Model Driven Framework for the Configuration and the Deployment of Applications in the Cloud

Hiba Alili, Rim Drira and Henda Hajjami Ben Ghezala
RIADI Laboratory, National School of Computer Sciences,
University of Manouba, 2010 la Manouba, Tunisia
E-mail: {alilihiba,drirarim,hhbg.hhbg}@gmail.com

***Abstract*—Cloud computing offers a distributed computing environment where applications can be deployed and managed. Many companies are seeing substantial interest to extend their technical infrastructure by adopting cloud infrastructures. Although the choice of such an environment may seem advantageous, users are faced with many challenges, especially with regard to deployment and migration of applications in the Cloud. To address some of these challenges, we propose a new approach based on model-driven engineering techniques (MDE), called MoDAC-Deploy, for the assistance to the configuration and the deployment of applications in the Cloud. This paper focuses on the design and the implementation of our approach. In fact, we developed a model-driven Framework with generative mechanisms to simplify and to automate cloud services deployment process, to overcome APIs heterogeneity, to minimize the vendor lock-in and to enable application portability among different cloud infrastructures by reusing configurations/deployments "Model a configuration once and deploy it anywhere". We conducted also a case study in order to validate our proposed approach. Our empirical results demonstrate the effectiveness of our MDE Framework to seamlessly deploy services in the cloud and to migrate easily between different Cloud Service Providers (CSPs) without any programming efforts.**

Keywords-Deployment; Cloud Computing; Model Driven Engineering.

I. INTRODUCTION

Cloud Computing is a paradigm shift that involves dynamic provisioning of shared computing resources on demand. It is a pay-as-you-use billing model that offers computing resources as a service in an attempt to reduce IT capital and operating expenditures [1].Particularly, Infrastructure as a Service (IaaS) allows users to allocate computational, storage and networking resources from Cloud Service Providers (CSPs). It offers to users the ability to customize the environment to suit their applications and even it supports the deployment of legacy applications without any modification in their source code. In order to make efficient use of such an environment, tools are needed to automatically deploy, configure and run services in a repeatable way. In this context, we focus in this paper on the deployment of applications in IaaS environment.

Deploying applications in cloud infrastructures is not a trivial task, as it relies on handcrafted scripts and it requires increased complexity and additional effort. Doing so is time consuming and error prone, especially for deployments with a large number of nodes. Moreover, the growing trend towards migrating applications and services to the cloud has led to the emergence of different CSPs, in turn leading to different specifications of provided resources and to heterogeneous APIs. These challenges make it hard for cloud customers to seamlessly transition their services to the cloud or migrate between different CSPs. These challenges can be classified into three main categories:

- Deployment Complexity: the deployment in the cloud is a very complex process given the large number of operations required to finish with a successful deployment (e.g., the restructuring of each application layer for the cloud, the auto-scaling of services, the monitoring and the optimization of the application services to take advantage of the cloud benefits) [2]. In fact, to successfully deploy an application in the cloud, a good preparation of the target environment is essential to be compatible with its architecture.
- Programming and Deployment Heterogeneity: CSPs such as Amazon Web Services [3], Google Cloud Platform [4], Rackspace, and Microsoft Azure [5] provide different APIs to their customers to manage their resources on the cloud, which is often carried out programmatically using this APIs. This API heterogeneity imposes a steep learning curve for cloud customers [6]. To overcome this concern, 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.
- Vendor lock-in and Portability: The fear of vendor lock-in is often cited as a major impediment to cloud service adoption. In fact, the proprietary APIs provided by each CSP are incompatible with those of other CSPs and as a result it limits the ability of cloud customers to seamlessly migrate their services between different CSPs. For that reason, many customers stay with a provider that doesnt meet their needs, just to avoid the cumbersome process.

Addressing these challenges requires a framework that holistically focuses on the core set of the deployment problems. In parallel, MDE has emerged as a software engineering paradigm for dealing with the problem of system interoperability and portability across different execution platforms. Model Driven Architecture (MDA) does this separating business and technical concerns and proposing techniques to integrate them. In addition, MDA techniques allow generating automatically code from models. Thus, we believe that MDE techniques are promising to address the challenges outlined above (automating the deployment process and ensuring the portability across different cloud infrastructures).

In this context, we propose in this paper an intuitive abstraction, based on MDE standards, to cloud customers to model software deployment in the cloud and to enable various CSP-agnostic. This abstraction is realized as a modeling tool based on a domain specific modeling language (DSML) with

 ISBN: 978-1-61208-460-2

generative capabilities. Our proposal, which we call MoDAC-Deploy (Model Driven Framework for the Assistance to Cloud Deployment), includes three key artifacts: (1) IaaSEditor, a modeling tool which provides an intuitive user interface that allows cloud customers to define the deployment model of their applications in the cloud. It presents an abstraction layer isolating applications from the underlying cloud provider and hiding APIs. This tool is based on (2) IaaSMetaModel, a meta-model that captures all the concepts needed to specify an accurate cloud deployment. And finally (3) ScriptGenerator, a generative tool that concludes automatically the deployment script from the deployment model created within IaaSEditor. Our framework shields cloud users from having to manually write scripts using low-level APIs and enables applicaion portability among different CSPs.

This paper is organized as follows: Section 2 briefly discusses scientific works closely related to ours. In Section 3 , we introduce the MDE basis, especially we describe the MDA process. Section 4 describes our model driven framework for the configuration and the deployment of applications in the cloud. In Section 5, we illustrate a case study to evaluate and to demonstrate the effectiveness of our deployment framework and finally, Section 6 provides concluding remarks and outlines future works.

II. Related works

Our work has taken shape in the context of a rich literature focused on simplifying the deployment of applications in the cloud. In fact, several works have shown an interest to automate the deployment process and to deal with API heterogeneity. We propose in this section to analyze the state of the art about software deployment and identifying the good practices to be reused in our own solution.

Juve and al. [7] have developed a system called Wrangler to provision, configure and manage virtual machine deployments in the cloud. Wrangler allows users to specify the layout of their application declaratively using an eXtensible Mark-up Langage (XML) format and then to send this deployment description to a web service that manages the provisioning of virtual machines, the installation and the configuration of software and services. This system is able to interface with different resource providers,as it currently supports only Amazon EC2 [8], Eucalyptus [9] and OpenNebula [10]. But authors haven't talk about the possibility to extend this system in order to support other CSPs. While our solution is designed specifically to support multiple CSPs and to easily add new cloud artifacts. Our approach intends also to completely shield software designers from any programming efforts contrary to Wrangler that requires the preparation of an XML description of the deployment model.

Caglar and al. [11] have proposed a solution based on MDE, including a domain-specific modeling language (DSML) for automating deployment of applications in the cloud and generative technologies. The meta-model of the deployment model in this DSML was designed in order to overcome the challenges resulting from heterogeneity in CSP APIs and deployment policies. It consists of Print, Sleep, Upload, Download, RunApp, Terminate, CreaeInstance, WaitforStartup, Connect, Entity, and Keyfile model components, which are used during the deployment process. Connections between components are also defined in the meta-model. The interpretation of the created deployment model generates the appropriate deployment script in Python, which contains and execute the deployment steps. Just like Wrangler [11], this work is limited to VM management, while our work supports also storage and network connectivity management. In addition, it allows users to specify resources into groups (availability group, security group and auto-scaling group).

In [12], the authors describe their automatic deployment platform that they developed for the Microsoft Azure cloud, driven by the need of a chemistry application performing Quantitative Structure-Activity Relationship (QSAR) analysis. The main goal was to enable the execution of existing non-.Net software in the Azure infrastructure which was designed only for applications based on the .Net framework, and which supports a specific, queue-based software architecture. By using the proposed deployment framework, the QSAR application was successfully running in the Azure infrastructure. However, this solution is dedicated only to the Azure cloud and it needs to be generalized.

Shekhar and al. [13] have proposed a framework for conducting price/performance tradeoffs in executing MapReduce jobs at various CSPs, selecting the best option and deploying and executing the job on the selected CSP infrastructure. All of these capabilities are driven by an MDE framework. However, the MDE abstractions are being developed and the realization as a web-hosted service is still under development. While this efforts is promising, they need to be tested and evaluated.

Other recent efforts like Deltacloud [14], Libcloud [15] and jclouds [16] have been developed to deal with API heterogeneity. These libraries hide away differences among multiple cloud provider APIs and allow users to manage different cloud resources through a unified common API. This has solved the multi-cloud problem in a very detailed manner, but the complexity is therefore even larger (i.e., users need to learn how to program using these APIs).

A model-driven approach for automating cloud deployment is also presented in [17]. Hamdaqa et al. have proposed a (5+1) architectural view model, where each view corresponds to a different perspective on cloud application deployment. This view model enables cloud stakeholders (e.g., providers, developers, administrators and financial managers) to leverage cloud platform capabilities. The (5+1) view model has been realized as a layered, domain specific modeling language (DSML), called StartusML, and the capabilities of this language have been illustrated using a representative domain example. The model was derived by investigating the process of architecting cloud applications, and then providing a set of meta-models to describe cloud applications within their ecosystem: an availability meta-model, an adaptation meta-model, a performance meta-model, a service meta-model, a workflow meta-model and finally a provider meta-model. Each meta-model in the (5+1) view model is dedicated a layer in StratusML. Our work has synergies with this work in the context of providing a user interface in order to facilitate the configuration and the description of the deployment model of applications in the cloud. Our deployment framework presents a fairly comprehensive DSML that allows the users to describe their applications deployment architecture in terms of services and interactions. It clarifies the cloud service model and its requirements in terms most cloud customers would understand. We developed also generative technologies to automate the de-

 ISBN: 978-1-61208-460-2

ployment process and to resolve the problem of the repetition of tedious tasks.

In the reminder of this paper, we will focus on presenting and evaluating our proposed contribution with respect to related work.

III. DEFINITIONS

This section gives a short overview of Model-Driven Engineering and its related concepts.

A. Model Driven Engineering

MDE is becoming an emergent software engineering paradigm to specify, develop and maintain software systems. In MDE, models are the primary artifact of the engineering process and are used, for instance, to (semi)automatically generate the implementation of the final software system.

According to the Object Management Group [6] MDE is a specific approach to software engineering that defines a theoretical framework for generating a code from models using successive model transformations [18]. The main goal of this approach is to separate the business side of a system from its implementation. The business model of a system can therefore drive its implementations on different platforms. In this way, we can expect to obtain better coherence between implementation and interoperability.

In brief, MDE aims to raise the level of abstraction in program specification and increase automation in program development. The best-known MDE initiative is the MDA proposed by the OMG [19].

B. Model Driven Architecture

MDA states that it models the environment and the requirements for a system in a Computational Independent Model (CIM). A CIM does not show the details of system structure. Thus, a CIM can be used to build a Platform Independent Model (PIM). A PIM focuses on the operation of the system while hiding details related to the use of a particular platform. PIM maintains platform independence in order to be suitable for use with different platforms. The transformation of a PIM into a Platform Specific Model (PSM) is based on the associated Platform Model (PM). A PSM is a system model for a specific platform. It combines PIM specifications with the details that specify how that system uses a particular platform. Figure 1 shows the main concepts used in MDA.

C. MDE for the Cloud deployment

Considerable attention has been focused recently on MDE as an alternative solution to overcome some of the deployment concerns in the cloud. In fact, the MDE approach holds promise in:

1) **Simplifying and (semi)automating the process of deployment of applications in the cloud**, by creating specific modeling languages/ tools that hide development complexity while also significantly reducing the learning curve involved in moving to a cloud platform. They allow accurate descriptions with a semantic precision.
2) **Ensuring portability and interoperability of systems across different platforms**, by developing generic and extensible cloud artifacts. This shields the users from the variabilities in CSPs.

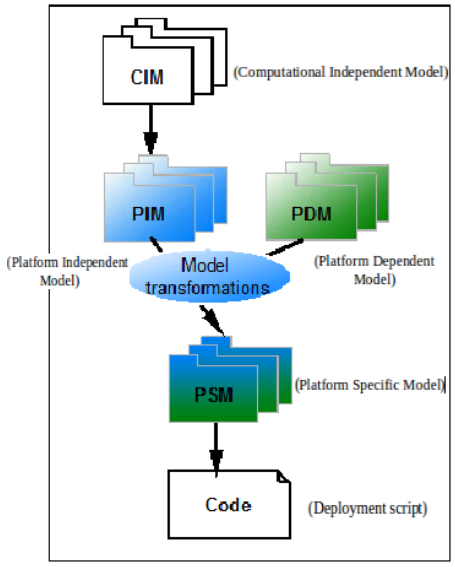

Figure 1. Main concepts of the MDA approach

In the literature, a number of recent papers have already explored this possibility as StartusML [17] and works done in [11][12][13]. Nevertheless, all of these works have focused only on resolving the heterogeneity problem. Our current research focuses on resolving all of the challenges mentioned above in Section 1 and on providing a complete solution for an automated deployment in the cloud.

IV. MODAC-DEPLOY: A MODEL DRIVEN FRAMEWORK FOR THE ASSISTANCE TO CLOUD DEPLOYMENT

Our goal is to provide a complete solution that assists software designers to configure and to deploy successfully their applications in the Cloud. In this section, we present more details about the MoDAC-Deploy architecture, giving the main steps required for deploying an application in the cloud and its capabilities.

A. Overview

The key idea of the MoDAC-Deploy framework is to simplify as much as possible the deployment process in the cloud by proposing an abstraction layer isolating applications from the underlying environment and hiding API details. In fact, shielding users from having to manually write scripts using low-level APIs hides the deployment complexity and dramatically reduces manual efforts and the time required to configure cloud resources. In addition, our framework has been designed specifically to support multiple CSPs in order to enable application portability among different CSPs. So applications can be easily moved from one cloud infrastructure to another which would satisfy more their needs without any additional efforts: ”model the software deployment once and deploy it anywhere”. In fact, users have only to change the selected provider from the available list presented in our framework and then reuse the same deployment model to

 ISBN: 978-1-61208-460-2

deploy their applications in the new chosen cloud infrastructure. Figure 2 shows our framework architecture. It includes three main modeling tools: *IaaSEditor*, *IaaSMetaModel* and *ScriptGenerator*. We have used Eclipse Modeling Framework to develop the DSML and the generative capabilities within our framework.

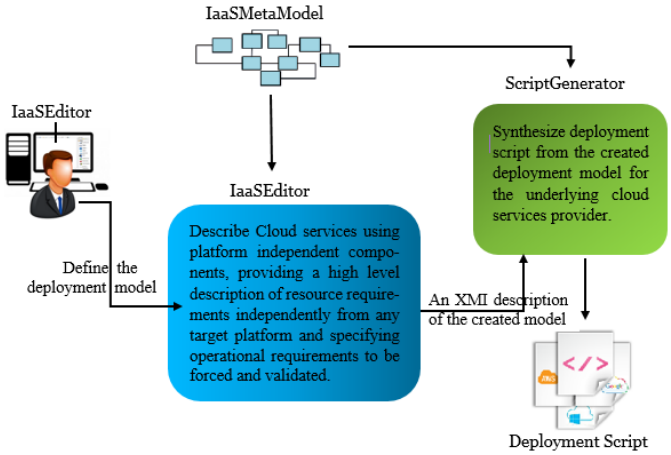

Figure 2. Overview of the MoDAC-Deploy approach

Both *IaaSEditor* and *ScriptGenerator* use the deployment meta-model *IaaSMetaModel* in order to guarantee the creation of a valid model and the generation of an executable deployment script.

B. Deployment process

Following MDA practices, an application deployment is achieved in three-step process:

First, users specify and define the hosting architecture of their applications through creating a new deployment model under *IaaSEditor*. The deployment model is saved as an XMI description consisting of several nodes (components). Each node may correspond to a virtual machine or to a storage medium, then it would be associated to a named group, namely a *SecurityGroup*, an *AutoScaling-Group* or an *AvailibilityGroup*. This deployment model is defined using the meta-model IaaSMetaModel. A simple validation should be done at this level to guarantee a valid deployment model and then an executable deployment script.

Second, a model transformation engine with specific rules is used to transform the preceding model into a CSP-specific model.

Finally, *ScriptGenerator* ensures the generation of the deployment script script.sh from the XMI document generated within *IaaSEditor*. The generated script presents an executable bash script which should be token later and executed on the command line interface of the underlying provider. Users have access to the generated script and they can identify possible values as wanted. Figure 3 illustrates the process that the MoDAC-Deploy framework goes through to facilitate and semi-automate the deployment into cloud infrastructures.

C. IaaSMetaModel

IaaSMetaModel is depicted in Figure 4. This meta-model captures all the concepts that are needed to specify a cloud

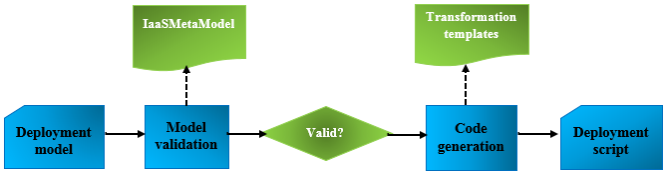

Figure 3. Deployment Process

deployment. It comprises multiple extensible and customizable classes, all of these classes are generic cloud artifacts, describing the functioning and the dependence of the different components and application services to be deployed on the cloud. The meta-model components and their responsibilities are as follows:

- *Hosting_Architecture*: is the main class of our deployment meta-model. It presents the hosting architecture of the application to be deployed in the cloud. Through this class, users can specify the cloud provider and their authentication credentials required by the provider. A good design of the hosting architecture is essential to have a successful deployment.
- *Application*: presents the name of the application to be deployed, its version, the URL designated by the developer to access to this application. It contains also entities named File such as text file, executable file, or any other library files to be uploaded onto the VMs that it is connected to. The application's set up and log files are copied from a local directory to another directory on a VM in the cloud.
- *VirtualMachine*: is used to define requested VMs from clients and to specify their characteristics such as the image ID, the VM size, the availability zone and the number of instances required to execute the application in the cloud. Through this class, we can also enable the monitoring of our instances VM.
- *StorageMedium*: we classify the storage mediums into three categories: *DataBaseStorage*, *SimpleStorage* and *VolumeStorage*. *DataBaseStorage* offers both relational and NoSQL database infrastructure. *SimpleStorage* provides a persistent storage of large amounts of distributed object, highly scalable, sustainable and available while *VolumeStorage* provides disk support, we can associate multiple disks to a virtual machine.
- *Group*: designates a collection of virtual machines or storage mediums with common characteristics, we distinguish between three group categories: *AvailibilityGroup*, *AutoScalingGroup* and *SecurityGroup*. A *SecurityGroup* consists of a set of access control rules that describe traffic filters to our VMs. It is analogous to an inbound network firewall, for which we specify the protocols, ports, and source IPs ranges that are allowed to reach the VM instances. An *AutoScalingGroup* presents scaling factors to apply on a set of virtual machines. The number of running VM instances can be dynamically scaled out and in, according to certain conditions in order to handle changes in traffic: it is possible to increase the size of a group of instances to meet a load peak or to

 ISBN: 978-1-61208-460-2

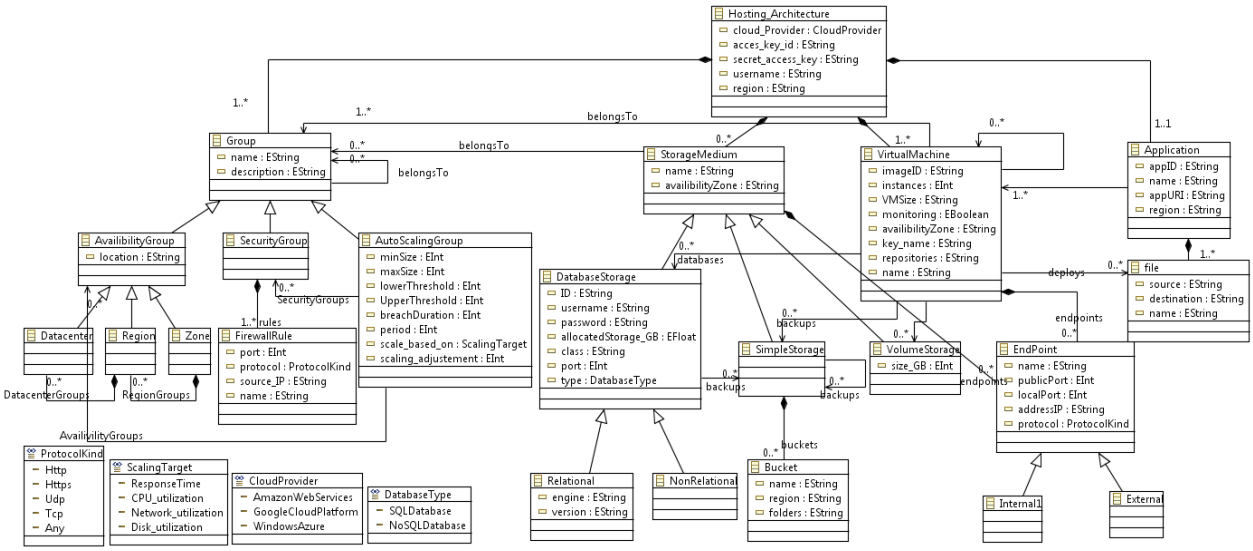

Figure 4. IaaSMetaModel

reduce the executing park in the case of the decrease of traffic. Finally, an *AvailibilityGroup* nests components which need to be hosted in the same location. It is a superclass for the three geolocation groups (i.e., Zone Region and Datacenter).

- *EndPoint*: handles incoming network traffic to cloud components. An endpoint is a URL that is the entry point for a web service. For each endpoint, we associate a range of IP addresses and the port through which a cloud component/ task can connect to others. An endpoint uses a specific protocol that determine the syntax and semantics of the messages that are exchanged between the two communication parties. An endpoint can be external if it is publicly visible or internal if it is only accessible within the cloud application. Also, each endpoint has a public port and a private port: the public port is used to listen for incoming traffic to the virtual machine from the Internet while the private port is used to listen for incoming traffic, typically destined to an application or service running on the virtual machine.

This meta-model was developed after inspecting manually three cloud infrastructures, namely Amazon Web Services, Windows Azure and Google Cloud Platform. Furthermore, adding additional provider concepts is designed to be relatively simple.

D. IaaSEditor

As described above, IaaSEditor provides an intuitive user interface that allows cloud customers to define their application services and to configure the target environment through a simple graphical modeling, shielding them from programming efforts. Once users have created the deployment model of their applications, they can choose any CSP supported by our framework and the created deployment model can then be reused to move the application into another Cloud infrastructure by changing only the selected CSP under IaaSEditor and some CSP-specific properties such as the VM image ID.

Figure 5 presents the editor *IaaSEditor*. it is composed of three layouts:

- **Design workspace :** Here users can design and validate their deployment models, and ask for the generation of the script deployment.
- **Palette :** it contains the different components to use in creating the deployment model, grouped together in different categories (Resources, Groups, Relations) according to their role.
- **Configuration Tabs :** Each tab opens a view that displays the properties of the selected element in the design workspace. These properties can be edited to change or set the parameters related to a particular component.

E. ScriptGenerator

We have used Acceleo, a code generation tool under the framework Eclipse, to implement our *ScriptGenerator*.

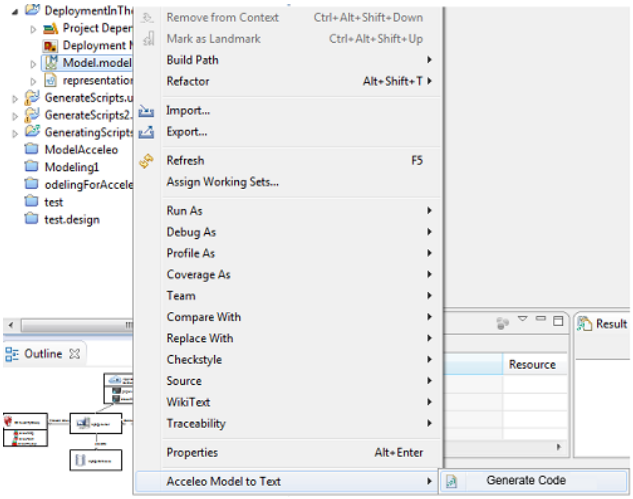

Figure 5. Generation of the deployment script

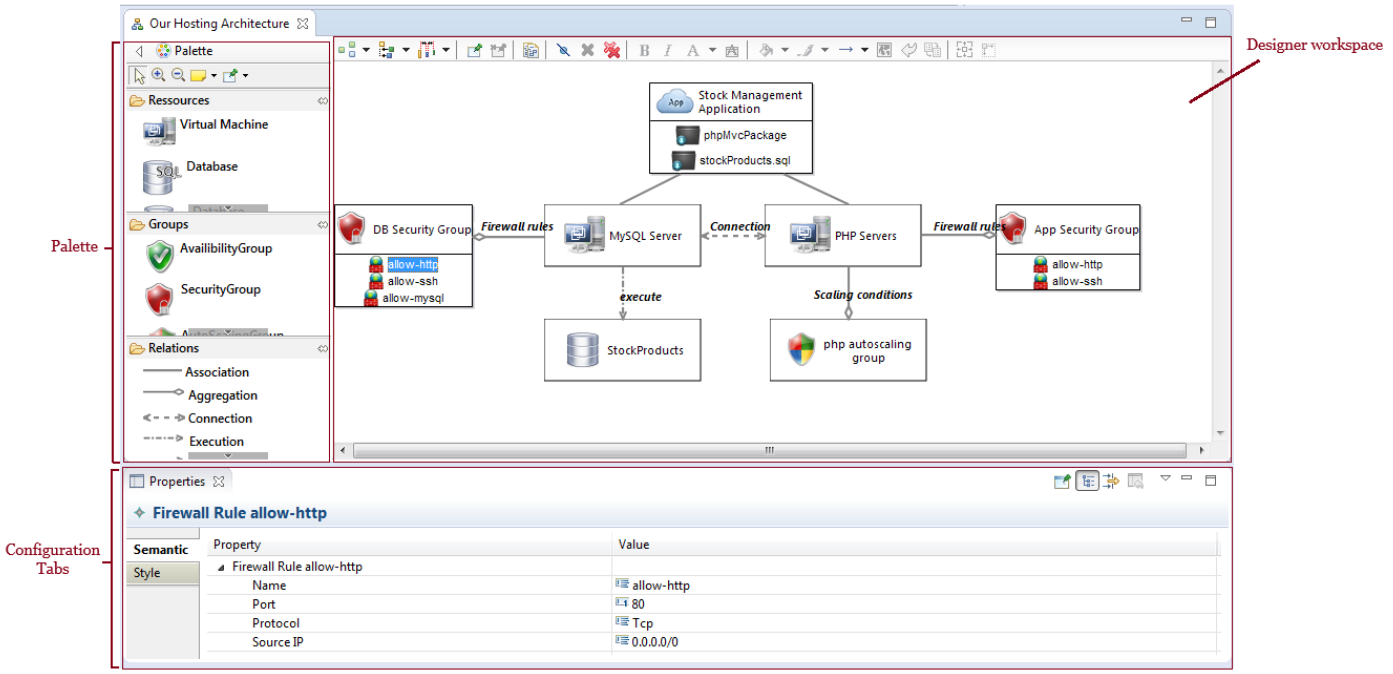

Figure 6. IaaSEditor

Users launch the generation of the deployment script with a simple click as shown in Figure 5.

Through a set of model transformation templates, *Script-Generator* synthesizes the deployment script from the deployment model created within the editor *IaaSEditor* for the underlying CSP (cf. Figure 6). In fact, templates convert data from the input model into the deployment script to configure cloud environment and to deploy the application.

In this Section, we described how the modeling capabilities are used to implement our framework and how the developed modeling tools can help to facilitate the deployment of services into cloud infrastructures. In the next Section, we will illustrate a case study in order to evaluate our deployment framework and to demonstrate its effectiveness.

V. CASE STUDY

We consider here a stock management application, sufficient to demonstrate the effectiveness of our deployment solution. we deployed this application across two different cloud infrastructures: Amazon Web Services (AWS) and Google Cloud Platform in order to underline the deployment portability among different CSPs.

To provision and to configure all necessary infrastructure, we need first to design the application architecture in order to ensure that it meets our requirements. This application is structured into logical tiers. The first tier is the web browser, which is responsible for presenting the user interface. The middle tier is an application server, which is responsible for the application's functionality. The third tier is a database server, which is responsible for data storage.

To this end, we defined two virtual machines and the number of instances to create from each one, one instance for MySQL (the database server) and 3 instances for Apache/PHP (application servers). In fact, we decided to deploy the MySQL database on an independent machine to properly manage the scalability of the application. Then we created a security group for each virtual machine to control and to filter the traffic allowed to reach the instances and we specified rules to each security group. In our case, we enabled inbound HTTP access from anywhere and inbound SSH traffic from our computer's public IP address so that we can connect to our instances. MySQL port was only opened for the Apache/PHP instances. In addition, an *AutoScalingGroup* is associated to PHP Servers in order to launch or terminate instances as demand on the application increases or decreases. So we configured auto-scaling to launch an additional Apache/PHP instance whenever CPU usage exceeds 60 percent for ten minutes and to terminate an instance whenever CPU usage under 30 percent. Every new added instance connect to the same MySQL database. Once our system was setup and configured, we installed needed repositories such as Apache 2 and PHP and we deployed our code to the application servers (PHP servers) and finally we associated a domain name with our web application. The final deployment model is presented in Figure 6.

Thanks to MoDAC-Deploy, all these steps are simply conducted by drag and drop operations and by filling properties in the tab "Configuration Tabs".

Figure 7 depicts the deployment script of the underlying application in AWS. The captured lines of code creates a MySQL instance and defines the characteristics of the virtual machine to be provisioned. We chose a linux image-32 bits for the MySQL server. It creates also a Key Pair, which presents the credentials we used to SSH into the box. Then, we opened the SSH port (22) and the HTTP port (80) for the

 ISBN: 978-1-61208-460-2

MySQL server. And finally, we imported the database backup file "stockProduits.sql" and we started the MySQL instance.

```
...
#Create a Key Pair for the VM MySQL Server :
aws ec2 create-key-pair --key-name key-pair-mysql-server --query 'KeyMaterial' --output text
    | out-file -encoding ascii -filepath key-pair-mysql-server.pem
chmod 400 key-pair-mysql-server.pem
#Create a Security Group for the VM MySQL Server:
aws ec2 create-security-group --group-name db-security-group-security-group
#Add  firewall rules:
aws ec2 authorize-security-group-ingress --group-name  db-security-group --protocol Tcp --
    port 80 --cidr 0.0.0.0/0
aws ec2 authorize-security-group-ingress --group-name  db-security-group --protocol Tcp --
    port 22 --cidr 0.0.0.0/0
#Create and Start the VM instances  "MySQL Server":
NUMBER_OF_INSTANCES=1
for i in $(seq 1 $NUMBER_OF_INSTANCES)
do
#Start an instance and Get her PublicDnsName:
PublicDnsName=$i=`aws ec2 run-instances --image-id ami-0c87ad78 --count 1 --instance-type 1
    --region us-east-1 --key-name key-pair-mysql-server.pem \
...security-groups  | grep PublicDnsName | awk -F": " '{print $2}' | sed s/\"//g |sed s/\,//g
```

Figure 7. The deployment script in Amazon Web Services

We reused the same deployment model to move the management stock application into the Google Cloud Platform. All what we did is to change the CSP as illustrated in Figure 8.

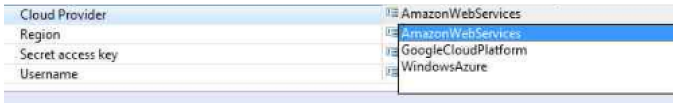

Figure 8. Cloud Services Provider Selection

We modified also the VM imageID from "ami-0c87ad78" in AWS to "https://www.googleapis.com/compute/v1/projects/ubuntu-os-cloud/global/images/ubuntu-1404-trusty-v20150316 " in Google Cloud Platform.

```
#!/bin/bash
#Download and install gcloud:
curl https://sdk.cloud.google.com | bash
sudo apt-get install googlecl
#Authenticate gcloud:
gcloud auth login
gcloud config set project application-de-gestion-de-stock
gcloud config set compute/region us-central1
# Virtual Machine:
#Create and Start the VM instances  "MySQL Server":
NUMBER_OF_INSTANCES=1
for i in $(seq 1 $NUMBER_OF_INSTANCES)
do
gcloud compute instances create mysql-server-$i --image https://www.googleapis.com/compute/v1
    /projects/ubuntu-os-cloud/global/images/ubuntu-1404-trusty-v20150316 \
--machine-type f1.micro  --zone us-central1-a --boot-disk-type "pd-standard"  \ ...
```

Figure 9. The deployment script in Google Cloud Platform

The generated deployment script is depicted in Figure 9.

VI. CONCLUSION AND FUTURE WORK

This paper presented the results of investigations on the main challenges of the deployment of applications in the cloud and on the modeling capabilities that can help to implement our proposed solution.

So, we have developed a model-driven framework for cloud deployment, which facilitates and semi-automates the deployment of services to cloud infrastructures. This reduces the deployment complexity and errors that can occur during manual configurations as well as costs. It helps also to shield users from complex programming efforts, the low- level API details and from the heterogeneity in cloud providers. In addition, our solution enables application portability between different clouds and allows to minimize the vendor lock-in. Generated script can be executed only on unix machine, we are currently working on generating deployment scripts running on windows.

As a minority of providers that offer autoscaling capabilities to automatically add or remove virtual machines from an instance group based on increases or decreases in load as Amazon Windows Azure (within an Auto Scaling group), Google Cloud Platform (i.e., define the autoscaling policy and the autoscaler performs automatic scaling) and Windows Azure (through configuring the autoscale status), our next research thread will definitely revolve around this feature, we plan to develop algorithms and techniques for dynamically scaling deployments in response to application demand in other IaaS and for re-configuring deployments. Cloud computing comes with a cost where 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. In that context, developing mechanisms for estimating deployment performance and cost and selecting the proper cloud deployment is an issue to be addressed in ongoing work.

Besides, the current capabilities presented by this framework can be extended further to handle complex architectures such as network applications by adding new cloud artifacts and why not make it able to deploy multi-cloud architectures (i.e., deploying applications across multiple cloud providers, e.g., deploy a single workload on one provider, with a backup on another). We plan also to move in the direction of making the deployment DSML as mature and complete by covering new CSPs as well as private IaaS.

REFERENCES

[1] M. Hamdaqa, T. Livogiannis, and L. Tahvildari, "A reference model for developing cloud applications," in CLOSER, 2011, pp. 98–103.

[2] R. Gadhgadhi, M. Cheriet, A. Kanso, and S. Khazri, "Openicra: Towards a generic model for automatic deployment and hosting of applications in the cloud," in IJ-CLOSER, 2013, pp. 249–275.

[3] Amazon Web Services, http://aws.amazon.com, [Accessed 09 November 2015].

[4] Google Cloud Platform, http://cloud.google.com, [Accessed 24 October 2015].

[5] Windows Azure, http://azure.microsoft.com, [Accessed 04 November 2015].

[6] OMG, Object Management Group, http://www.omg.org.

[7] G. Juve and E. Deelman, "Automating application deployment in infrastructure clouds," in CLOUDCOM '11, pp. 658–665.

[8] Amazon, Elastic Compute Cloud (EC2), http://aws.amazon.com/ec2, [Accessed 09 November 2015].

[9] D. Nurmi et al., "The eucalyptus open-source cloud-computing system," in CCGRID '09, 2009, pp. 124–131.

[10] OpenNebula, http://www.opennebula.org, [Accessed 27 October 2015].

[11] F. Caglar, K. An, S. Shekhar, and A. Gokhale, "Model-driven performance estimation, deployment, and resource management for cloud-hosted services," in DSM '13, 2013, pp. 21–26.

[12] J. Cala and P. Watson, "Automatic software deployment in the azure cloud," in Distributed Applications and Interoperable Systems, 2010, vol. 6115, pp. 155–168.

 ISBN: 978-1-61208-460-2

[13] S. Shekhar et al., “A model-driven approach for price/performance tradeoffs in cloud-based mapreduce application deployment,” in MODELS, 2013, pp. 37–42.

[14] Deltacloud, https://deltacloud.apache.org/, [Accessed 18 September 2015].

[15] Libcloud, http://libcloud.apache.org/, [Accessed 22 September 2015].

[16] jclouds, http://jclouds.apache.org/, [Accessed 28 September 2015].

[17] M. Hamdaqa and L. Tahvildari, “The (5+1) architectural view model for cloud applications,” in 24th CSSE, 2014, pp. 46–60.

[18] J. Bézivin, “Model driven engineering: An emerging technical space,” in Generative and Transformational Techniques in Software Engineering, 2006, pp. 36–64.

[19] MDA, Model Driven Architecture, http://www.omg.org/mda, [Accessed 27 November 2015].

 ISBN: 978-1-61208-460-2