An Interoperability Ontology for Multi-Cloud Computing Platforms

Surachai Huapai, Thepparit Banditwattanawong
School of Information Technology, Sripatum University, Thailand

***Abstract*—In a multi-cloud environment, there are usually several cloud computing platforms with different features being deployed in the same or different organizations.This leads to the need to interoperate between different cloud platforms. This paper presents an ontology towards a cloud broker system to achieve the interoperability among multi-cloud platforms specifically OpenStack and CloudStack. We validated our ontology by means of platform command data fulfillment against the ontology. The result was that our ontology supported thecommands of both of the platforms.**

Keywords—multi-cloud computing;cloud broker; ontology; openstack; cloudstack

I. INTRODUCTION

There are several reasons that mandate multi-cloud computing as the following examples. First, when different organizations become partner and cloud resource sharing is mutually approved. Second, when there are unique features (e.g., server application images, operating system images, tools, and security offered) offered by different cloud computing platforms. Third, to prevent platform lock-in problem in risk management or financial management. [1] In these situations, not only multiple cloud platforms co-exist but also their resources are shared as a pool.

One of the techniques used to accomplish multi-cloud platform interoperability issue is by using cloud broker [2]. A cloud broker is a system that is capable of communication between two or more different cloud platforms to achieve a given task. This paper focuses on two popular cloud computing IaaS platforms, OpenStack [3] and CloudStack [4]. For a practical example, when users want to create a virtual machine (VM) of certain main memory, storage capacity, operating system image and probably development kit, they cloud simply issue a GUI-based requirements via a cloud broker that in turn figures out the available resources that most match the user requirement and finally creates the VM as commanded by the broker.

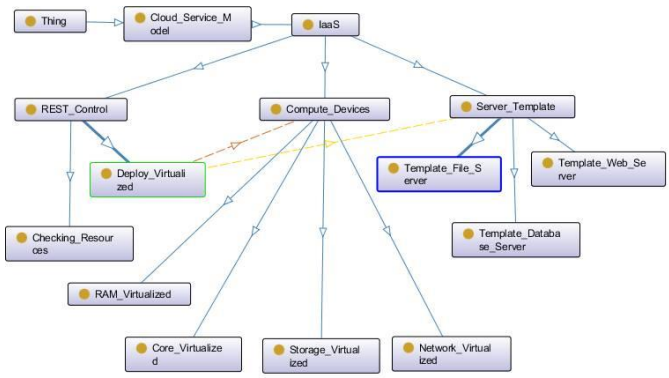
As exemplified above, the broker system must have a knowledge base for recognizing the user requirementsand transforming them into valid administration commandsfor desired cloud computing platforms. The knowledge base can be represented in the form of an ontology, which is defined as a formal explicit description of concepts (classes) in a domain of discourse, properties of each concept, and (role) restrictions on properties. [5] Any ontology can be verified and validated in three standard ways: application implementation, data storing and experts. This research proposes a novel ontology for interoperability between OpenStack and CloudStack towards the cloud broker system.

As for related works, there are several papers in the field of cloud computing interoperability such as [1, 2, 6]. However, they do not support the interoperability between OpenStack and CloudStack at the same time. There are also works pertained to cloud ontologies including [7, 8, 9]. Nevertheless, they aim for resource description, service discovery or security instead of interoperability.To recap, there is no existing workon an ontology specifically developedfor OpenStack and CloudStack interoperability, which is our contribution.

II. A PROPOSED ONTOLOGY

Our proposed ontology contains threeconcepts of IaaS service model as showed in Fig.1: computing devices, REST control, and server template.Each of the concepts has associated properties as will be detailed later on. We developed our ontology by using Protégé software tool [10].

Fig. 1. All classes in our proposed cloud interoperability ontology.



The three classes of the ontology are described as follows.

- First, class Compute_Devices represents totally available VM resources that comprise four subclasses representing virtualized CPU cores, virtualized RAMs, virtualized storage space, and virtualized network. Any resources to be consumed on both of the IaaS platforms must be registered with this class so that a resource pool is created for cloud broker.

- Second, REST_Controlclass represents the RESTful platform command sets of CloudStack and OpenStack. This class has two subclasses forVM deployment commands and resource checking commands where the cloud broker system selectively fetches to issue commands to corresponding target cloud platforms.
- Finally, class Server_Template represents the server environment configurations and server image deployment command script of various system applications: file servers, database servers and web servers. Users must also decide the applicability types of their VMs based on these configurations then the cloud broker creates a corresponding image deployment commands to be executed on a target cloud platform.

III. EVALUATION

We evaluated our ontology by means of storing two types of data sets: platform-specific commands and platform-independent resource specification data. These data setswere used to filled out the properties of the representative class instancescreated based on the ontology.

A. REST_Control class

First, we validated REST_Control class and its subclasses against CloudStack commands but described herein only a representative one, VM creation command (Fig.2) against Deploy_Virtualized subclass, for the sake of conciseness.

Fig. 2. CloudStack's VM creation RESTfulcommand.

```
http://192.0.2.10/client/api?
command=deployVirtualMachine
&serviceOfferingId=1
&diskOfferingId=1
&templateId=2
&zoneId=4
&cpunumber=2
&cpuspeed=1000000
&memory=2000000
&size=100000000
&apiKey=miVr6X7u6bN_sdahOBpjNejPgEsT35eXq-
jB8CG20YI3yaxXcgpyuaIRmFI_EJTVwZ0nUkkJbPmY3y2bciKwFQ
&signature =Lxx1DM40AjcXU%2FcaiK8RAP0O1hU%3D
```

The command is composed of three portions as follows. Base URL is the base URL to the cloudbroker.API Path is the path ("/client/api?") to the API Servlet that processes the incoming requests. Command String is the part of the query string comprises of the command, its parameters, API Key that identifies the account, and signature hash created to authenticate user account executing API command.We found that the command in Fig.2 along with related meta data could be filled in into the instance properties of Deploy_Virtualized subclass of the ontology completely as showed in Fig.3.

Similarly, we verified REST_Control class and its subclasses against OpenStack commands whose representative one is presented in Fig.4 comprising base URL, API path and command string for OpenStack platform. We could fill in this command and its meta data against the ontology's Deploy_Virtualized subclass seamlessly as in Fig.5.

The thorough validations using full data sets, whose parts are described above, proved that REST_Control class is complete for realizing interoperability between CloudStack and OpenStack.

Fig. 3. Deploy_Virtualized class with CloudStack's VM creation command filled out.

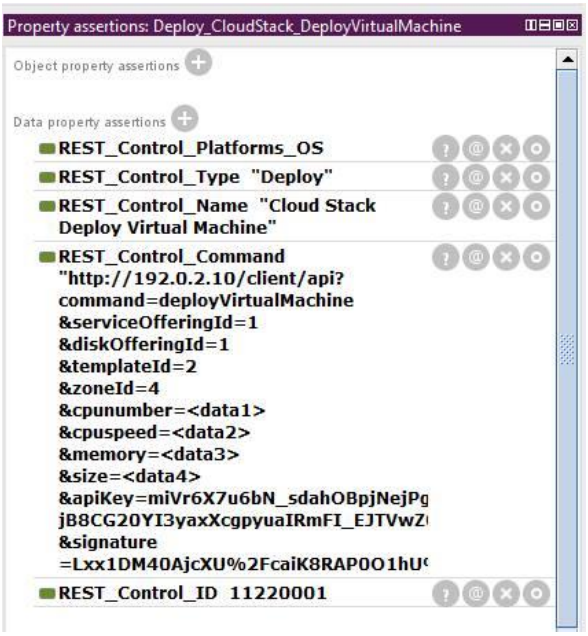
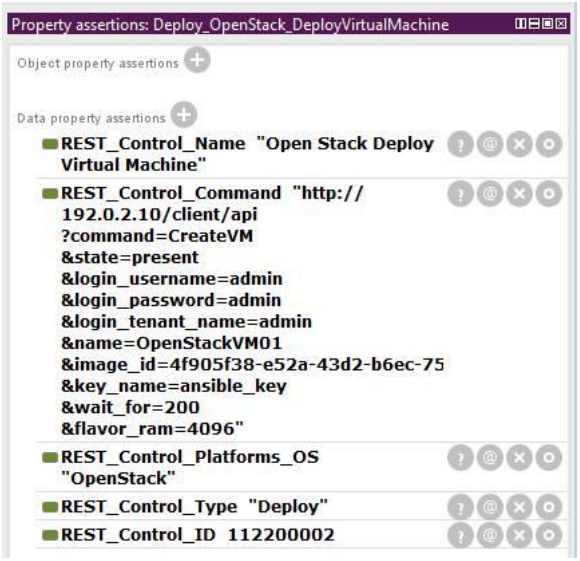


Fig. 4. OpenStack's VM creation RESTful command.

```
http://192.0.2.10/client/api?
command=CreateVM
&state=present
&login_username=admin
&login_password=admin
&login_tenant_name=admin
&name=OpenStackVM01
&image_id=4f905f38-e52a-43d2-b6ec-754a13ffb529
&key_name=ansible_key
```

Fig. 5. Deploy_Virtualized class with OpenStack's VM creation command filled out.



```
&wait_for=200
&flavor_ram=4096
```

Fig. 6. Template_Web_Server class with test data and CloudStack's image loading command filled out.

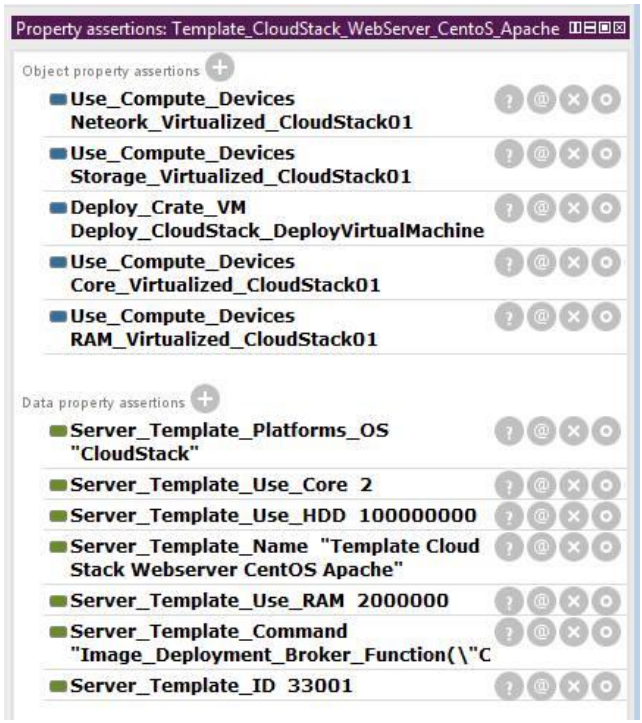
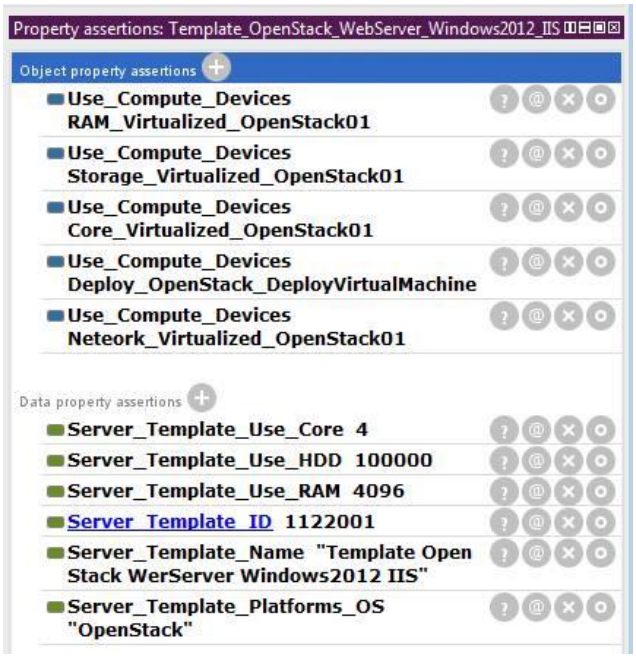


Fig. 7. Template_Web_Server class with test data and OpenStack's image loading command filled out.



B. *Server_Template class*

We validated Server_Template class and its subclasses by using platform-independent resource specification data that is server image preconfiguration data, which is bound to the other class Compute_Devices. For the sake of conciseness, we select to show only subclass Template_Web_Server, representing virtual web server preconfiguration, with filled-up representative data for CloudStack and OpenStack in Fig.6 and Fig.7, respectively.In this way, the Server_Template class of our ontology was verified and validated.

C. *Compute_Devices class*

Finally, we also used platform-independent resource specification data to validate Compute_Devices class and its subclasses. We illustrate here merely subclass Core_Virtualized with filled-in sample computing resource pool characteristics for CloudStack and OpenStack in Fig.8 and Fig.9, respectively.

To recap, our proposed ontology is valid for CloudStack and OpenStack platform commands and platform-independent resource specification datathat altogether aim forthe life-cycle

Fig. 8. Core_Virtualized class with test data for CloudStack.

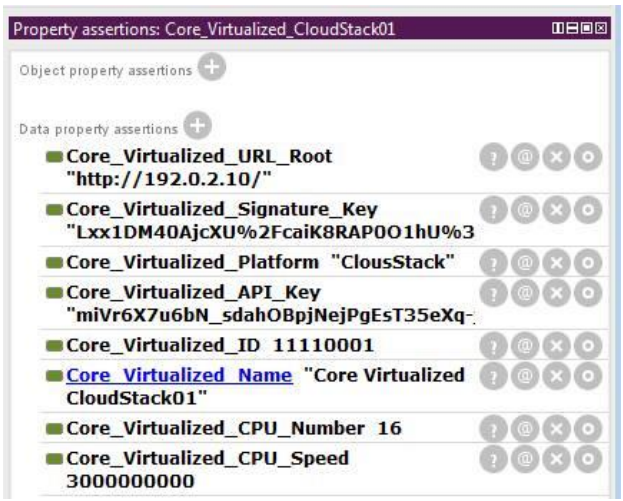
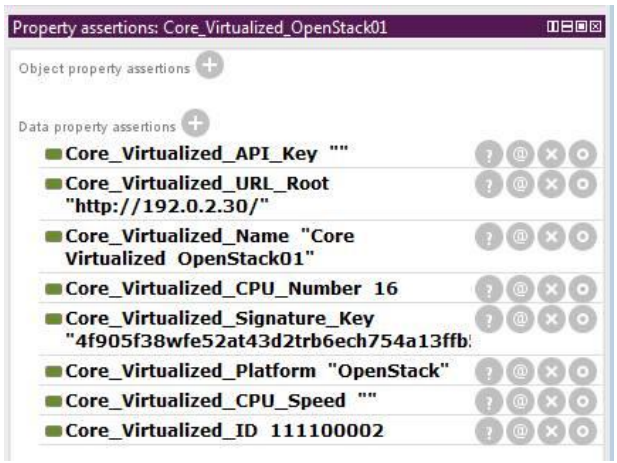


Fig. 9. Core_Virtualized class with test data for OpenStack.



management of the virtual servers of various application kinds. At a further stage, we plan to validate and utilize our ontology in even more practical way by implementing a cloud broker system that is capable of interoperability issue solving.

CONCLUSION

This paper presents a novel ontology for interoperability among CloudStack and OpenStack IaaS platforms. The ontology was designed to support the core requirements for managing virtual server life cycles. We assessed the ontology by means of practical data containment. As our future research, the ontology will be used to implement the knowledge base for the operation of a cloud broker system.

ACKNOWLEDGMENT

We would like to thank Rattanabundit University fora Doctoral scholarship.

REFERENCES

[1] T.S.Somasundaram, K.Govindarajan, MR.Rajagopalanand S.Madhusudhana Rao, "An Architectural Framework to Solve the Interoperability Issue Between Private Clouds Using Semantic Technology," IEEE ICRTIT. India, pp.162-167, April 2012.

[2] N.Loutas, E.Kamateri, F.Bosi and K.Tarabanis, "Cloud computing interoperability: the state of play," IEEE CloudCom. Athens Greece, pp.752-757, 29 Nov - 01 Dec 2011.

[3] OpenStack(2015, August). OpenStack [Online]. Available: https://www.openstack.org

[4] Apache CloudStack (2015, August). CloudStack [Online]. Available: https://cloudstack.apache.org

[5] N.Drummond, S.Jupp, G.Moulton and R.Stevens , "A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools," Edition 1.2, March 13, 2009, pp. 9-33..

[6] B.Rashidi, M.Sharifi and T.Jafari, "A Survey on Interoperability in the Cloud Computing Environments," I.J. Modern Education and Computer Science, pp.17-23, June 2013.

[7] D.Androcec, N.Vrcek and J.Seva,"Cloud Computing Ontologies: A Systematic Review," The 2012International Conference on Models and Ontology-based Design of Protocols, Architectures and Services, pp.9-14, 7 February 2012.

[8] T.Han and K.M.Sim,"An Ontology-enhancedCloud Service Discovery System," International MultiConterence of Engineers and Computer Scientists 2010, 17-19 March 2010.

[9] C.Choi, et.al., "A Design of Onto-ACM(Ontology based Access Control Model) in Cloud Computing Environments," Journal of Internet Services and Information Security, vol. 2, no. 34, pp. 54-64, 2012.

[10] Stanford University (2015, August). protege [Online]. Available: http://protege.stanford.edu/