# Model-driven Performance Estimation, Deployment, and Resource Management for Cloud-hosted Services *

Faruk Caglar Kyoungho An Shashank Shekhar Aniruddha Gokhale
Vanderbilt University, ISIS and EECS
{faruk.caglar,kyoungho.an,shashank.shekhar,a.gokhale}@vanderbilt.edu

## Abstract

There is a growing trend towards migrating applications and services to the cloud. This trend has led to the emergence of different cloud service providers (CSPs), in turn leading to different cost models offered by these CSPs to lease their resources, variabilities in the granularity and specification of resources provided, and heterogeneous APIs offered by the CSPs to the users to program resource requests and deployment for their cloud-hosted services. These challenges make it hard for customers of the cloud to seamlessly transition their services to the cloud or migrate between different CSPs. To address these challenges, this paper presents a solution based on model-driven engineering (MDE). Specifically, we describe the design of the domain-specific modeling languages in our MDE framework and the associated generative mechanisms that address the challenges related to estimating performance and cost to host the services in the cloud, automated deployment and resource management.

***Categories and Subject Descriptors*** Computing Methodologies [*Simulation and Modeling*]: Applications

***General Terms*** Design, Performance

***Keywords*** model-driven analysis, deployment, cloud computing

## 1. Introduction

Cloud computing [1] offers scalability, extensibility, elasticity, flexibility, and cost savings to the customers of cloud service providers, which is the reason it is increasingly becoming an attractive technology to host different types of applications and services. Even mission-critical and real-time applications are moving to the cloud. Although these trends demonstrate the promise that cloud computing holds for the future, multiple unresolved challenges must be overcome before it becomes easy for users to access the services of the cloud. These challenges can roughly be classified into three categories: *Programming and Deployment Heterogeneity, Resource Management,* and *Performance and Cost Estimation*.

• **Programming and Deployment Heterogeneity:** Cloud Service Providers (CSPs), such as Amazon EC2, GoGrid, RackSpace, and Microsoft Azure provide different APIs to their customers to manage their resources on the cloud. This API heterogeneity imposes a steep learning curve for cloud customers while also limiting their ability to seamlessly migrate their services between the CSPs. Some recent efforts to deal with API heterogeneity include DeltaCloud [5], libcloud [6], and jclouds [7]. Some of these libraries are programming language-dependent such as jclouds and libcloud which could be utilized by a Java-based and Python-based applications respectively. Another tool, JetS3t [8], is applicable to Amazon S3, CloudFront, and Google Storage. While, these efforts are promising, we believe these technologies help address only the API heterogeneity issue.

A closely related issue is that of deployment of services to the cloud, which is often carried out programmatically using the APIs. Consequently, the above limitations exist in this case also. To overcome some of the deployment concerns, CSPs often provide a web-based management console. Unfortunately, these user interfaces are very specific to the CSP and hence do not resolve the original problem.

• **Resource Management:** Depending on the service hosting model, the responsibility of resource management (*i.e.*, determining the properties of the virtual machine and autoscaling as the demand changes) remains the responsibility of the cloud customer. Effective decisions on autoscaling of resources is a runtime property and is dictated by the operating environment of the CSP, the workload, and degree of resource sharing – which is a prominent feature of public clouds. These decisions must be programmed using the APIs, which is already shown to be challenging.

• **Performance and Cost Estimation:** Cloud computing comes with a cost; the accounting is based on a utility model. Making decisions on how many cloud resources to use to host a service, and when and how much to autoscale is a significant challenge for the cloud customers. Understanding what will the impact of these decisions be on both the expected performance delivered to the service and cost incurred by the customer is even harder.

Addressing these challenges requires a framework that holistically focuses on the core set of problems by providing intuitive abstractions to the cloud customer to enable various CSP-agnostic "what-if" analyses while automating the deployment and resource management. To that end we have developed a solution based on model-driven engineering (MDE). The key artifacts of our MDE

* This work was supported in part by NSF CNS CAREER award 0845789. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.



*DSM '13*, October 27, 2013, Indianapolis, Indiana, USA.

http://dx.doi.org/10.1145/2541928.2541933

solution includes domain-specific modeling languages (DSMLs) and generative technologies.

In [2] we outlined the vision behind this work. In this paper, we focus on describing the framework including the DSMLs and their metamodels, the model interpreters, and middleware capabilities developed for simulation and automated deployment. Specifically, this paper makes the following contributions to address the challenges outlined above:

• **Performance, Cost and Resource estimation –** we describe a DSML that allows a cloud customer to describe their service and resource needs. Generative capabilities associated with the DSML generate scripts to drive a simulator for a CSP. Feedback from executing these simulations provide customers an idea about performance and cost estimates. Using a DSML shields the customer from having to learn a simulator and its interface.

• **Overcoming heterogeneity –** the same models developed in the first step are then used to synthesize deployment scripts for the underlying CSP thereby shielding the user from having to manually write scripts using low-level APIs, and promoting easy migration between the CSPs.

The rest of the paper is organized as follows: Section 2 describes the MDE process to analysis and deployment; Section 3 describes related work; and finally Section 4 provides concluding remarks and outlines future work.

## 2. A Holistic Model-driven Framework for Cloud Hosting

Our MDE solution is described in three parts. First, we outline the use of modeling in our solution. Second, we show how the modeling capabilities are used in the context of a simulator to conduct what-if analysis used in performance and cost estimation for the different kinds of resources used to host the service. Third, we show how the modeling tools can help automate the deployment of the services to cloud platforms shielding the user from the heterogeneity in cloud providers.

### 2.1 Overview of the Modeling Process

Our MDE solution comprises two DSMLs and associated tools: a DSML for simulator-based analysis capability used to estimate the performance and price for hosting a service in the cloud, and a DSML for automating its deployment across a range of CSPs.

We have used the Generic Modeling Environment (GME) [9], which is a language workbench.

### 2.2 Model-based Cloud Simulation

Estimating the performance of deployed services in the cloud is not straightforward because of different and often varying number of resources, such as hardware and network that exist in cloud data centers, which are required by the hosted services. Additionally, the impact on performance by resource scheduling and allocation polices of the cloud platform may differ. Finally, varying and dynamic workloads and QoS requirements of services make it harder to evaluate performance of these services on the cloud platforms.

To overcome these challenges, CloudSim [4] provides a simulation environment of the cloud infrastructures and services. Developers can test the performance of their services deployed in heterogeneous cloud infrastructures, such as Amazon EC2 and Microsoft Azure via a simulation environment provided by CloudSim as well as determine the cost of cloud hosting. CloudSim provides diverse modeling and simulation features for cloud infrastructures: large scale cloud data centers, virtualized server hosts with customizable policies for provisioning host resources to virtual machines, energy-aware computational resources, and data center network topologies.

Even though CloudSim offers a cloud computing simulation environment, it is not easy for users to use it without incurring a learning curve that includes understanding the CloudSim APIs available in the Java programming language. Given the plethora of choices and heterogeneity with cloud platforms, it is a significantly comprehensive and involved task of having to program the CloudSim simulator. Moreover, it is difficult to integrate the simulation tool with other tools, such as deployment tools and data center network simulation tools.

We have used GME to address these interfacing and integration challenges and develop a DSML and generative capabilities for the domain of cloud simulation for performance and cost estimation for resource allocations specified by the user. Figure 1 illustrates the metamodel, which is at the heart of the DSML for cloud simulation. The purpose of the different colors in the metamodels is to distinguish entities from each other for the reader. The metamodel primarily comprises first class entities, such as DataCenter, DataCenterBroker, virtual machine (VM), and Cloudlet. Although most of these elements are generic cloud artifacts, the Cloudlet is specific to the CloudSim simulator.

The metamodel components in the DSML and their responsibilities are as follows:

- *DataCenter:* defines CSPs such as Amazon EC2, Windows Azure, or private data centers. It is simply a resource provider where host machines are virtualized. This component aims to provide information for all the data center components such as host, VM, and storage. It contains default attributes, but the attributes such as specification of physical servers, VM allocation policy, and pricing information can also be configured through components (e.g. Cost and Host) by users.
- *DataCenterCharacteristics:* contains *DataCenter* component and defines characteristics of the data center and the storage components. It stores the properties of a datacenter such as architecture, OS, and cost of using a specified resource.
- *VM:* is used to define requested VMs from clients. Attributes of a VM include ID, millions of instructions per second (MIPS), memory, image size, number of CPUs, bandwidth, size, and virtual machine manager type.
- *Host:* represents a physical host in a data center. Memory, storage, bandwidth, and number of CPUs are some of the attributes that could be defined.
- *Cloudlet:* defines the application services deployed in clouds. CPU, memory, and bandwidth utilization models are some of the attributes it stores. Based on this template, the number of the *Cloudlets* can be configured by users.
- *DatacenterBroker:* acts as a bridge between cloud data centers and cloud users. Therefore, the *VMs* and the *Cloudlets* defined by cloud users are connected to the *Datacenter* via the *DatacenterBroker* component.
- *Cost:* aims to simulate the cost information when the service model is deployed and executed in the datacenter defined in the simulation. It stores cost per memory, cost per second, cost per bandwidth, and cost per storage attributes, and provides them to the *DataCenterCharacteristics* component
- *Storage:* our proof of concept application does not require cloud storage, so we integrated the simplified version of it only for the benefit of future enhancements.

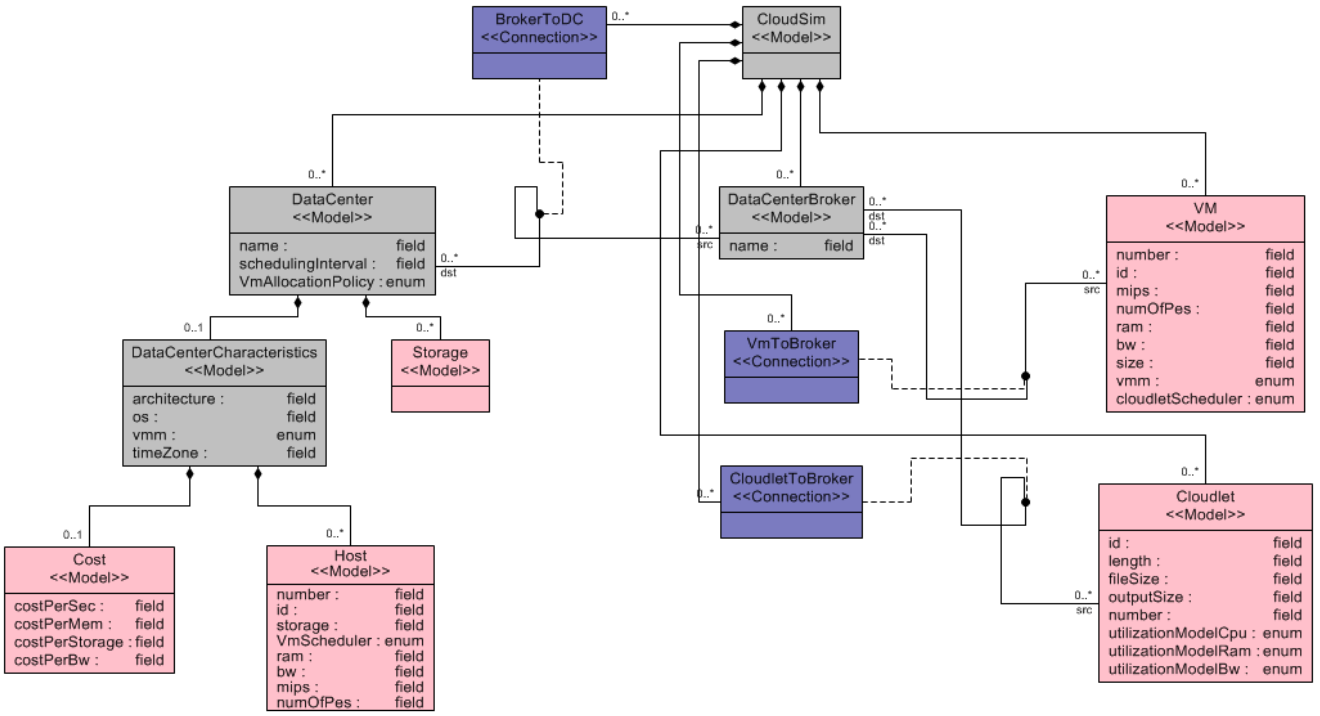


**Figure 1.** Metamodel for Estimating Cloud-based Service Performance and Cost, and Cloud Resource Usage

A simple example model of cloud simulation using the DSML is shown in Figure 2. In the figure, DSML components to simulate a cloud environment, such as a *VM* template, a *Cloudlets* template, a *DatacenterBroker*, and a *Datacenter* are defined. The *VM* template contains default attributes, such as CPU, RAM, network bandwidth, and the hypervisor (*e.g.* Xen, KVM, and HyperV). The default attributes can be modified according to users' environment. Users can also change the number of VMs defined by the *VM* template. The *Cloudlet* template, which defines cloud-based application services such as content delivery, social networking, and business workflow [4], also has configurable attributes such as pre-assigned instruction length and data transfer overhead.

The overall process of the example model is that the defined *VMs* and *Cloudlets* are submitted to the *DatacenterBroker*, which is mediated by the *DatacenterBroker*, which in turn makes requests for deploying VMs and cloud application services to the *Datacenter* on behalf of a user. After the configuration of the example CloudSim model is completed, the example model is transformed into Java-based CloudSim codes to simulate the cloud environment. Once the generated code is executed, cost and performance are simulated by CloudSim application.

### 2.3 Model-based Automated Deployment in the Cloud

The second modeling capability in our MDE approach enables automated deployment to the cloud, which is needed to overcome the challenges resulting from heterogeneity in CSP APIs and deployment policies. The metamodel of the deployment model in our DSML is depicted in Figure 3, which consists of *Print*, *Sleep*, *Upload*, *Download*, *RunApp*, *Terminate*, *CreateInstance*, *WaitforStartup*, *Connect*, *Entity*, and *Keyfile* model components, which are used during the modeling process.

The metamodel was partitioned into three viewpoints (called Aspects in GME), which show or hide the design details, named as *AllKeywords*, *DisplayKeywords*, and *ActionKeywords*. The connections between components are also defined in the metamodel, such as *CreateInstance* component could connect to *WaitforStartup* component, and the components of *ActionKeywords* and *DisplayKeywords* aspects. The `aws_access_key_id`, host, and port are some of the attributes associated with the *Connect* component. The rest of the attributes associated with each component are also defined in the metamodel.

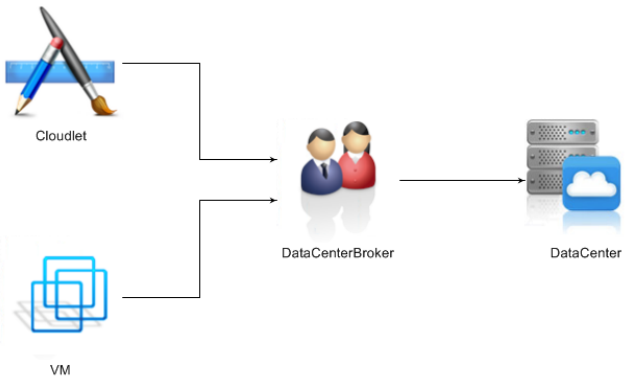


**Figure 2.** Model Example

The metamodel components in the DSML and their responsibilities are as follows:

- *Print:* prints a message specified on the screen. This command aims to provide information to the cloud user during the deployment.
- *Sleep:* stops the program flow and waits for an interval specified. Generally, even though VMs are switched to the running status from pending status after they are created, physically they

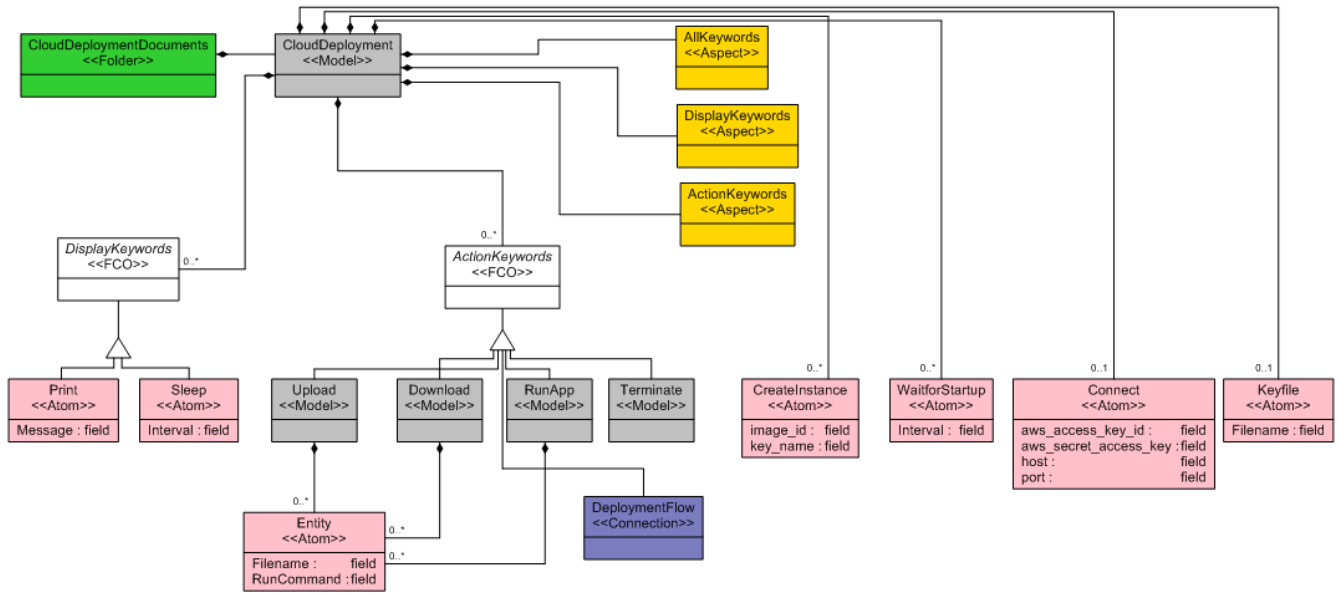


**Figure 3.** The Metamodel for Automated Deployment in the Cloud

might not be available instantaneously and at the running status yet. Copying a file onto the created VM will not succeed. Therefore, this command is needed to wait for VMs to launch. Additionally, it allows applications to run for a while and retrieve their outputs.

- *Entity:* keeps the information of an entity which will run a command or the file name to be uploaded or downloaded. To upload or download a file, the file name needs to be known. To execute a command on a VM, what command will be executed needs to be provided. Therefore, this information is supplied by this command.
- *Upload:* contains entities such as text file, executable file, or any other library files and uploads them onto the VMs that it is connected to. This command is used to copy the application's set up and log files from a local directory to another directory on a VM in the cloud.
- *Download:* is used to download entities from the VMs it is connected to. This command copies the log file of an application from a directory on a VM in the cloud to a local directory.
- *RunApp:* is used to execute the commands specified by different entities against the VMs it is connected to. After an application is deployed in the cloud, it is required to launch it. This command simply runs the application deployed in the cloud remotely.
- *Terminate:* stops all the VMs created by each CreateInstance component. After the mission of the application deployed in the cloud is accomplished, it might no longer be needed to have all the VMs running, and hence this command is used to terminate all the running VMs associated with an application.
- *CreateInstance:* creates a VM and runs it. Image id and key name properties of the VM are specified by its attributes. To deploy an application in the cloud, a VM(s) is needed to be created first, which is responsibility of this command.
- *WaitforStartup:* waits for all the VMs that it is connected to be launched. It checks for their status and lets the flow continue after all the instances are at the "running" status. A VM has to be in the running status to start the deployment process. Therefore, this command is a kind of switch to initiate migration process.
- *Connect:* creates a connection to the host and port number specified. This command represents a connection object to the cloud.
- *Keyfile:* keeps the file name of the key file to make ssh connection. To make ssh connection to a VM, the local machine should authenticate first. This command provides the name of the key file previously created for authentication purposes.

In Figure 4, an example model of automated deployment using the DSML in Figure 3 is depicted. The connections back to CreateInstance component from Download, Upload, and RunApp components are to indicate the target such as Download component downloads the file specified in its attribute from the VM that it is connected to. The deployment script generation algorithm finds the *Connect* and *Keyfile* components in the model first and then processes the model recursively starting from the *Connect* component till it reaches the *Terminate* component when the interpreter is run. We used the Builder Object Network (BON2) in GME to code the model interpretation, which generates the deployment script in Python. For this model, the generated code deploys a network application, which comprises (1) two server applications, (2) a client application, and (3) a fault manager application, in the cloud. The overall picture and further details of each application being deployed in the cloud is outside the scope of this paper, and hence not described further. Each application is deployed onto a separate VM, respectively.

The model interpreter for our example in Figure 4 generates a deployment script which will execute the following steps:

1. A connection to the public or private cloud server which is a public interface to the clients is created
2. Four VM instances are created and run
3. Waits for all the VMs to be advanced to the running state
4. Prints "Waiting 30 sec" before copying files over to the newly created instance(s)

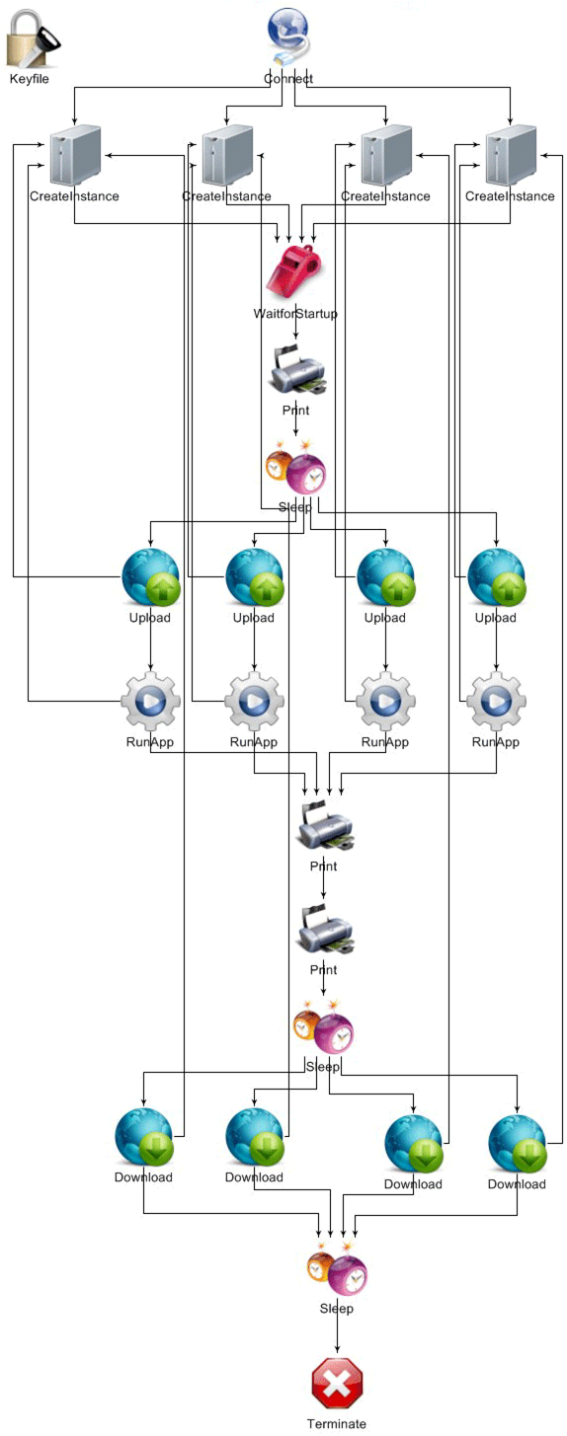


**Figure 4.** An Example Model of Model-based Automated Deployment in the cloud

5. Waits for 30 sec after all the VMs are at running state before the *Upload* component is interpreted
6. Uploads entities specified under each *Upload* component onto the VMs that they are associated with. (The entities under each *Upload* components are Server.exe, Server.exe, Client.exe, and FaultManager.exe, respectively)
7. *RunApp* component executes the command in the entities under it against the VM that it is associated with. Basically, these are the commands that need to be executed as part of launching the application deployed.
8. Prints Waiting 1 min to get output files
9. Prints "Please terminate the main server" 1 min later: kill -9 process ID
10. Waits for 180 sec after all the applications are executed before the *Download* component is interpreted.
11. Downloads entities specified under each *Download* component onto the VMs that they are associated with. (The entities under each *Download* component are Server.txt, Server.txt, Mainclient.txt, and FaultManager.txt respectively)
12. Waits for 10 sec after all the files are downloaded from the VMs
13. Finally terminates all the VMs created

## 3. Related Work

The work presented in [11] provides a model-based proxy for unified Infrastructure as a Service (IaaS) management. The purpose is to manage services provided by any cloud platform from a common interface. Amazon Elastic Compute Cloud (EC2) service is the only cloud platform supported by that work. The work in that paper differs from this research in that it does not provide price simulation and automated deployment. However, similar concepts of having model-based structure and providing unified proxy are present in both works. Our approach intends to be cloud platform-agnostic and through its generative mechanism be able to operate with the platform.

Unlike Deltacloud [5] and Libcloud [6], however, our work in this paper comprises the price simulation, automated deployment, and limited support on VM management tasks. In contrast, they already provide a single API with multiple cloud platforms with no model-based interaction.

EMUSIM is another simulation environment which supports the modeling, evaluation, and validation of performance of Cloud computing applications. It is built on top of Cloudsim that we have used in our research [3].

Uni4Cloud [10] is an approach promising (1) automated deployment independent of cloud service-providers and (2) deployment of an application components over multiple clouds. They propose modeling, deployment, and management of applications in multicloud platforms and facilitate the Open Virtualization Format (OVF) format to deploy an application to different clouds. Our work has synergies with this related work in the context of being applicable to multiple cloud platforms. In contrast, however, our work provides price and performance simulation in advance of deployment.

## 4. Conclusions and Future Work

This paper presented the results of investigations on the DSMLs and generative capabilities that yield the model-based simulation and automated deployment in the cloud. Cost and performance results for a given model are simulated, and automated deployment scripts are generated by the MDE tooling. This helps shield the users from possible complex price calculations, uncertainties, and the low-level API details.

The current MDE capabilities can be extended further to handle more complex analysis problems for the users when they must handle more complex, multi-objective optimization functions to transition to the cloud. Similar objectives exist for codifying deployment and resource management approaches within modeling frameworks. This research will finally yield to a complete

MDE tooling and model-based middleware supporting all the cloud service-provider APIs and many cloud simulation tools. Our aim is to move in the direction of making these two DSMLs as mature and industry-strength languages.

## References


[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. A View of Cloud Computing. *Communications of the ACM*, 53(4):50–58, 2010.

[2] F. Caglar, K. An, A. Gokhale, and T. Levendovszky. Transitioning to the cloud?: a model-driven analysis and automated deployment capability for cloud services. In *Proceedings of the 1st International Workshop on Model-Driven Engineering for High Performance and CLoud computing*, page 8. ACM, 2012.

[3] Calheiros, R. N., Netto, M. A.S., C. A. De Rose, and R. Buyya. EMUSIM: An Integrated Emulation and Simulation Environment for Modeling, Evaluation, and Validation of performance of Cloud Computing Applications. *Software: Practice and Experience*, pages 00–00, 2012.

[4] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya. CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. *Software: Practice and Experience*, 41(1):23–50, jan 2011.

[5] deltacloud.org. Deltacloud. `deltacloud.org`, 2012.

[6] incubator.apache.org. Libcloud. `incubator.apache.org/libcloud/`, 2012.

[7] jclouds.incubator.apache.org. jclouds. `jclouds.incubator.apache.org/`, 2013.

[8] jets3t.s3.amazonaws.com. JetS3t. `jets3t.s3.amazonaws.com`, 2013.

[9] Á. Lédeczi, Á. Bakay, M. Maróti, P. Völgyesi, G. Nordstrom, J. Sprinkle, and G. Karsai. Composing Domain-Specific Design Environments. *Computer*, 34(11):44–51, 2001.

[10] A. Sampaio and N. Mendona. Uni4Cloud: An Approach based on Open Standards for Deployment and Management of Multi-cloud Applications. 2011.

[11] S. Yan, B. S. Lee, and S. Singhal. A Model-Based Proxy for Unified IaaS Management. 2010.