International Journal of Electrical and Computer Engineering (IJECE)
Vol. 6, No. 2, April 2016, pp. 708~716
ISSN: 2088-8708, DOI: 10.11591/ijece.v6i2.8270

A Model Driven Framework for Portable Cloud Services

Aparna Vijaya*, Neelanarayanan V*
* School of Computer Science and Engineering, Vellore Institute of Technology Chennai Campus, India

Article Info

Article history:

Received Jun 4, 2015
Revised Nov 28, 2015
Accepted Dec 17, 2015

Keyword:

Feature Model
Heterogenous cloud
Model driven development
Platform agnostic

ABSTRACT

Cloud Computing is an evolving technology as it offers significant benefits like pay only for what you use, scale the resources according to the needs and less in-house staff and resources. These benefits have resulted in tremendous increase in the number of applications and services hosted in the cloud which inturn has resulted in increase in the number of cloud providers in the market. Cloud service providers have a lot of heterogeneity in the resources they use. They have their own servers, different cloud infrastructures, API's and methods to access the cloud resources. Despite its benefits; lack of standards among service providers has caused a high level of vendor lock-in when a software developer tries to change its cloud provider. In this paper we give an overview on the ongoing and current trends in the area of cloud service portability and we also propose a new cloud portability platform. Our new platform is based on establishing feature models which offers the desired cloud portability. Our solution DSkyL uses feature models and domain model analysis to support development, customization and deployment of application components across multiple clouds. The main goal of our approach is to reduce the effort and time needed for porting applications across different clouds. This paper aims to give an overview on DSkyL.

Corresponding Author:

Aparna Vijaya
School of Computer Science and Engineering,
Vellore Institute of Technology Chennai Campus,
Chennai 600127, Tamil Nadu, India.
Email: aparnav85@gmail.com

1. INTRODUCTION

Software industry is now in a transition phase from desktop computing to cloud computing. Cloud computing offers several benefits including virtualized hardware infrastructure, user self provisioning, elasticity and pay per use. In this era of cloudification there is enormous number of cloud providers in the market. The market keeps moving on and the ability to survive in a market crowded with cloud vendors becomes a tedious task. Every cloud vendor try to introduce flexibility in their offerings so as to adapt to the market and to survive. This flexibility creates a lot of confusion among the users and thereby choosing a cloud provider becomes more or less a guessing game. User needs might be satisfied at inception by a cloud vendor, but user needs change based on their business escalations. At this point switching cloud providers can come at a cost. There might be hidden lock-in costs with a specific cloud provider or proprietary service model such as the cost of moving code and data from one cloud service provider to another might be prohibitive. It is not only moving into and out of a cloud environment a costly endeavor, but tasks such as refactoring the code can result in unexpected costs. In some scenarios, this switch needed major changes in software and caused project delays and even productivity losses.

Cloud computing [1], [2] enables an organization to deploy and run their applications in a platform maintained by third party cloud providers rather than maintaining their own data centers. Organizations (customers) rent the resources of the cloud, IaaS on which applications can be deployed or PaaS where

developers can create their own SaaS applications. Finally, end users access and use the deployed SaaS applications. In this scenario of cloud adoption, users of PaaS services are raising the question of the portability of their applications from one provider to a different one, or even back to the data center. Portability is the ability to move applications among different platforms without having to rewrite it partly or fully. It is the prerequisite for building truly agile and flexible systems that do not lock in their users. Due to the enormous number of cloud providers in the market and the increasing number of services offered or created, portability of services is now becoming a major concern.With the cloud becoming more competitive and some providers are at long-term stability risks, developers or customers have to understand that adhering to a specific vendor's Platform as a Service (PaaS) can be a risk because of the uneven support for platform features by different cloud providers. When there are multiple options in private as well as public cloud, it is a very challenging task to identify the best suited cloud provider for an organization. Providers have tried introducing new services, entered the open source world, merged with giants in the cloud market and so on to withstand the competition. They have tried reducing the price as they challenge each other to see how low they can go. This battle led many vendors fail in the market. When we have a diversified cloud strategy it gives multiple options so that we can always handle some of the unforeseen risks. Sometimes the cloud vendors fail to offer continual cloud services. As a result the application that is hosted in the cloud becomes inaccessible affecting business continuity. If the services are portable the chances of resource sharing between the clouds are high which will improve business continuity and reliability.

Several open standards, APIs and tools have been proposed [5] [6] [7] [8] [9] [10] [11] in recent years for achieving cloud portability. A comparative study of various approaches has been done in [1], [4] and is consolidated in the following table.

Table 1. Comprative Study of Existing Approaches

Approach Name	Languages supported	Data Support	OS	Clouds Tested	Vendor Independent	Methodology adopted
OCCI	Java, Ruby, erlang	Amazon	Cross platform	OpenNebula, mnesia	Yes	Cloud-specific standards
SimpleCloud	PHP	Amazon S3 and Nirvanix IMFS	Cross platform	Zend Cloud	No	API
deltaCloud	Ruby	Amazon, swift, walrus	Cross platform	RackSpace, OpenNebula, AWS,GoGrid	No	API
jCloud	java	Amazon	Cross platform	RackSpace, HpHelion, AWS,GoGrid	No	API
mOSAIC	Java, Python, erlang, node.js	Riak, CouchDB, MemcachDB, Redis, MySQL, Amazon S3, HDFS	Linux	Amazon EC2, OpenNabula, Eucalyptus,	Yes	Multiagent Systems /tool
OASIS TOSCA	Java, PHP	MySQL	Linux	OpenStack	Yes	Cloud-specific standards
Docker	Java, Ruby, Perl, Python, node.js	Riak, Cassandra, MongoDB	Linux	Azure, AWS, OpenStack		Container / tool
MODA Clouds	Java, Python, erlang, node.js	Riak, CouchDB, MemcachDB, Redis, MySQL, Amazon S3, HDFS	Linux	Eucalyptus	Yes	Model-driven application engineering
Openshift	Java, Python, perl, Ruby, PHP, .NET	MongoDB, MySQL, PostgreSQL, Microsoft SQL Server	Linux	-	Yes	Hybrid Platform as a Service
Artist	Java	MySQL	Cross platform	-	Yes	-

2. PROPOSED APPROACH

For achieving application portability we have used the technique of model driven software development. Model driven software development aims to increase the visibility of knowledge compared to the traditional process by explicit representation of information through models. It provides knowledge at different levels of abstraction; that is it separates the conceptual details from the implementation details**.** It

also promotes automatic generation of code to overcome the abstraction gap between the models and the implementation platform. The models also act as a common communication term between stakeholders and business during requirements and design.

Feature modeling is independent of the underlying programming language. However we have used Java as end user programming language. The main reason for choosing Java is the greater flexibility compared to a custom domain specific language (DSL) and it also improves reusability of domain objects. DSkyL's [1] [4] user interface has three panels - the diagram editor, the property panel and the package explorer panel. The diagram editor displays the diagram and all the elements it contains, and allows the user to drag and drop new features and relationships in the editor. The properties panel allows users to view and modify the properties of every element. Package Explorer displays the element location or element hierarchy.For each element which is dropped into the diagram panel, the string given in the property panel is interpreted, and the graphical representation of that element is modified according to this interpretation. This is achived using java reflections API. The code is compiled continuously in the background, and immediate feedback is provided to the end user by redrawing the element. If the code contains errors, the affected source code lines are highlighted.

The topology of the application is represented in terms of features as a Feature Model. Feature model represents a tree whose nodes are the application components or features, and whose edges are the relations between these application components as specified in Figure 1. In software product line engineering, one can configure the product by selecting the desired features from a feature model based on customer's functional requirements. The commonalities and variations are identified during domain analysis and modeled as features in a feature model. In DSkyL feature model is represented as a feature tree where nodes represent features and edges represent the "selection" relationships among features. From a feature model, a specific variation of a product can be derived by selecting the desired features based on customer's requirements and feature relationships can be specified in the feature model. We can have any number of configuration files. But only one will be the current. With the implementation of the domain there are two variants: From top to bottom or from the bottom to the top. Top down approach starts with the concept in the broadest sense which then includes (contains) others. In the second case, we start with the basic concepts. We have used the top down approach in DSkyL. So we begin with the concept of Moodle application. To create an application, feature modules are composed together. The file containing the class definition we put into the folder Classes/Domain/Model/ and rename it according to the last part of the class name: Moodle.php.

A new feature is created with the following attributes and then the business logic is added to it.

```
public Feature(FeatureModel featureModel, String name) {
                this.featureModel = featureModel;
                this.name = name;
                this.mandatory = false;
                this.concret = true;
                this.multiple = false;
                this.hidden = false;
                this.constraintSelected = false;
                this.color = new color (this);
                this.location = new featurePoint (0, 0);
                this.description = null;
                this.parent = null;
        }
```

Out tool supports different types of relationship between these features which acts as our domain objects. A few of them along with their notations are represented below:

- Optional
- Mandatory
- Alternative (OR)
- Requires
- Uses
- Implementation
- Generalization
- Composition

Multiplicities are also added to thefeature model. We use the multiplicity notation to express the constraint for the selection of features from the set. The multiplicity 1...* tells us, we have to choose at least one of the possibilities, but we might also choose all of them. There are rules which are associated to

features. During the validation of domain objects the business logic often looks for the properties of the objects present. This comes into effect during feature selection. Some rules which are present in our model are:

"X ALWAYS Y: = If X is selected then Y is selected in every valid configuration."

"X MAYBE Y: = If X is selected then Y is selected in at least one but not all valid configurations."

"X NEVER Y: = If X is selected then Y cannot be selected in any valid configuration."

As a first step features and their relationships are identified by understanding and analyzing the domain. Then features are represented using the feature models. Based on the features which are selected, specific programs are composed. Each feature will be implemented as a different module to enhance reusability. Thus variants of an application are created by selecting and implementing features as per the requirements. In Figure 1b, we depict a simple example of three classes and four feature modules. The classes are associated to different features. In order to generate a program, classes of the selected features are composed with a tool. This way, many different programs can be created from a set of features.

Having the application topology ready, the next step is to model deployment topology and build plan as specified in Figure 1 and 2. DSkyL uses BPMN notation to model the build plan. For deployment, a BPMN workflow that provisions the Moodle application on Amazon EC2 virtual machines is attached. The workflow installs the applications as defined in the topology and establishes the connectsTo relation by assigning the IP address of the MySQL instance to the Moodle configuration on the Apache Web Server. After finishing modeling, the backend allows for exporting a CSAR file containing all required definitions. The resulting CSAR file deploys the implementation artifacts and the management plans to appropriate runtime environments. Finally, the users can instantiate an application instance.

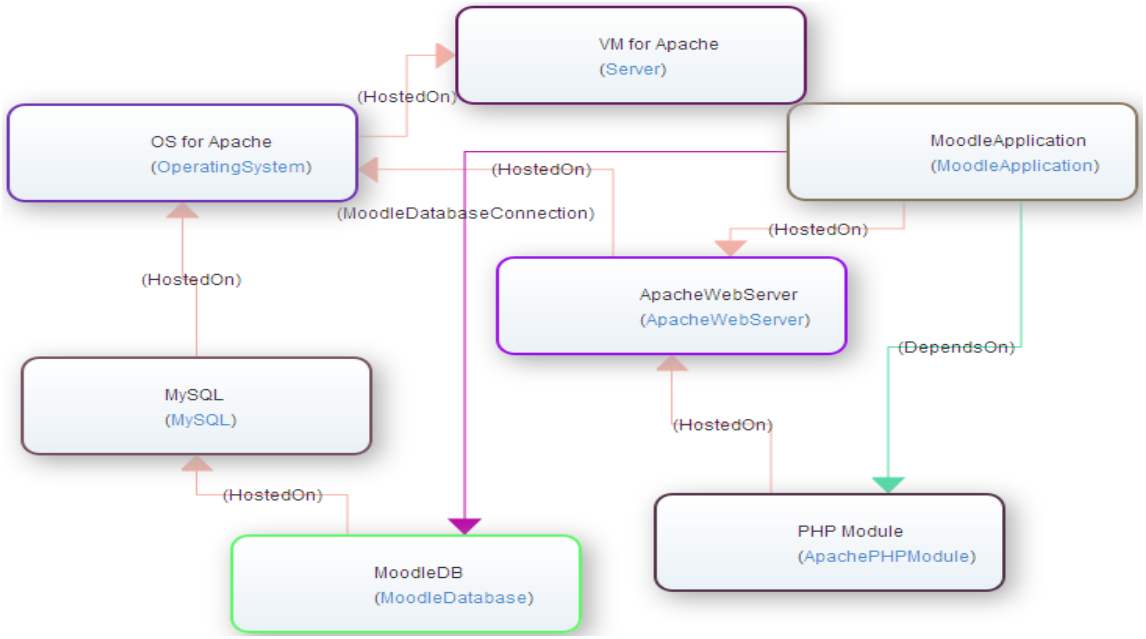

Figure 1. Deployment Model for Moodle

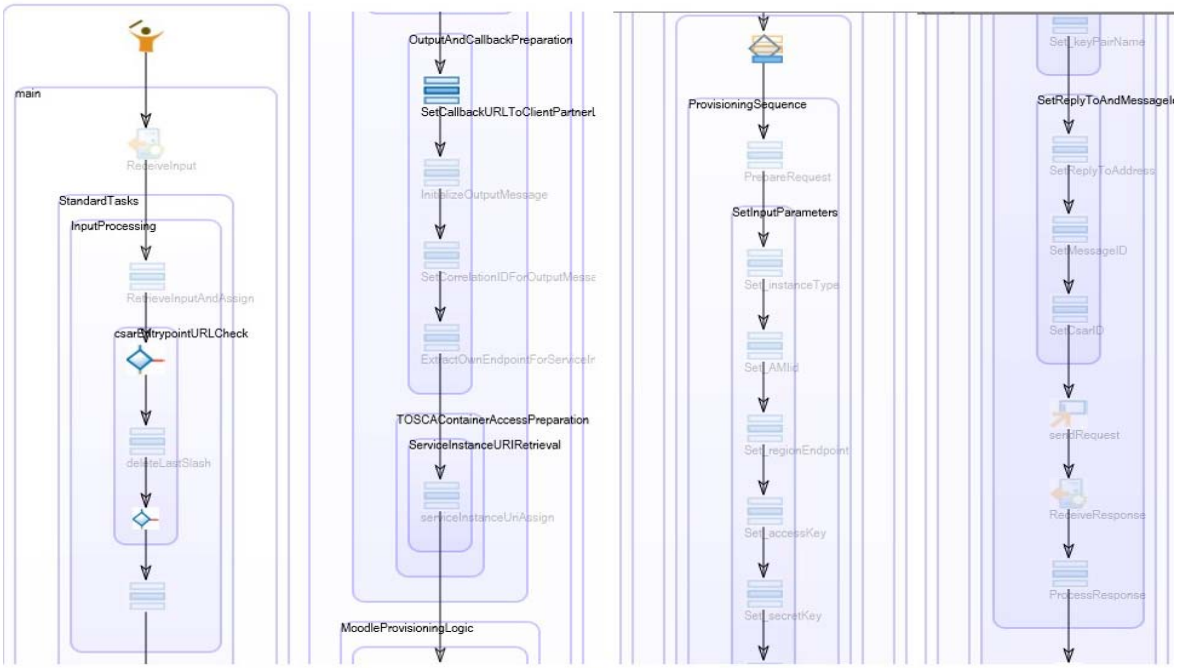

Figure 2. Build Plan for Moodle

The Cloud Service Archive [12], [13] is a container file which holds the service template of a cloud application, all artifacts required to manage the lifecycle of the corresponding cloud application (i.e. the implementation artifacts of the operations of the node types) as well as all artifacts to execute the cloud application (i.e. the deployment artifacts of the node types like virtual images, EJBs, WSDL files, SQL DDL etc). These files are typically organized in several subdirectories each of which contains related files. Each CSAR must contain a subdirectory called Meta-Inf. This subdirectory must contain a so-called manifest file. This file is named MANIFEST and has the file extension .MF. It represents metadata of the other files in the CSAR. These metadata are given in the format of name/value pairs. These name/value pairs (<*name*>*:* <*value*>) are organized in blocks. Each block provides metadata of a certain artifact of the CSAR. The first block of the manifest file provides metadata of the CSAR itself (e.g. its version, creator etc) as follows:
Manifest-Version: x.x
CSAR-Version: x.y
Created-By: test
Entry-Service-Template: file name that is the entry point for the cloud application.
Each other block begins with a name/value pair that point to an artifact within the CSAR by means of a path-name as shown in Figure 3.

```
Name: Definitions/tst__MySQL.tosca
Content-Type: application/vnd.oasis.tosca.definitions

Name: Definitions/tst__ApachePHPModule.tosca
Content-Type: application/vnd.oasis.tosca.definitions

Name: Definitions/tbt__Database.tosca
Content-Type: application/vnd.oasis.tosca.definitions

Name: Definitions/brt__test_type.tosca
Content-Type: application/vnd.oasis.tosca.definitions

Name: Definitions/ns93__PhpApplication.tosca
Content-Type: application/vnd.oasis.tosca.definitions

Name: Definitions/tbt__HostedOn.tosca
Content-Type: application/vnd.oasis.tosca.definitions

Name: Definitions/tbt__DependsOn.tosca
Content-Type: application/vnd.oasis.tosca.definitions

Name: Definitions/tbt__OperatingSystem.tosca
Content-Type: application/vnd.oasis.tosca.definitions

Name: Definitions/tst__ApacheWebServer.tosca
Content-Type: application/vnd.oasis.tosca.definitions

Name: Definitions/tst__MySQLImplementation.tosca
Content-Type: application/vnd.oasis.tosca.definitions
```

Figure 3. Manifest *file*

The structure of applications is defined by a topology a graph of typed nodes and directed typed edges. Nodes represent components forming an application / features and edges define the relations and dependencies between them. For instance, the topology of the moodle application (Figure 1) consists of the actual PHP module, an Apache Web Server, a MySQL database, two operating systems (one for the Web server and one for the MySQL database), and two virtual machines. The relationships in this topology define, for instance, that the Moodle application is hosted on a Web server and that the application connects to the MySQL database. The types of nodes and relationships specify their properties and management operations. The type ApacheWeb Server defines properties, such as port or version, and management operations, such as start or deploy. The actual implementation of a node is provided by one or many Deployment Artifacts, e. g., a Linux VM image, an operating system package for the Apache Web Server, or an archive containing the PHP files of Moodle. In addition, types may define Implementation Artifacts that implement the management operations for the respective element. The application topology and related artifacts are bundled into a Cloud Service ARchive (CSAR), which is a standardized packaging format for applications. Application topologies can be processed in an imperative or declarative way: Imperative processing relies on the implementation of management plans that can be executed fully automated to perform the desired management task, e. g., to instantiate, backup, upgrade, or terminate an application. These high level management tasks are implemented by orchestrating low level management operations provided by Implementation Artifacts of nodes and relationships. Since the management plans are typically implemented by the application developer, they enable operators to manage the application by running predefined plans without the need to understand all the technical details of the management task. Technically, management plans are implemented as workflows. Declarative processing, on the other hand, shifts the deployment and management logic from plans to the runtime. To perform the aforementioned high-level management tasks, the runtime has to know the operations that have to be called and their order. Declarative processing is well suited for the deployment of simple applications but is not able to facilitate complex management tasks for various kinds of application structures.

In order to port an application,

- Specify the nodes and node types and model the application service topology.
- Create artifacts of the software which needs to be installed.
- Identify and describe the relationship between the components.
- Generate the build plan and write the respective parameter to invoke the service.
- Upload it to the container and invoke the service to use the application.

Portability is ensured by the two engines working together when binding management plans. Strict separation of architectural components through welldefined OSGi interfaces enables the replacement of implementations of components. This also allows each component to be scaled independently. After uploading the CSAR, the deployment of the application follows three steps:

1. First, the CSAR is unpacked and the files are put into the Files store, which is backed either by the local file system or Amazon S3.
2. Then, the XML files are loaded and processed to deploy the related dependencies.

3. Finally, the portabilityAPI reads the topology model, instance data, properties of nodes like the port of web server etc and relationships to provision and configure the Cloud application according to the build plan.

The deployed application can be instantiated by executing the build plan of the application. Credentials (e. g., for Amazon EC2) or configurations (e. g., machine size) are passed as input message to the workflow. To instantiate Moodle, the build plan first starts two virtual machines with a Linux operating system and installs Apache Web Server and MySQL on them. Then, it uses the respective management operations to install the PHP application, import the database schema, and establish the database connection. After completion, a build plan may return certain information, for example, the web address of the deployed application instance. The Moodle build plan returns the URL of the running Moodle instance, which includes the public URL of the virtual machine running the Apache Web Server. The screen shots of implementation for porting application from HP Horizon and Open Stack to Amazon EC2 are shown in the following figures.

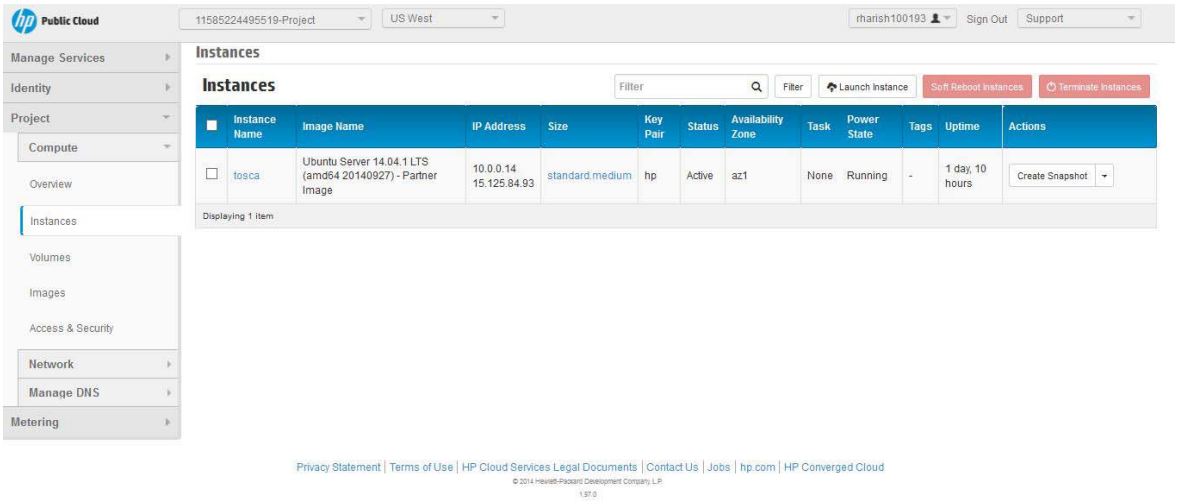

Figure 4. HP Helion Instance

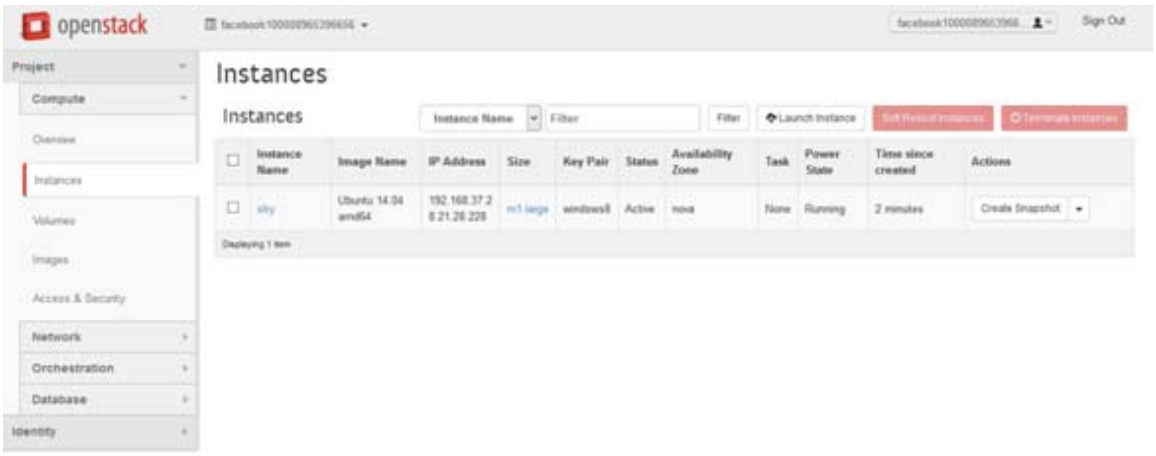

Figure 5. OpenStack instance

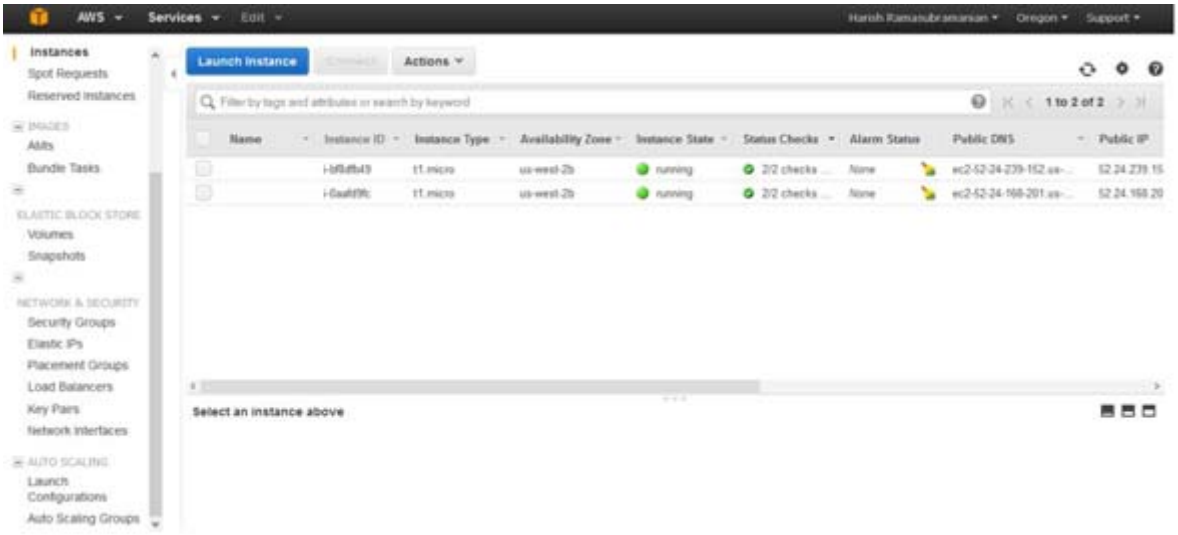

Figure 6. Amazon Web Service instance

3. PRELIMINARY EVALUATION

We have ported the Moodle Application from two different cloud environments Openstack and HP Horizon Cloud to Amazon AWS and results imply that the application is deployed on Amazon AWS successfully. The tables below showcase the time difference and platform used for porting.

Table 2. Tier Node Definition Template

Server Property	OpenStack	HP Horizon
Number of CPUs	4	2
Memory Size	8GB	4GB
Disk Size	80GB	50GB
Initial instances	1	1
Security Rule Protocols	HTTP	HTTP
Security Rule Ports	22,80,443	22,80,443

Table 3. Time Evaluation for porting Moodle

Source Server	Transfer Direction	Time
HP Horizon	Amazon EC2	331sec
Openstack	Amazon EC2	327sec

We have also tried to estimate the effort required for porting the application to cloud using Cloud Migration Point (CMP) [14]. The final value of CMP is determined as a weighted sum of its four components CMP_i with $i \in$ {connection, code changes, installation and configuration, database}. The complexity level for connections that will be affected during porting is LOW since livemigration is not considered in the scope of the work. It is considered that an application is ported from LAN to LAN. The code changes are mainly applicable in classes that are responsible for communication between external systems. Query modification tasks associated to database changes will not be accountable here since the same database will be ported to the taget cloud. Tasks related to installation and configuration of third party library and database does not require any manual effort but changes need to be done in the build plan and deployment model accordingly. For the Moodle application portability we have identified 36 Cloud Migration Points and the effort in hours is less than 3 hours.

4. CONCLUSION

This paper presented DSkyL, a tool developed for application portability across multiple clouds by using model driven architecture. We described the problem handled by the tool in section 1 and summarized the method behind the tool and its main functionalities with its evaluations in sections 2 and 3. Our approach was tested with three cloud vendors. Our goals for future work include the improvement of DSkyl by

supporting provision for data portability. Further work should also address more complex applications supporting addition and removal of features and cloud services on the go in order to ensure that realistic instances of requirements are being used, through direct contact with the business and customers.

REFERENCES

[1] Gurudatt Kulkarni Cloud Computing Software as Service, *International Journal of Cloud Computing and Services Science (IJ-CLOSER),* Vol 1, 2012 pages 11-16.

[2] Pratiyush Guleria Guleria, Vikas Sharma, Manish Arora, Development and Usage of Software as a Service for a Cloud and NonCloud Based Environment An Empirical Study, *International Journal of Cloud Computing and Services Science (IJ-CLOSER)* ,Vol 2 No 1, 2013 pages 50-58.

[3] Aparna Vijaya, Neelanarayanan V, *"Framework for Platform Agnostic Enterprise Application Development Supporting Multiple Clouds"*, Proc. Symp of BigData and Cloud computing Challenges - Elsevier Procedia Computer Science , 2015, Volume 50, pp 73-80.

[4] Aparna Vijaya, Pritam Dash, Neelanarayanan V, "Migration of Legacy Enterprise Applications to Multiple Clouds: A Feature based approach". *Lecture Notes on Software Engineering* (LNSE, ISSN: 2301-3559, DOI: 10.7763/LNSE) Journal.

[5] N. Loutas, E. Kamateri, and K. Tarabanis, ―*A Semantic Interoperability Framework for Cloud Platform as a Service*, IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom), Athens, 2011, pp. 280–287.

[6] Fotis Gonidis, Iraklis Paraskakis, Dimitrios Kourtesis, *Addressing the Challenge of Application Portability in Cloud Platforms*, Balkan Conference in Informatics, BCI '13, Greece, September, 2013.

[7] D. Petcu, G. Macariu, S. Panica, and C. Crăciun, ―Portable Cloud applications—from theory to practice, *Future Generation Computer Systems*, 2012.

[8] Magdalena Kostoska, Marjan Gusev, Sasko Ristov, A New Cloud Services Portability Platform, *24th DAAAM International Symposium on Intelligent Manufacturing and Automation*, 2013.

[9] Danilo Ardagna, Elisabetta Di, Giuliano Casale, Dana Petcu , Parastoo Mohagheghi, S´ebastien Mosser, Peter Matthews, Anke Gericke, Cyril Ballagny, Francesco D'Andria, Cosmin-Septimiu Nechifor, Craig Sheridan, *MODACLOUDS: A Model-Driven Approach for the Design and Execution of Applications on Multiple Clouds*, MiSE-2012.

[10] Redhat: http://www.redhat.com/developers/openshift/ (2015).

[11] Fotis Gonidis, Iraklis Paraskakis, Anthony J. H. Simons, Dimitrios Kourtesis, *Cloud Application Portability: An Initial View,* Balkan Conference in Informatics, BCI '13, Thessaloniki, Greece, September, 2013.

[12] Tobias Binz, Uwe Breitenb¨ucher, Florian Haupt, Oliver Kopp, Frank Leymann, "TOSCA: Portable Automated Deployment and Management of Cloud Applications". http://www.iaas.uni-stuttgart.de/RUS-data/INBOOK-2014

[13] Gerd Breiter, Frank Leymann, Thomas Spatzier, " Topology and Orchestration Specificationfor Cloud Applications (TOSCA):Cloud Service Archive (CSAR) Version 0.1", 2012.

[14] Van T.K. Tran, Kevin Lee, Alan Fekete, Anna Liu,Jacky Keung, *Size Estimation of Cloud Migration Projects with Cloud Migration Point (CMP)*, International Symposium on Empirical Software Engineering and Measurement, 2011.