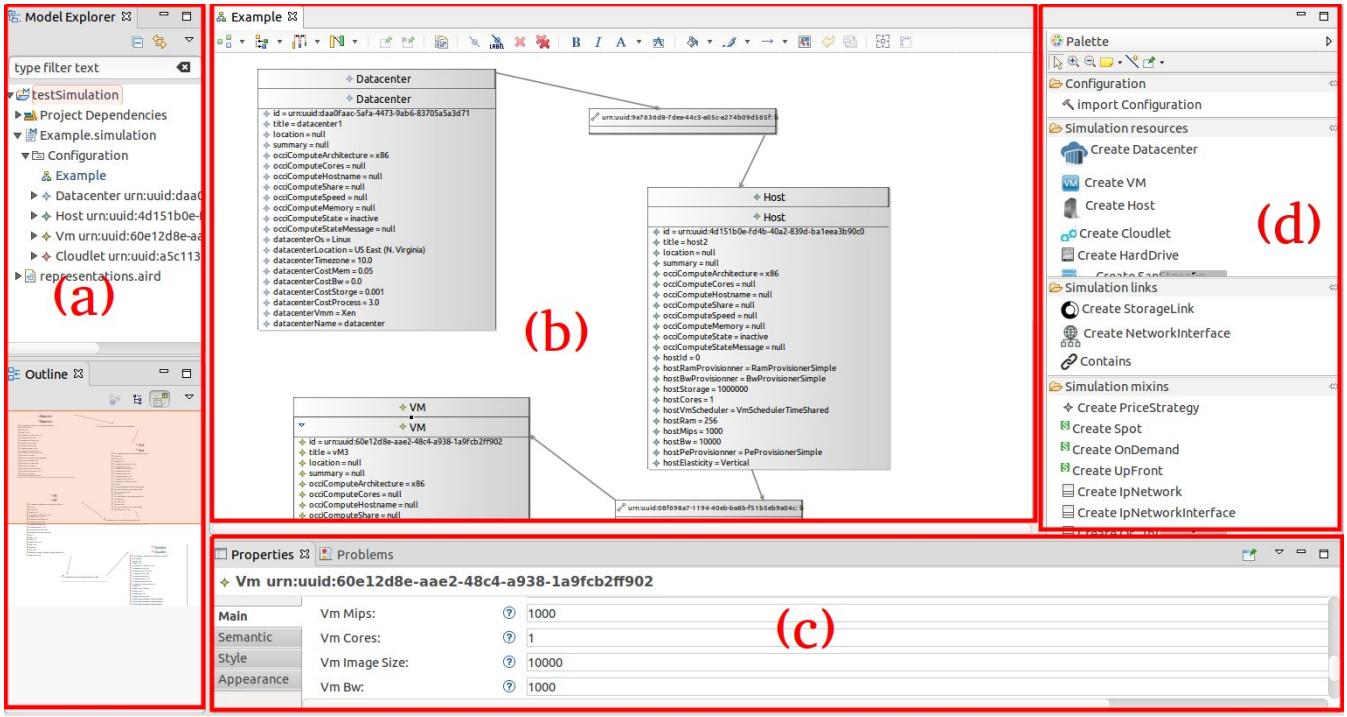


FIGURE A.4: Screenshot of OCCIware Simulation Designer

Specifications	Defined concepts
OCCI - Core [3]	Entity, Resource, Link, Kind, Mixin, Action, Attribute, Category
OCCI - Infrastructure [12]	Compute, Storage, Network, StorageLink, NetworkInterface
OCCIware metamodel v1 [4]	Extension, Configuration, AttributeState
OCCIware metamodel v2 [5]	MixinBase, Type, Constraint, DataType (and its different subtypes)

TABLE A.2: OCCIware concepts and their specifications

Appendix B.1. Simulation Designer Interface

The OCCIware simulation designer, generated from the simulation extension, allows the creation of simulation configurations and runs the simulation. It was developed on top of Eclipse Sirius. Figure A.4 presents the OCCIware Simulation Designer.

- Frame (a) in Figure A.4 displays the Eclipse Model Explorer used to navigate through the simulation project containing the simulation configuration;
- Frame (b) in Figure A.4 shows the OCCIware simulation designer that provides a graphical representation of simulation configuration;
- Frame (c) of Figure A.4 contains the Eclipse properties editor for viewing and modifying attributes of a selected modeling element;
- Frame (d) of Figure A.4 shows the palette of the OCCIware simulation designer, with palette elements to import an existing configuration, create OCCIware resources and EPriceCloudSim resources, establish relations between resources, and assign new attributes to resources.

From the proposed designer, the cloud architect models a cloud configuration by using drag-and-drop operation. The configuration in Frame (b), in Figure A.4, contains four compute resources: a datacenter, a Host linked to the datacenter, a VM linked to the Host and a Cloudlet linked to the VM instance. Each resource is defined by a list of attributes such as the instance capacity (in term of storage, cpu, bandwidth), the desired region, elasticity type, etc.

Once the simulation configuration is designed and verified, the architect generates the Java source code of this configuration and launches the simulation. These are made automatically from the contextual menu.

Appendix B.2. Simulation

EPriceCloudSim receives the simulation configuration as input in *Map<Resource, List<Resource>>* format. This map contains EPriceCloudSim resources (datacenter, Host, VM and Cloudlets) and links between them. Each resource is defined by a list of attributes extracted from the simulation configuration.

Firstly, EPriceCloudSim constructs the infrastructure of the application from the received simulation configuration. This step is done by using the source code generated automatically by the simulation designer. EPriceCloudSim extracts the different resources with their attributes and

runs the simulation. During the simulation process EPriceCloudSim can detect an insufficient capacity of the resources. In this case, a new provisioning algorithm is provided to support the elasticity.

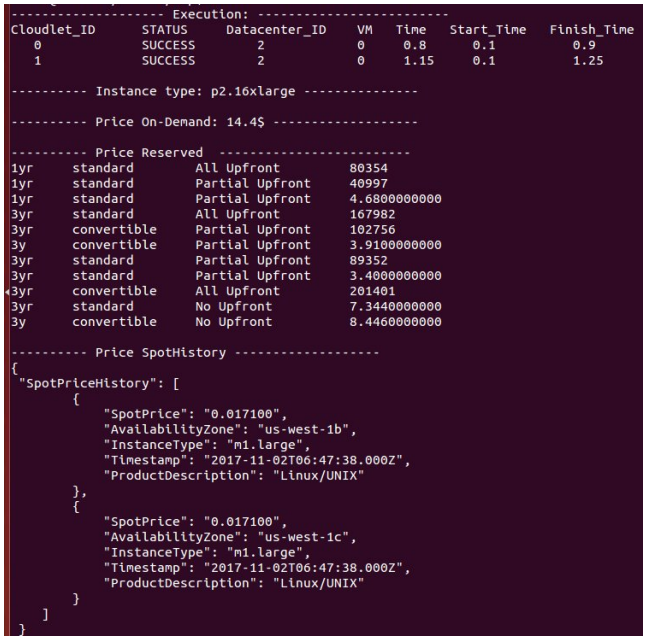


FIGURE B.1: The simulation results

Then, EPriceCloudSim simulates the execution of the application over the infrastructure constructed. As output, EPriceCloudSim estimates the execution time and the resources needed on the infrastructure.

Afterward, EPriceCloudSim calls the REST application to retrieve the real cost of the infrastructure generated on AWS. To this end, EPriceCloudSim calls the GET method with the characteristics in the attributes such as memory size, time estimated by EPriceCloudSim in the first step, CPU size and the availability zone.

The simulation results are stored in a log file. This later contains the informations about the execution time of the application and the real deployment cost in different strategies. An example of the log file is presented in Figure B.1.