Developing and Deploying Cloud Services Based on Abstraction Approach

Binh Minh Nguyen, Viet Tran, Ladislav Hluchy
Institute of Informatics
Slovak Academy of Sciences
Dubravska cesta 9
845 07 Bratislava, Slovakia
{minh.ui, viet.ui, hluchy.ui}@savba.sk

JDIM
Journal of Digital Information Management

ABSTRACT: *With the advance of cloud computing, the trend of developing and deploying applications on cloud environment also has appeared recently. There are economic as well as technological reasons why an application should be developed and deployed on the cloud. Reducing costs and business agility are typical business factors for cloud-based applications. Cloud computing can provide significant cost savings because of the increased utilization resulting from the pooling of resources. In addition, this technology enables rapid delivery of IT services, which also improves business efficiency. On the operational side, manageability, performance, and scalability are the typical reasons why application developers consider cloud computing. However, cloud computing is still considerable complex to use. Not only the migration between clouds is difficult - but also the development and deployment of a new, complex service from the beginning could be quite a challenge using today's cloud computing. In this paper, novel high-level abstraction layers for cloud computing is presented. The abstraction layers will allow users to manage cloud resources form various clouds and simplify the process of developing and deploying services into those clouds. This approach will also solve absolutely the interoperability issue, thus improving the flexibility of cloud computing.*

Categories and Subject Descriptors: C.2.4 [**Computer Communication Networks**]: Distributed Systems – Cloud computing, Distributed applications

General Terms: Cloud Computing, Object Abstraction

Keywords: Cloud Computing, Abstraction, Object-Oriented Programming, Cloud-Based Application, Interoperability

Received: 1 March 2012, Revised 14 April 2012, Accepted 19 April 2012

1. Introduction

The term of cloud computing may not be strange to scientific communities as well industry nowadays, as it grows very fast in the last five years with the support of infrastructures over network. Users' computer and companies have gradually changed their habit ways of using computational resources from own computer or server to centralized third party providers. In cloud environment, services are provided with higher availability and often lower cost than the traditional IT operations [1]. Therefore, cloud computing has become the main tendency of high-performance computing (HPC) technology today.

At the moment, it is difficult for users to develop and deploy applications on various clouds at the same time [2-4]. In principle, cloud applications could be built over PaaS (Platform as a Service) or IaaS (Infrastructure as a Service) types of cloud computing. PaaS clouds provide environments for hosting application services and API (Application Programming Interface) for implementing these services. The platforms will manage the execution of these services and offer some advanced features like automatic scaling. However, for existing (legacy) applications, PaaS may require complete rewritten applications using the dedicated API provided by the platforms what is not feasible for cloud developers. Furthermore, each platform can have different key features and API, what make moving applications from a platform to another is practically impossible.

IaaS clouds provide resources (virtual machines, storage) as services where the users can have full access to the resources (often virtualized) and manipulate with them directly. For example, users can log into the virtual machines provided by IaaS providers and directly execute

some commands or modify some files on the virtual machines. For existing applications, this approach requires many efforts for porting the applications to cloud computing.

In brief, the development, deployment and migration of cloud services are limited by several factors, including:

- PaaS are special purposed and limited to concrete platforms.

- IaaS are too low-level service. Users are forced to be administrators of their systems, they have to install and configure everything by themselves.

- The lack of suitable programming model for application development in IaaS.

- The lack of interoperability between clouds with each other.

Therefore, from the view of users, the need of an instrument which enables to simplify development and deployment of cloud services that plays an important role [5]. In this way, it is possible to achieve the highest work performance from diverse clouds. Users can write cloud services, pack them and deliver the code of the services for deployment on various clouds without any obstacles. In addition, users may choose these target cloud infrastructure to deploy the developed services and manage them in a unified user interface. The scientists are not IT experts who can use cloud computing to solve large or complex tasks.

This research primarily focuses on managing IaaS (like Amazon EC2 [6], Eucalyptus [7], OpenNebula [8], Openstack [9] and others) to simplify the creation and use of virtual machines/clusters for developing and deploying services to various clouds at the same time. The abstraction layers could also make interoperability between providers from the view of users and enables opportunities for creating optimization (substituting, brokering) for users/consumers.

2. The Overview of the Abstraction Layers

In our concept, we change the relationships between the cloud providers, users and developers. The developers will have much more active role: they will define the interface, the list of actions what users can do, also the implementation how the actions will be realized in the cloud infrastructures. The users will interact with the cloud resources exclusively via the interface defined by the developers, not via the middleware functionalities. Implementation details of these actions, however, are not visible to users. Figure 1 show the relationships between participants in the cloud computing.

Technically, we implement high-level abstraction layers between developers/users and clouds with operating support for managing resources from the cloud systems. The abstraction layers will hide the implementation details of different cloud infrastructures, remove vendor lock-in. Users will manage the services via methods of the reference objects so developers could control what users can do, and users will know what they should do via the methods without studying implementation details and technological backgrounds.

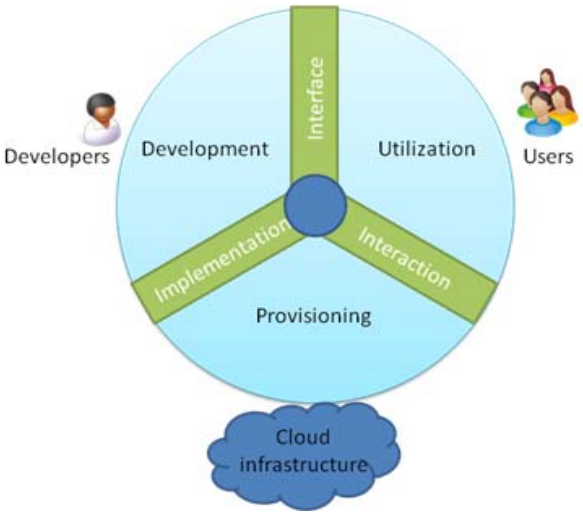

Figure 1. Relationship between parties in cloud computing

The purposes of the abstraction layers are as follows:

- Abstraction between cloud resources and users: resources in the clouds (virtual machines, images, storages) are abstracted as objects and users can manipulate with them via the methods provided by the objects. The abstraction will allow changes in the backend without affecting functionalities and modification of developed applications in the abstraction layers;

- Abstraction of complex systems: via mechanisms like inheritance, composition and polymorphisms, developers can make abstraction of more complex systems with several components easily, and deploy them with a single click;

- Simplification of user interface: Users can manipulate resources as objects without dealing with implementation details;

- Interoperability: Applications and user' scripts developed in the abstraction layers will work for different cloud middleware from different providers;

- Optimization: The abstraction layers will allow optimization mechanisms like brokering, substitutions, load balancing and so on. For example, when the user creates a new virtual machine, the mechanism can choose which provider is best for the current instance.

3. Designing

3.1 Object-oriented Approach

The abstraction layers rely on object-oriented approach to abstract computing resources. The resource is represented as an object where all information related to the resource is encapsulated as data members of the object. In this way, the manipulation with the resource will be done via member methods of the object. For example, assume that a virtual machine in the cloud is

represented by an object *vm*, then starting the machine is done by *vm.start*(), uploading data/application code to the machine is done by *vm.put* (*data*), execution of a program on the machine is done by *vm.execute* (*'command-line'*), and so on. More importantly, developers can concretize and add more details to resource description using derived class and inheritance. They also can define what users can do with cloud application via methods of abstract objects. Within commands, default values will be used whenever possible. Sometime, the users only want to create a virtual machine for running their applications; they do not care about concrete Linux flavor, or which key pair should be used. The interface should not force users to specify every parameter even if the users do not care about it. Abstraction layers also make space for resource optimization. The optimization can decide which options are best for users.

3.2 Abstraction Layers

As mentioned in the overview section, developers will define a clear interface what users can do with the application software and also implementation how the actions will be realized. The approach will require the developers to create implementation of all actions that users can do with their applications. However, due to object-oriented approach, developers can reuse all code from lower software layers (e.g. operating system, platforms) and focus only on actions specific to their applications.

The core of the abstraction layers is a *basic instance layer*. This layer only will be implemented with basic functionalities of virtual machines in order to create cloud applications. The number of variations of these instances are limited (some major image flavors), therefore, making these functionalities operate correctly on a given cloud infrastructure can be solved with reasonable efforts. As long as the infrastructures are supported by the instance layer, application developers do not have to care which infrastructure the users will choose for running their applications. For instance, developers can implement, test and validate their applications on a private cloud with Opennebula or Openstack while the users can use their abstracted objects to deploy the application to Amazon EC2 cloud. The architecture of the basic abstraction layer is depicted in Figure 2. It contains three components: interface, drivers and image repository. In which, interface provides interaction between users and basic abstraction layer, thereby, it is visible with users. Conversely, drivers and image repository are designed to hide with users.

The interface of basic instance layer will be designed with the following functionalities:

- *Provisioning*: these functionalities are provided by the instance layer for provisioning, involving start, stop of virtual machines.

- *Monitoring*: getting actual information of the virtual machines (CPU loads, memory use, IP address and so on).

- *Execution*: run commands or install new packages to the virtual machines.

- *Transfer*: upload/download data from virtual machines.

- *Backup/Restoration*: backup/restore machine configurati -on and user data.

From the view of users, they only use and inherit the functionalities that are provided by the interface such as provisioning, execution, transfer and so forth to build their applications on clouds. Functionally, provisioning allows users to create (or terminate) virtual machines. Afterwards, they will use the monitoring function to find information about the created machines. The most necessary information is IP addresses that provide end-point connection with the machines for users. Execution and transfer offer the mechanisms to execute, install, upload and download database or application softwares to these instances. To avoid time-consuming in re-developing created applications, the backup/restoration functionalities enables users to create/restore a backup of systems or appliances. The backup function will make a backup copy of a system into an image. Such backup data can be later restored on a newly created machine on the suitable target infrastructure. So, developers can re-build their applications in the fastest way. It is similar to most cloud middlewares, which have supported backup or creating a snapshot in order to store user data and program-specific configuration on virtual machines. As a result, users save the time in re-installing applications as well as re-creating system. The inheritance feature of object-oriented programming that enables users to reuse all these functionalities in the upper layers.

Each driver is a module, which allows the basic instance layer to manage a cloud via its API. For example, if the layer supports Amazon EC2 and Openstack cloud, it means that basic abstraction layer will have two drivers, in proportion to number of clouds. However in reality, there are several APIs, which have considered as de-facto standards, typically OCCI [10]. If a driver supports OCCI, as the result, the instance layer thus will support clouds that use OCCI as their API. Otherwise, when the basic instance layer needs to expand for other clouds, the only thing programmer has to do is create the new driver without changing the original code of instance layer, including its interface and drivers of clouds that have already supported. The relationship between drivers and interface of basic instance layer is similar to the relationship of hardware devices (e.g. graphic card) and an application or operation system in traditional computer, where the driver acts as a translator between the device and the application. Indeed, the drivers of the abstraction layer will act as an intermediary to translate the users' command (via abstraction interface) to cloud. Due to the incompatibility of APIs, every cloud or every type of API must have a driver for its own.

A cloud API provides users with many different functions, for example, creation, termination of virtual machine, creation of storage, monitoring and so on. Nevertheless, according to the design idea, necessary API functions for the basic instance layer only include: *provisioning* (*start,*

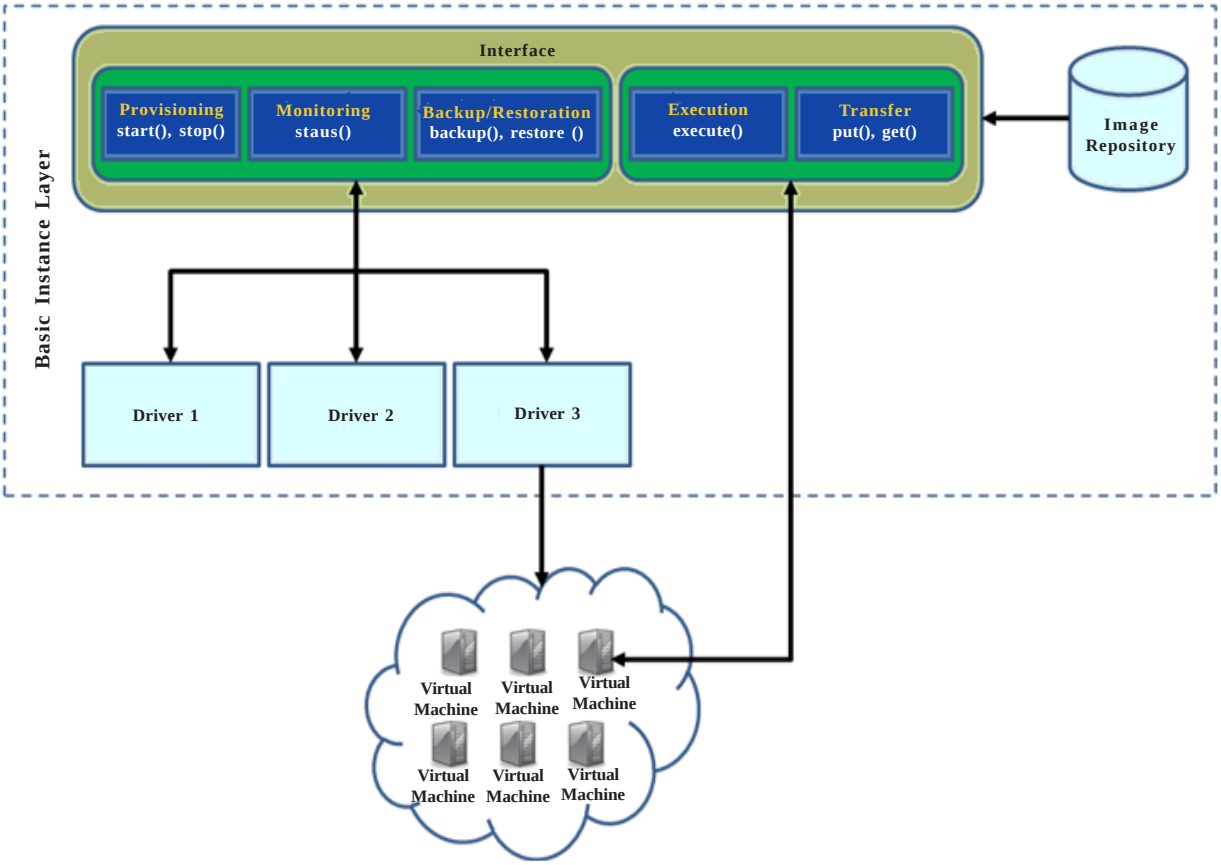

Figure 2. Architecture of basic instance layer

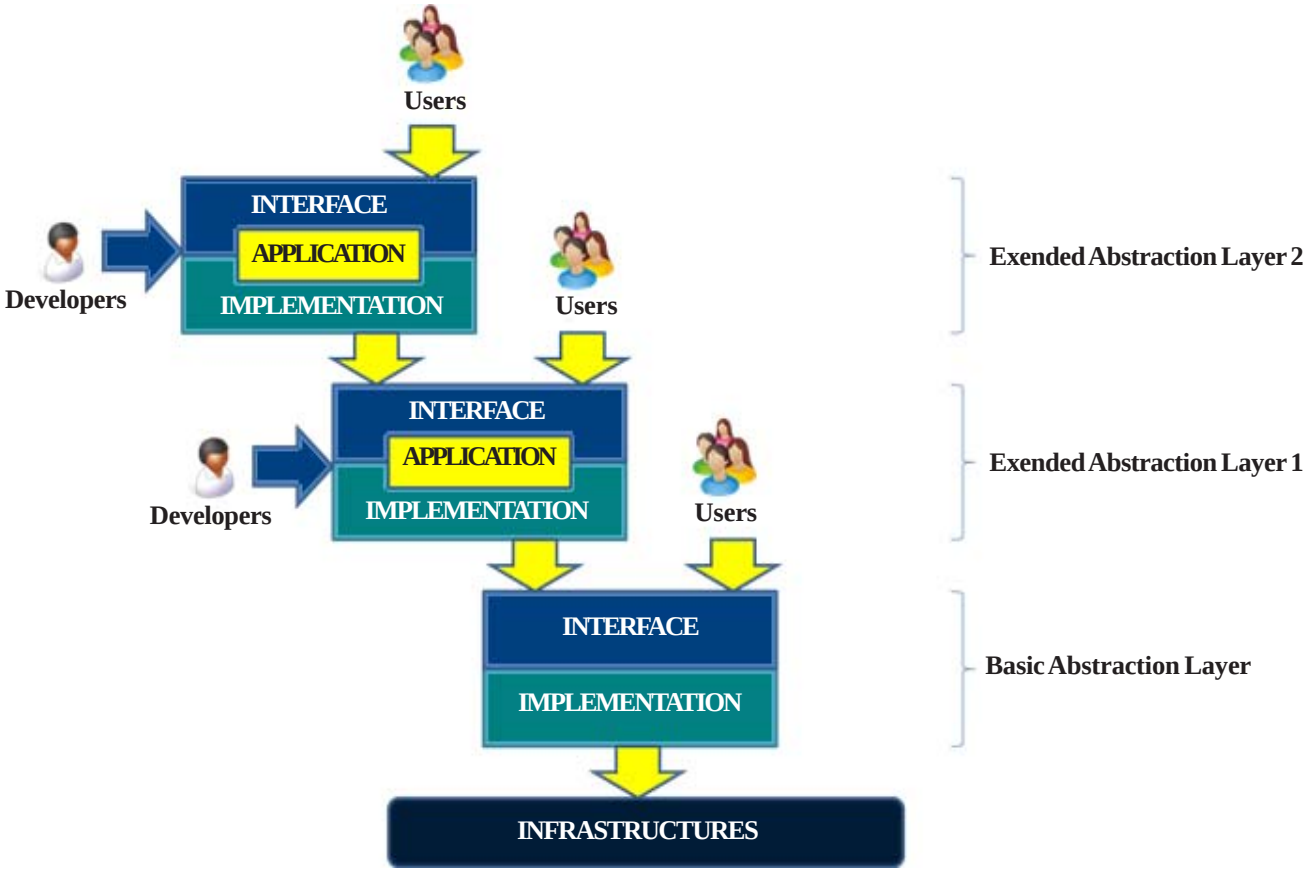

Figure 3. Inheritance of abstraction layers

stop), *monitoring and backup/restoration of virtual machine*. Most IaaS clouds offer these features. Thus, the layer through drivers does not need to support all API functions that are provided by vendors. A driver will only use middleware-specific API to interact with cloud in order to realize only these operations above. Relying on that, the huge number of unnecessary information about cloud is eliminated.

In fact, to *execute* commands, install software applications, *upload* or *download* data from local to virtual machine, users must directly connect to the virtual machine and manipulate these operations. No API provides functions to carry out those works. Basically, despite virtual machines belong to various clouds, but they have typically a public IP address. Users connect to the virtual machines via the IPs under authentication (keypair). Consequently, each driver does not need to contain programming code of these functionalities inside it. The abstraction layer will hide implementation details of connecting and performing the tasks by its interface. Thereby, the execution and transfer functionality of abstraction layer will directly interact with the virtual machines, and they do not depend on cloud middlewares as well as APIs. The implementation of these functionalities can be used for all the virtual machines from different clouds.

Besides its methods, the basic instance layer is linked to a repository containing virtual machine images of IaaS clouds which basic instance layer supports. In this way, assume that a user requires a virtual machine running on Amazon EC2 resource. The basic instance layer will use an available Amazon Machine Image (AMI) in the repository to create virtual machine at users' request. Some other time, user wants to create a virtual machine that runs on OpenNebula resource, the layer thus will use an available OpenNebula image in the repository. The number of IaaS clouds which the basic instance layer manages to be given, so the number of available images will be given. An image repository for the basic instance layer will bring many advantages, including respect for all of the features, functions and technologies from vendors, who are not required changing themselves for supporting the basic instance layer and conversely, the layer does not need any common standard among cloud systems. From the perspective of users, they can manipulate multiple clouds at the same time under a unified interface irrespective of how each cloud works. Through basic instance layer, the users also can easily upload, execute/ install and download application packages on virtual machines from diverse clouds.

3.3 Inheritance of Abstraction Layers

Based on object-oriented approach, the basic instance layer allows inheritance in order to help developers create new abstraction layers upon it. The inheritance of abstraction layers is described in Figure 3. Thus, the developer can define a software layer 1 with new methods for on demand from users. In other words, the software layer 1 will hide implementation details of the basic instance layer in its methods. Similarly, developers can also define methods for software layer 2 over software layer 1 according to user' needs without having to worry about how the preceding layer's methods operate.

A simple example, a user needs to store a database in cloud computing. Then, developer will use the basic instance layer to easily create a virtual machine and install MySQL package into this machine on any cloud. On this platform, the developers will define methods to interact with MySQL for users as *upload_database*(), *connect_database*(), *search_database*(), *import_item*() and so on which are call back basic instance layer's methods (e.g. *get*(), *execute*()) . In the principle, users do not have to care how their commands interact with basic instance layer. They only use the method that developer defined to use the database service. Moreover, after storing a number of different databases, the user wants to build wiki pages for easy database lookup. Developers can easily define a series of new methods for software layer 2 to create wiki pages with the connected available databases. For instance, *config_wiki*() method is defined by developer for software layer 2. It will automatically call back the *connect_database*() method of storage service (software layer 1) in order to link a wiki page with an available database. At the time, user only uses *config_wiki*() without knowing how the database is linked to his pages.

The compound object could be used for abstraction of systems with more than one virtual machine (e.g. cluster, PBScluster). Then, all virtual machines of the cluster will be placed in the same cloud, ensuring the network connection between virtual machines. More importantly, developers can deploy the whole system with a single command and customize monitoring service with user-defined actions on events.

4. Example

4.1 Abstraction of a Virtual Machine

We started with a simple example how to create a virtual machine in the cloud and execute an application on the newly created machine. The following methods which are provided within basic instance layer:

```
Instance t = new Instance ();
t.start();
t.put (" input.dat, myapp.exe ");
t.excute (" myapp.exe input.dat ouput.dat ");
t.get ("output.dat ");
t.shutdown ();
t.delete ();
```

As Java language, users will execute the commands one by one in interactive mode of Java shell. The command in the first line will create an instance (a virtual machine) with default parameters (defined in configuration files or in the *defaultInstance* variable). The users can customize the instance by adding parameters e.g. *t = Instance*

(*type = large*) or even more complex *t = Instance* (*type = medium, image = myImage, keypair = myKeypair*). If the users want to create more instances with the similarity parameters, they can set common parameters to *defaultInstance*. Note that the instance is created without starting, so users can change parameters, e.g. *t.set Keypair* (*public, private*).

The next commands in the example will start the virtual machine, upload the application and its data, execute the application on the virtual machine, download output data and terminate the machines. Users can get information about the virtual machines simply by command print t. The information given by the command is similar to the *xxx-describe-instance* in Amazon EC2 or Eucalyptus.

As it is shown in the example above, users do not have to deal with detailed information like IP address, SSH commands connection to the virtual machines and so on. They simply upload data, run application or download data with simple, intuitive command like *t.put* (), *t.execute* (), *t.get* () and so on. Of course, if the users really need to run its own SSH connection, they can do it with information (IP address, SSH key) from the print t command.

Now we can go further in abstraction by creating a function *execute* () from the commands. From this point, users can execute an application in the cloud only with a single command *execute* () with input/output data and command line as parameters.

Note that the abstraction (like Instance class or *execute* () command) does not only simplify the process of using cloud computing, but also allows experts (e.g. IT support staff of institutes/companies) to do optimization for users. The actual users of cloud computing do not have to be IT professionals but may be scientists, researchers, experts from other branches. For example, the IT staff can customize the virtual machines' creation by checking if there is free capacity in the private clouds first before going to public clouds.

4.2 Abstraction of MySQL Server

We continue with an example how to develop a database service within software layer 1. The service will inherit all functionalities from basic instance layer. In this way, developers can design quickly service commands for users what they are needed. The initialization of MySQL server which was configured by developers as follows:

```
public class MySQLServer {
    public void init (){
     Instance service = new Instance ();
     service.execute ("apt-get install –y mysql " );
                  //install mysql-server package
   }

   public void config () { //define method for users
   ...
   }
     public void upload_database (Data dat) {
       //define method
       service.put (dat);
      service.execute (" ");
   }

   public void import_item(Item ite) {
     //define method
     service.execute (" ");
   }
   ...
}
```

After defining abstract objects for database service, users then can use simply the service:

```
MySQLServer m = new MySQLServer ();
m.config ();
m.upload_database ("dat " );
m.import_item (" ite ");
…
```

The developers also can go further in the use of abstraction layers to deploy appliances that are linked to the available database (e.g. web services, wiki pages, and forum). Meanwhile, users can use these appliances in simple and intuitive way.

4.3 Abstraction of Complex Systems

Besides simplifying the process of developing, deploying and using services, through abstraction layers, developers also are welcome to design complex systems such as cluster, HadoopCluster, PBSCluster etc. In this subsection, we will demonstrate how they create a cluster with compound object. In this example, N is the number of virtual machines running in parallel on the cluster. The commands look as follows:

```
public class Cluster {
  public void init () {
    for (int i = 1; i <= N; i++) {
       Instance head = new Instance ();
       }
  }

  public void start () {
    // define method
    for (int i = 1; i <= N; i++) {
       head.start ();
    }
  }
  public void config (){
```

```
    // define method
    …
  }

  public void upload_cluster (Data dat){
    // define method
   for (int i = 1; i <= N; i++) {
      head.put (dat);
    }
  }
  …
}
```

Then users can easily create and use the cluster as follows:

```
Cluster c = new Cluster ();
c.start ();
c.config ();
c.upload_cluster ("dat ");
…
```

Of course, developers can define derived classes from Cluster, e.g. PBSCluster, HadoopCluster, by modifying the configuration method in the Cluster class *config*(). More complex, they also can create an abstract object ElasticPBSCluster, a cluster with PBS where worker nodes are dynamically added and removed according to the actual loads of the PBS.

5. Related Work versus Our Contribution

Although some enterprises have provided standardizations and aim towards a scenario in which clouds could be interacted with each other without any trouble. However, each provider has built the standardizations which based on the features of their services. A typical example is Microsoft Azure [11-12] that provides an open, standards-based, and interoperable environment with support for multiple Internet protocols, including HTTP, REST, SOAP, and XML, etc. Thus, if other providers want to apply the available Azure's standards, they must change their clouds in order to suit them. Therefore, in this way, most of the standardizations were defined that are not widely accepted.

Besides, Open Grid Forum (OGF) [13] also has the ambition to solve the problem of interoperability for IaaS. OGF defined OCCI (Open Cloud Computing Interface) to provide an open standard API for existing IaaS cloud using RESTful (Representational State Transfer) protocol. Moreover, the main goal of OCCI is the creation of hybrid cloud operating environments independent from vendor and middlewares [14]. OCCI's documents separate OCCI core from OCCI interface, which has published. OCCI core model specifies base types, including: Entity, Resource, Link and Action. The Entity is an abstract type of the Resource and Link type; Resource describes concrete resources as the objects; while Link defines the relationship between Resources, the Action defines operation applicable to Entities. The OCCI model is developed in UML (Unified Modeling Language), but the types of the model are described by a graph structure which is similar to an OWL (Web Ontology Language) [15] ontology definition. This work has not been finished, it still under development in a preliminary state.

Distributed Management Task Force (DMTF) [16] has published Open Virtualization Format (OVF) [17]. The OVF is a virtual machine standard that provides a flexible, secure, portable and efficient way to distribute virtual machines between different clouds. Called virtual appliances [18], users can package a virtual machine in OVF and distribute it on a hypervisor. The OVF file is an XML file that describes a virtual machine and its configuration. Application-specific configuration can be packaged and optimized for cloud deployment, as multiple virtual machines, packaged and maintained as a single entity in OVF format.

Institute of Electrical and Electronics Engineers (IEEE) has two working groups (P2301 [19] and P2302 [20]) that have researched on standardization of cloud computing aspects. P2301 will serve as metastandard for cloud computing in critical areas such as applications, portability, management, interoperability interfaces, file formats and operation conventions. As the results, P2301 will provide a roadmap for all cloud providers building services under the standard. By the way, the project will enable interoperability, portability between clouds. Meanwhile, P2302 defines topology, protocols, functionalities and governance required for cloud interoperability and cloud federation. When completed, P2302 will ensure the ability of exchange data between clouds. However, cloud providers still may build their systems with distinct features to enable commercial competition.

Technically, the projects above force cloud providers to accept and support their products. Otherwise, they just have stopped at creating a common standard in order to solve interoperability issue. Some projects have not yet completed. Others have provided a product, but even then it is not considered comprehensive solution to the cloud problems (identified in Section 1). In comparison with the efforts, our approach has some advantages, including:

1. Providing the programming model for easy and controlled development and deployment of services over IaaS.

2. Providing a general-purpose solution for the application migration issue.

3. The abstraction layers are independently developed, not require support from underlying infrastructures.

4. Simplification of using cloud computing under the unified interface, lower barriers for new users.

5. Solving the interoperability among clouds, removing vendor lock-in.

6. Enabling the ability of automatic optimization in the background.

6. Conclusion and Future Work

In this paper, we have presented the novel high-level abstraction layers for development and deployment of cloud services. In our approach, we separate cloud developers and users with specific roles. Both only use functionalities of the abstraction layers without care about middleware implementation details. The core of the abstraction layers is the basic instance layer which provides implementation of basic instance for each known cloud middleware. Through the basic layer, the manipulation with the resource will be done via member methods of the object. The layer also allows developers to create cloud appliances easily via inheritance mechanisms whereas users can use simply these appliances. In addition, the abstraction layers also ensure interoperability between different cloud infrastructures which is one of the invaluable characteristic in cloud environment.

For the future, we plan to develop another abstraction layer of data management in clouds. The data abstraction layer will provide fully data storage and access in a secure way through different cloud providers. Like the abstraction layers of cloud resources and appliances, data abstraction layer also will be used to manipulate the data in order to decrease the access time and to increase flexibility with data. Data's mechanisms such as migration, replication and stripping will be exploited by the layer.

7. Acknowledgments

This work is supported by projects SMART ITMS: 2624012005, SMART II ITMS: 26240120029, VEGA No. 2/0211/09, VEGA 2/0184/10.

References

[1] Buyya, R., Ranjan R., Calheiros, R. N. (2010): InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services, *Algorithms and Architectures for Parallel Processing, LNCS* vol. 6081/2010. p. 13-31.

[2] Ramakrishnan, L., Jackson, K. R., Canon, S., Cholia, S., Shalf, J. (2010). Defining Future Platform Requirements for e-Science Clouds. *ACM Proceedings of the 1st ACM symposium on Cloud computing*. p. 101-106.

[3] Goscinski, A., Brock, M. (2010). Toward dynamic and attribute based publication, discovery and selection for cloud computing. *Elsevier, Zv. Future Generation Computer Systems*, p. 947-970.

[4] Kothari, C., Arumugam, A. K. (2010). Cloud Application Migration. http://soa.sys-con.com/node/1458739.

[5] Golden, B. The Case Against Cloud Computing. http://www.cio.com/article/477473/.

[6] Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/.

[7] Eucalyptus community. http://open.eucalyptus.com/.

[8] OpenNebula. The Open Source Toolkit for Cloud Computing. http://opennebula.org/.

[9] Openstack, The open source software for building private and public cloud. http://openstack.org/.

[10] Metsch, T., Edmonds, A., Nyrén, R. (2011). Open Cloud Computing Interface – Core. *In: Open Grid Forum, OCCI-WG, Specification Document.* http://forge.gridforum.org/sf/go/doc16161/.

[11] Microsoft Windows Azure. http://www.microsoft.com/windowsazure/windowsazure/.

[12] Hay, C., Prince, B. H. (2010): Azure in Action. *Publisher: Manning Publications*.

[13] Open Grid Forum. http://www.gridforum.org/.

[14] Moscato, F., Aversa, R., Di Martino, B., Fortis, T., Munteanu, V. (2010). An Analysis of mOSAIC ontology for Cloud Resources annotation. *In: Proceedings of the Federated Conference on Computer Science and Information Systems*. IEEE Xplore. p. 973–980.

[15] Baader, F., Horrocks, I., Sattler, U. (2005). Description Logics as Ontology Languages for the Semantic Web. *In*: Hutter, Werner; Stephan. *Mechanizing Mathematical Reasoning: Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday. Heidelberg: Springer Berlin*. ISBN 978-3-540-25051-7.

[16] Distributed Management Task Force Inc. http://www.dmtf.org/.

[17] Open Virtualization Format. http://dmtf.org/sites/default/files/OVF%20Overview%20Document_2010.pdf.

[18] Virtual Appliances. http://www.vmware.com/appliances/getting-started/learn/.

[19] IEEE P2301 working group. http://grouper.ieee.org/groups/2301/.

[20] IEEE P2302 working group. http://grouper.ieee.org/groups/2302/.

Author Biography

Binh Minh Nguyen received his Dipl.-Ing. degree in informatics from Tambov State Technical University (Russia) in 2008. Currently, he is working as a project assistant at Department of Parallel and Distributed Computing, Institute of Informatics, Slovak Academy of Sciences (Slovakia). He is also a PhD student at Faculty of Informatics and information Technology, Slovak University of Technology in Bratislava. His research interests include cloud computing, distributed systems, service-oriented architecture and data integration.