

Architecture-based Integrated Management of Cloud Resources

Xing Chen^{1,2}, Ying Zhang^{3,4}, Xiaodong Zhang^{3,4}, Xianghan Zheng^{1,2}*, Wenzhong Guo^{1,2}, Guolong Chen^{1,2}

¹ College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350108, China

² Fujian Provincial Key Laboratory of Networking Computing and Intelligent Information Processing

³ Key Laboratory of High Confidence Software Technologies (Ministry of Education)

⁴ School of Electronics Engineering and Computer Science, Peking University, Beijing, 100871, China

{chenxing08, zhangying06, zhangxd10}@sei.pku.edu.cn; {xianghan.zheng, guowenzhong, cgl}@fzu.edu.cn

Abstract—Cloud management faces with great challenges, due to the diversity of Cloud resources and ever-changing management requirements. For constructing a management system to satisfy a specific management requirement, a redevelopment solution based on existing management system is usually more practicable than developing the system from scratch. However, the difficulty and workload of redevelopment are also very high. As the architecture-based runtime model is causally connected with the corresponding running system automatically, constructing an integrated Cloud management system based on the architecture-based runtime models of Cloud resources can benefit from the model-specific natures, and thus reduce the development workload. Therefore, in this paper, we present an architecture-based approach to managing diverse Cloud resources. First of all, the manageability (such as APIs, configuration files and scripts) of Cloud resources are abstracted as runtime models, which can automatically and immediately propagate any observable runtime changes of the target resources to the corresponding architecture models, and vice versa. Then, a customized model is constructed according to the specific Cloud management requirement. Finally, the operations on the customized model are mapped to the ones on Cloud resource runtime models through model transformation. Thus, all the management tasks can be carried out through executing operations on the customized model. The experiment on a real-world cloud demonstrates the feasibility, effectiveness and benefits of the new approach to integrated management of Cloud resources.

Keywords - Cloud Management; Software Architecture; Models at Runtime.

I. INTRODUCTION

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction [1]. And they always allocate virtual machine (VM)-based computing resources on demand through the virtualization technology, and deploy different kinds of fundamental software onto virtual machines, which are finally provided in a service-oriented style. Nowadays, more and more software applications are built or migrated to run in a cloud, with the goal of reducing IT costs and complexities. This trend brings unprecedented challenges to system management of Cloud, which mainly comes from the following two aspects:

On one hand, the virtualization not only makes the physical resources easier to share and control but also increases the complexity of management. For instance, there are different kinds of Cloud resources, which include CPU, memory, storage, network, virtual machines and different types of software, such as web servers and application servers.

On the other hand, there are various management requirements which require using different Cloud resources with different management styles. For instance, a 3-tier JEE (Java Enterprise Edition) application typically has to use the web server, EJB server and DB server. These servers have different management mechanisms. An EJB server should comply with JMX management specification and rely on the JMX API, while a DB server is usually managed through the SQL-like scripts. In addition, the EJB server can usually sustain the running of several applications simultaneously. What's more, all of the platforms are in a resource sharing and competing environment. Administrators have to carefully coordinate each part to make the whole system work correctly and effectively.

Actually, Cloud management is the execution of a group of management tasks, from the view of system implementation. A management task is a group of management operations on one or more kinds of Cloud resources. A management operation is an invocation of a management interface provided by Cloud resources themselves or a third-party management service. Due to the specificity and large scale of Cloud, the management tasks of different Clouds are not the same. For instance, Amazon EC2 mainly manages infrastructure level Cloud resources such as virtual machines, while Google App Engine manages platform level Cloud resources such as operating systems execution environment. There are many Cloud management systems, which implement a group of management tasks for a given set of Cloud resources based on their management interfaces. Then Cloud administrators can conduct redevelopment based on existing management systems to satisfy specific requirements. However, the redevelopment is usually implemented in general purpose programming languages like Java and C/C++, which can bring enough power and flexibility but also cause high programming effort and cost. For instance, the existing VM and middleware platforms have already provided adequate proprietary APIs (e.g., JMX) to be used by monitoring and

executing related codes. Administrators first have to be familiar with these APIs and then build programs upon them. Such a work is not easy due to the heterogeneity of platforms and the huge numbers of APIs provided. In a management program, proper APIs have to be chosen for use and different types of APIs (e.g., JMX and scripts) have to be made interoperable with each other. Such “boring” work is not the core of the management logics comprised with the analyzing and planning related codes, but it has to be done to make the whole program run effectively. During this procedure, the irrelevant APIs as well as the collected low-level data can sometimes make administrators exhausted and frustrated. Furthermore, as the programs are built on the codes that directly connect with the runtime systems, they are not easy for reuse. Administrators have to write many different programs to manage different cloud applications and their platforms even the management mechanisms are the same.

The fundamental challenge faced by the development of management tasks is the conceptual gap between the problem and the implementation domains. To bridge the gap, using approaches that require extensive handcraft implementations such as hard-coding in general purpose programming languages like Java will give rise to the programming complexity. Software architecture acts as a bridge between requirements and implementations [2]. It describes the gross structure of a software system with a collection of managed elements and it has been used to reduce the complexity and cost mainly resulted from the difficulties faced by understanding the large-scale and complex software system [3]. It is a natural idea to understand management tasks through modeling the architecture of the system. And current research in the area of model driven engineering (MDE) also supports systematic transformation of problem-level abstractions to software implementations [4]. The complexity of bridging the gap could be tackled through developing control systems based on the models that describe the architecture of the cloud and through the automated support for transforming the changes on the architectural models to the ones on the running systems and vice versa.

This paper presents an architecture-based approach for integrated management of diverse Cloud resources. We construct the runtime model of each kind of Cloud resource (Cloud resource runtime model) automatically based on its architecture meta-model and management interfaces, and define the customized model which satisfies the specific Cloud management requirement. Then we map operations on the customized model to ones on Cloud resource runtime models through a rule-based resolution engine. Finally, all the management tasks can be carried out through executing operations on the customized model. The whole approach only needs to define a group of meta-models and mapping rules, thus greatly reduces the workload of hand coding. As an additional contribution, we develop an architecture-based tool for the unified management of the hardware and

software resources of virtual machines in our approach, based on OpenStack and Hyperic.

The rest of this paper is organized as follows: Section II gives an overview of the architecture-based approach to managing diverse Cloud resources. Section III presents the construction of Cloud resource runtime models and the customized model. Section IV describes Model Operation Transformation from the customized model to Cloud resource runtime models. Section V illustrates a real case study and reports the evaluation. Section VI discusses the related works. Section VII concludes this paper and indicates our future work.

II. A BRIEF OVERVIEW OF THE APPROACH

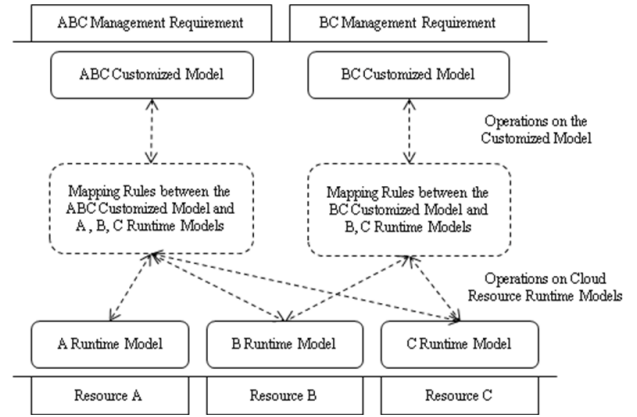


Fig. 1. Overview of the architecture-based approach to managing diverse Cloud resources

Fig. 1 is an overview of the architecture-based approach to managing diverse Cloud resources. The approach introduces the runtime model that describes the architecture of the cloud, for the management of Cloud resources, and mainly includes two parts: 1) construction of Cloud resource runtime models and the customized model; 2) Model Operation Transformation from the customized model to Cloud resource runtime models.

First, we present an approach to constructing the runtime model of Cloud resources and the customized model. There are many different kinds of resources in Cloud. For example, there are virtual machine platforms such as Xen, VMware and KVM, operating systems such as Windows and Linux, application servers such as JOnAS, JBoss and WebLogic, web servers such as Apache, IIS and Nginx, database servers such as MySQL, SQL server and Oracle. The Cloud resource runtime model in our approach is abstracted from the organization architecture of this kind of resources. Thus we can manage the resources through operations on the runtime models, and these operations will finally propagate to the underlying cloud resources. Based on the underlying Cloud resources, there may be various management requirements which consist of specific Cloud resources and management styles. We also construct the customized model according to the specific management requirement. The customized model is not directly connected with the corresponding

Cloud resources, but it is causally connected with Cloud resource runtime models through model transformation.

Second, the customized model and Cloud resource runtime models are used to manage the same underlying resources. But they are different in organization architectures and management styles. Thus, when we manage the resources through operations on the customized model, they need to be mapped to ones on Cloud resources runtime models. We present a model transformation mechanism which transforms the customized model to Cloud resource runtime models. Administrators just need to write some mapping rules according to the specific management requirements; the model transformation is automatically completed and the operations on Cloud resource runtime models are automatically generated. Then the operations will be transferred to Cloud resource runtime models and be executed.

After the two steps above, Cloud management tasks can be carried out through executing different kinds of model operating programs, while without considering the management interfaces of underlying Cloud resources.

III. CONSTRUCTION OF ARCHITECTURE-BASED MODELS

As we introduced in section 2, the Cloud resource runtime model in our approach is abstracted from the software architecture of the running system. The input of the runtime model construction includes a meta-model for resource management specifying what kinds of elements can be managed and an access model of the configurations specifying how to use the management APIs to monitor and modify those manageable elements, i.e. the underlying cloud resources. Then the runtime model is automatically constructed by the code generation tool named SM@RT, which is proposed in [5], and the correct synchronization between the runtime model and the running system is also ensured by SM@RT. (The source code of SM@RT can be downloaded from [6]) Thus we can manage the resources through operations on the runtime model, and these operations will finally propagate to the underlying cloud resources. For instance, in Fig. 2, the synchronization engine builds a model element in the runtime model for the running JOnAS platform. When the model element of JOnAS is deleted, the synchronization engine is able to detect this change, identify which platform this removed element stands

for and finally invoke the script to shut down the JOnAS platform. Due to page limitation, the details of the runtime model construction with SM@RT can be found in [7].

In Cloud environment, there are different management requirements which consist of objective managed resources and appropriate management styles. In Fig. 1, the customized model reflects the objective managed resources and the appropriate management style. The input of the customized model is a meta-model for resource management specifying what kinds of elements need to be managed. The basic types of model operations include “Get”, “Set”, “List”, “Add” and “Remove”. Then the customized model is restricted by the meta-model and any operation is aimed to monitor some parameter of the underlying systems or execute some management operation.

IV. MODEL OPERATION TRANSFORMATION

In our approach, we ensure the synchronization between the customized model and Cloud resource runtime models through model operation transformation which is based on a set of mapping rules between the two models. The mapping rules describe the mapping relationship between the elements of two models. Each attribute of the element in the customized model is related with one of the element in the composite model. Then all changes on the customized model will be transformed to operations on the composite model. We have proposed an approach to model transformation, which can generate transformation codes automatically, based on the customized model, Cloud resource runtime models and the mapping rules.

A. Basic Mapping Rules between Model Elements

There are three types of basic mapping rules between model elements, as shown in Fig. 3.

One-to-one mapping rule: One element in the objective model is related to a certain element in the source model. Particularly, the attributes of the elements in the objective model are also corresponding to the ones of related elements in the source model. For instance, the *Flavor* elements in the objective model and the *MachineType* elements in the source model both reflect the configurations of virtual machines. And the *id*, *name*, *ram*, *disk* and *vcpus* attributes of the *Flavor* elements in the objective model are corresponding to the *id*, *name*, *memoryMb*, *imageSpaceGb* and *guestCpus*

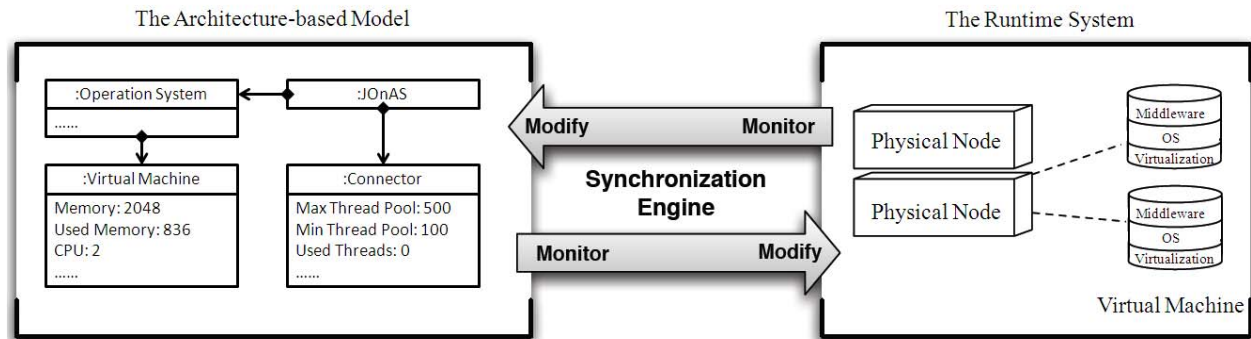


Fig. 2. Synchronization between the runtime model and the running system

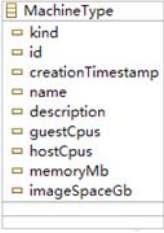
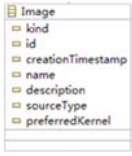
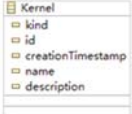

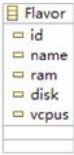
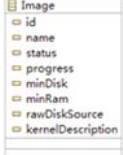
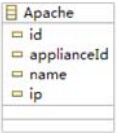
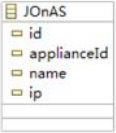
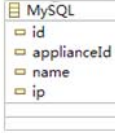
	One-to-One Mapping Rule	Many-to-One Mapping Rule	One-to-Many Mapping Rule
Classes in Source Model		 	
Classes in Objective Model			  

Fig.3. Three types of basic mapping relationships between model elements

attributes of the *MachineType* elements in the source model.

Many-to-one mapping rule: One type of element in the objective model is related to two or more types of elements in the source model. Particularly, the attributes of a certain type of the element in the objective model are corresponding to the attributes of two or more types of elements in the source model. For instance, the *Image* element in the source model and the *Image* element in the objective model both reflect the types of virtual machines. But there is no attribute of the *Image* element in the source model which is corresponding to the *kernelDescription* attribute of the *Image* element in the objective model. And the related one is in the *Kernel* element in the source model, whose *id* attribute equals to the *preferredKernel* attribute of the *Image* element in the objective model.

One-to-many mapping rule: One type of elements in the source model is related to two or more types of elements in the objective model. For instance, the *Server* elements in the source model represent virtual machines, and the *Apache*, *JOnAS* and *MySQL* elements represent virtual machines with software deployed. During model transformation, any *Server* element is mapped to one of the *Apache*, *JOnAS* and *MySQL* elements, according to its *imageId* attribute.

Any mapping rule can be demonstrated as the combination of the three basic types of mapping rules. We have presented the method to describe the mapping rules. The mapping rules are described in XML files and the keywords are defined as follows:

1. **Helper:** It is used to describe the mapping relationship between elements. There are usually three attributes in the “helper” tag, the *key* attribute, the *value* attribute and the *type* attribute. The *value* attribute describes the element in the source model and the *key* attribute describes the element in the objective model. The *type* attribute describes the type of the mapping rule. When

its value is “basic”, it is the one-to-one mapping rule or the many-to-one mapping rule. When its value is “multi”, it is the one-to-many mapping rule. The “helper” tag is used to describe the mapping relationship between elements and elements often have attributes or other elements. So the “helper” tag usually nests “helper” tags, “mapper” tags and “query” tags. Additionally, the “mapper” and “query” tags are used to describe the mapping relationship between attributes.

2. **Mapper:** It is used to describe the mapping relationship between attributes. There are usually two attributes in the “mapper” tag, the *key* attribute and the *value* attribute. The *value* attribute describes the attribute in the source model and the *key* attribute describes the attribute in the objective model. Specially, the elements, which the attributes belong to, are defined in the “helper” tag by which the “mapper” tag is nested.
3. **Query:** It is used to describe the mapping relationship between attributes. There are usually four attributes in the “query” tag. The definitions of the key and value attributes in the “query” tag are similar with the ones in “mapper” tag. However, the element in the source model, which the attribute belongs to, are defined by the node and condition attributes, which describe the type of the element and the constraints which the element should follow. Particularly, it is usually used to describe the many-to-one mapping rule between model elements.

Based on the key words above, we can define the mapping rules between elements, according to their mapping relationships.

	One-to-One Mapping Rule	Many-to-One Mapping Rule	One-to-Many Mapping Rule
Example	A → B A1.a1 → B1.b1	A → B A1.a1 → B1.b1 C1.c1 → B1.b2	A → B or C A1.a1 → B1.b1 A2.a1 → C1.c1
Get	Get A1.a1 → Get B1.b1	Get A1.a1 → Get B1.b1 Get C1.c1 → Get B1.b2	Get A1.a1 → Get B1.b1 Get A2.a1 → Get C1.c1
Set	Set A1.a1 → Set B1.b1	Set A1.a1 → Set B1.b1 Set C1.c1 → Set B1.b2	Set A1.a1 → Set B1.b1 Set A2.a1 → Set C1.c1
List	List *A → List *B <u>Get A. properties → Get B. properties</u>	List *A → List *B <u>Get A. properties → Get B. properties</u>	List *A → List *B and List *C <u>Get A. properties → Get B. properties or</u> <u>Get C. properties</u>
Add	Add *A → Add *B <u>Set A. properties → Set B. properties</u>	Add *A → Add *B <u>Set A. properties → Get C. properties</u> <u>and Set B. properties</u>	Add *A → Add *B or Add *C <u>Set A. properties → Set B. properties or</u> <u>Set C. properties</u>
Remove	Remove *A → Remove *B	Remove *A → Remove *B	Remove *A → Remove *B or Remove *C

Figure 5. Mapping Rules of Model Operation Transformation

B. Model Operation Transformation

When we manage the resources through operations on the customized model, they need to be transformed to ones on Cloud resources runtime models. The basic types of Operations include “Get”, “Set”, “List”, “Add” and “Remove”. Fig. 4 shows different types of model operations and their execution effects on the models.

Name	Meta element	Parameter	Description
Get	Property(1)	-	Get the value of the property
Set	Property(1)	newValue	Set the property as newValue
List	Property(*)	-	Get a list of values of this property
Add	Property(*)	toAdd	Add toAdd into the value list of this property
Remove	Property(*)	toRemove	Remove toRemove from the list of this property

Fig. 4. All Kinds of Operations

As shown in Fig. 3, there are three types of basic mapping rules between model elements. According to the mapping rules, the operations on the element in the source model can be mapped to the operations on the related element in the objective model. We have defined the mapping rules of operation transformation, as shown in Fig. 5.

One-to-one mapping rule: The *A* element in the source model is mapped to the *B* element in the objective model, and it is a one-to-one mapping rule. Thus, the operation to add, remove or list the *A* elements is mapped to the same operation on the related *B* element. And the operation to get or set the value of *A*’s attribute is mapped to the same operation on the related attribute.

Many-to-one mapping rule: The *A* element in the source model is mapped to the *B* element in the objective model, and the attribute of the *C* element is also corresponding to the attribute of the *A* element. It is a many-

to-one mapping rule. Thus, the operation to get or set the value of the attribute of the *A* or *C* element is mapped to the same operation on the related attribute of the *B* element. And the operation to add, remove or list the *A* elements is mapped to the same operation on the related *B* element. In addition, when the *B* element is created, the initial values of properties come from both of the *A* and *C* elements.

One-to-many mapping rule: The *A* element in the source model is mapped to one of the *B* and *C* elements in the objective model, according to some attribute of the *A* element. It is a one-to-many mapping rule. The operations are mapped to the same ones on the related elements or attributes. Particularly, the operation to list the *A* elements is mapped to the operation to list all of the related elements, including the *B* and *C* elements.

V. CASE STUDY

In a Cloud environment, the hardware and software resources of virtual machines need to be managed together in order to optimize the allocation of resources. However, to the best of our knowledge, there is currently no open source product to satisfy the management requirement above. There are many Cloud management systems provide solutions to managing specific kinds of Cloud resources. For instance, OpenStack [8] is an open source product which is aimed to manage Cloud infrastructure. Hyperic [9] is another product which manages different kinds of software including web servers, application servers, database servers, and so on.

In order to validate the feasibility and efficiency of our approach, we implement a prototype for the unified management of the hardware and software resources of virtual machines based on OpenStack and Hyperic. Then we

conduct some experiments on the prototype to make an evaluation.

A. Construction of the Models

OpenStack is aimed to manage the entire life cycle of virtual machines. The management elements in OpenStack include *Project*, *Server*, *Flavor*, *Image* and so on, as shown in Fig. 6. The virtual machine (the *Server* element) is the basic unit of resource allocation, each of which is included in a project. The resources of the infrastructure are divided into several projects. The configuration of the virtual machine contains the image, which describes the file system of the virtual machine, and the flavor, which describes the hardware resource of the virtual machine. The *Images* element contains a list of images which can be used in the project. The *Image* element is regarded as one type of image (For instance, web server image and DB server image). The *Flavors* element contains a list of flavors which can be used in the project. The *Flavor* element describes one type of hardware resource configuration (such as tiny-flavor: CPU 1G, Memory 512M; large-flavor: CPU 4G, memory 8G).

Hyperic provide management interfaces of middleware software products, which is based on the agents (the *Agent* element) deployed on each managed node. Due to the large number of management interfaces, the model of Hyperic in this case just contains the main management interfaces of Apache, JOnAS and MySQL, as shown in Fig. 9. The attributes of the *Apache*, *JOnAS* and *MySQL* elements reflect the metrics and configurations of the running systems.

The correct synchronization between the runtime models and the management systems are guaranteed by the SM@RT tool. Thus, administrators can manage the hardware and software resource at an architecture level separately.

In the management requirement above, the virtual machine and the software deployed can be regarded as an appliance [10], which is the basic managed unit. Several appliances compose a project that provides the infrastructure and software resources to a distributed application system. According to this management style, we construct the customized model. Fig. 6 shows the main elements in the customized model. The *Project* element contains an *Appliances* element, which is regarded as a list of appliances. The *Appliances* element contains a list of *Apache* elements, a list of *JOnAS* elements and a list of *MySQL* elements, which are all regarded as appliances. The elements of each appliance contain configurations of the hardware and software resources. For instance, the *Apache* element contains an *ApacheSwConfig* element and an *HwConfig* element. So management tasks can be described by sequences of operations on the customized model.

B. Model Operation Transformation

The customized model reflects the specific management style and the composite model stands for the specific managed resources. In order to ensure the correct synchronization between the two models, we define the mapping rules between their elements. The key challenge is to describe the mapping from the *Apache*, *JOnAS* and *MySQL* elements in the customized model to the *Server* and

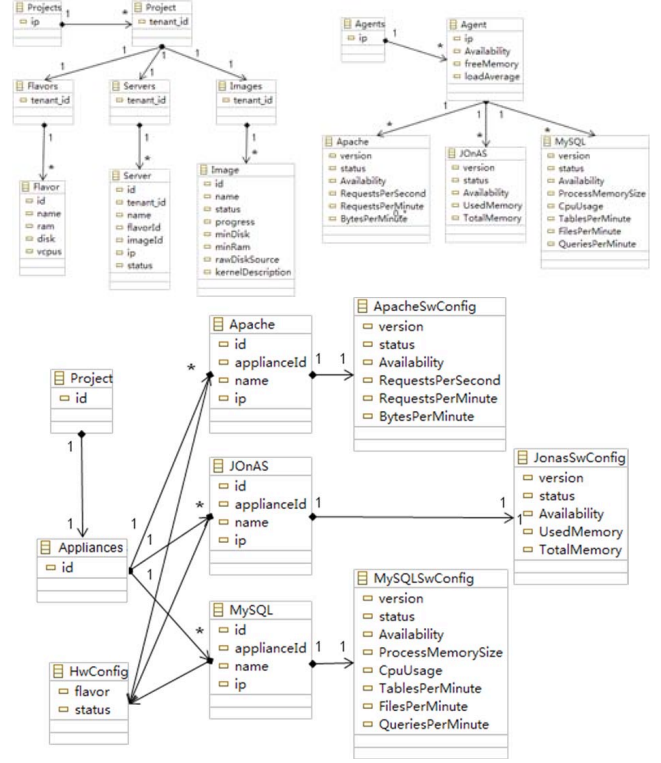


Fig.6. The Architecture-based Models of OpenStack (up-left), Hyperic (up-right) and the Objective Management System

Agent elements in runtime models of *OpenStack* and *Hyperic*. We take the *Apache* element for an example, as shown in Fig. 7.

1. The *Apache* element is mapped to the *Server* element in the OpenStack model and the mapping condition is if their *id* attributes has the same value. It is a one-to-one mapping rule. The *applianceId* attribute of the *Apache* element describes the type of image, which is mapped to the *imageId* attribute of the *Server* element, as the same as the *name* and *ip* attributes. The *flavor* attribute of the *HwConfig* element describes the type of flavor, which is mapped to the *flavorId* attribute of the *Server* element, as the same as the *status* attribute.

```

.....
<helper key="Appliances" value="Servers" condition="id=self.tenant_id" type="composite">
  <helper key="Apache" value="Server" condition="id=self.id" type="composite">
    <mapper key="name" value="name" type="basic" />
    <mapper key="applianceId" value="imageId" type="basic" />
    <mapper key="ip" value="ip" type="basic" />
    <helper key="HwConfig" value="Server" type="composite">
      <mapper key="flavor" value="flavorId" type="basic" />
      <mapper key="status" value="status" type="basic" />
    </helper>
  </helper>
.....
<helper key="Apache" value="Agent" condition="ip=self.ip" type="composite">
  <helper key="ApacheSwConfig" value="Apache" type="composite">
    <mapper key="version" value="version" type="basic" />
    <mapper key="status" value="status" type="basic" />
    <mapper key="Availability" value="Availability" type="basic" />
    <mapper key="UsedMemory" value="UsedMemory" type="basic" />
    <mapper key="TotalMemory" value="TotalMemory" type="basic" />
  </helper>
.....

```

Fig.7. Snippets of Mapping Rules

2. The *Apache* element is mapped to the *Agent* element in the Hyperic model and the mapping condition is if their *ip* attributes has the same value. It is a one-to-one mapping rule. The attributes of the *ApacheSwConfig* element describe the configurations of the software resource, which are mapped to the attributes of the *Apache* element in the Hyperic model.

According to the mapping rules, the operations on the element in the customized model can be mapped to the operations on the related element in the runtime models. Then the operation will be transferred and executed on the runtime models. Fig. 8 shows an example of this progress. The operation to create an *Apache* element is mapped to the one to create a *Server* element in the OpenStack runtime model. The file transferred to the runtime model describes the action in detail:

1. Query: Find the *Servers* element whose *tenant_id* attribute has the value of “f9764071”.
2. Add: Create a *Server* element.
3. Set: Do configuring for the server according to the action description.

When the operation is executed, the changes on the runtime model are reacting on the running system at the same time.

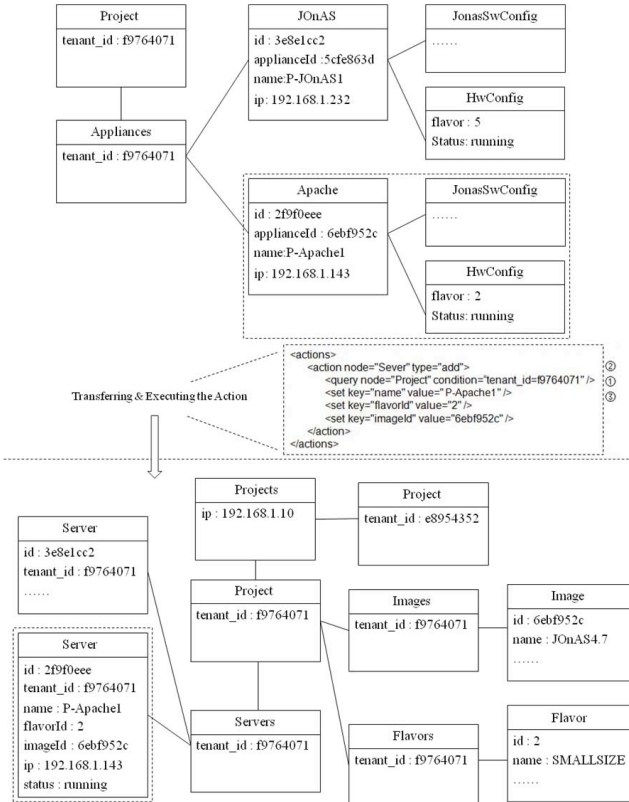


Fig.8. The action to create a virtual machine executed on the runtime model of OpenStack

C. Evaluation

In the approach above, we have developed an architecture-based tool for the unified management of the hardware and software resources of virtual machines, as shown in Fig. 9. Every *Appliance* element stands for a virtual machine with software deployed and these elements compose the runtime model of the running system. Administrators can manage the resources at an architecture level and the operations are transformed to the invocations of management interfaces of underlying Cloud resources. Particularly, we just reuse and reorganize the management interfaces provided by OpenStack and Hyperic, instead of modifying the two management systems.

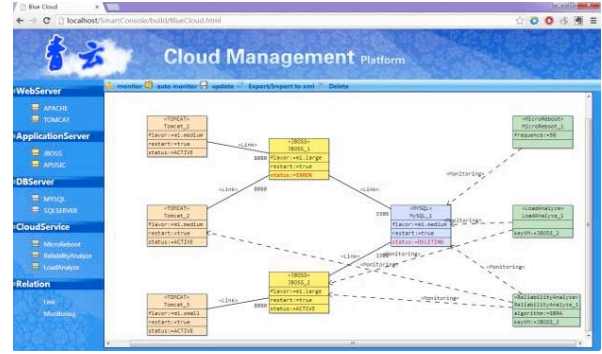


Fig. 9. The architecture-based tool for the unified management of the hardware and software resources of virtual machines

We manage a Cloud infrastructure, which consists of 15 physical servers and supports about 100 appliances, through our runtime model based tool. To evaluate its performance, we develop Java and QVT [11] programs to execute two groups of management tasks, respectively based on the management interfaces or the runtime model. The first group of management tasks is to query properties of the appliances, and the second group of management tasks is to create a set of appliances, as shown in Fig 10. The execution time of Java programs is less than the QVT ones. The main reason is that the two sets of programs are based on the same management APIs and there are some extra operations in runtime model based approach, which are aimed to ensure the synchronization between the architecture-based models and the underlying systems. However, the difference is small and completely acceptable for Cloud management.

Management Tasks	the Number of the Appliance	Management Interfaces	Runtime Model
		Execution Time (second)	Execution Time (second)
Monitoring	5	1.2	2.6
	20	4.2	8.5
	100	20	37
Executing	1	0.2	0.5
	5	1.0	1.7
	20	4.0	6.1

Fig. 10. The performance test result of the unified management tool

VI. RELATED WORK

There are many management systems which are aimed to manage different types of resources and provide different kinds of services. For instance, Eucalyptus, OpenStack, Tivoli and Hyperic separately help administrators manage certain types of Cloud resources. However, none of them can adjust and extend its management interfaces efficiently and flexibly to satisfy different management requirements.

There are some research works which try to integrate management systems based on service-oriented architecture. For instance, Heiko Ludwig et al. [12] propose a solution to the system management in a distributed environment, which encapsulates management functions into RESTful services and makes them subscribed by administrators. In general, it is feasible to integrate management function based on service-oriented architecture. However, management services are not as good as system parameters for reflecting the states of running systems, and service subscription and composition are also more complicated, which may lead to extra difficulties in system management.

Architecture-based approach is usually used in system management. For instance, the work [13] is proposed to support the configuration and deployment of services with an automated closed control loop. The automation is enabled by the definition of a generic information model, which captures all the information relevant to the management of the services, describing the runtime elements, service dependencies and business objectives. In those works, the architecture-based models are used to analyze management problems, but they cannot be used to monitor the systems or to execute some management operations, because the correct synchronization between the architecture-based models and the running systems cannot be guaranteed.

We have made some researches in the area of model driven engineering. For a given meta-model and a given set of management interfaces, SM@RT [5] [6] [7] can automatically generate the code for mapping models to interfaces with good enough runtime performance. In the work [14], we have tried to construct the runtime model of a real-world Cloud and develop management programs in a modeling language. But it is difficult to construct the runtime model to satisfy every management requirement from scratch. In the work [15], based on runtime models, specific Cloud management requirements can be satisfied with executing a set of model operating programs through model construction, merge and transformation. However, the composite model causes the data delay of management programs, which increases linearly with the scale of the underlying systems.

VII. CONCLUSION AND FUTURE WORK

Due to the diversity of Cloud resources and specific management requirements, Cloud management is faced with huge challenges. To satisfy a new specific management requirement, the most common way is conducting redevelopment based on existing management systems. However, the difficulty and workload of redevelopment are

very high. To avoid the difficulty and heavy workload of redevelopment, this paper proposed an architecture-based approach to integrated management of Cloud resources. We construct runtime models of Cloud resources and the customized model to satisfy the specific management requirement. The operations on the customized model are mapped to the ones on Cloud resource runtime models through model transformation. Thus, all the management tasks can be carried out through executing operations on the customized model.

For the future work, we plan to add some more advanced management functions with the help of model analysis and reasoning techniques to ease the tasks of Cloud management.

ACKNOWLEDGMENTS

This work was supported by the Technology Innovation Platform Project of Fujian Province under Grant No. 2009J1007 and the Research Program of Fuzhou University under Grant No. 022543.

REFERENCES

- [1] P. Mell and T. Grance. NIST definition of cloud computing. National Institute of Standards and Technology. October 7, 2009.
- [2] Garlan, D. Software Architecture: A Roadmap. In: Proc. of the 22nd International Conference on Software Engineering, Future of Software Engineering Track. New York: ACM Press, 2000. 91-101.
- [3] Mei H, Shen JR. Progress of research on software architecture. Journal of Software, 2006, 17(6): 1257-1275 (in Chinese with English abstract). <http://www.jos.org.cn/1000-9825/17/1257.htm>
- [4] Robert France, Bernhard Rumpe. Model-driven Development of Complex Software: A Research Roadmap. In: Proc. of the 29th International Conference on Software Engineering, Future of Software Engineering Track. Washington: IEEE Computer Society Press, 2007. 37-54.
- [5] Gang Huang, Hui Song, Hong Mei. SM@RT: Applying Architecture-based Runtime Management of Internetwork Systems. International Journal of Software and Informatics, 2009, 3(4): 439-464.
- [6] SM@RT: Supporting Models at Run-Time, <http://code.google.com/p/smatr/>
- [7] Hui Song, Gang Huang, Franck Chauvel, Yingfei Xiong, Zhenjiang Hu, Yanchun Sun, Hong Mei. Supporting Runtime Software Architecture: A Bidirectional-Transformation-Based Approach. Journal of Systems and Software, 2011, 84(5): 711-723.
- [8] OpenStack. <http://www.openstack.org/>
- [9] Hyperic. <http://www.hyperic.com/>
- [10] Virtual appliance. http://en.wikipedia.org/wiki/Virtual_appliance
- [11] QVT. <http://en.wikipedia.org/wiki/QVT>
- [12] Heiko Ludwig, Jim Laredo, Kamal Bhattacharya. Rest-based management of loosely coupled services. In: Proc. of the 18th International Conference on World Wide Web. New York: ACM Press. 2009. 931-940.
- [13] Félix Cuadrado, Juan C. Dueñas, Rodrigo García-Carmona. An Autonomous Engine for Services Configuration and Deployment. IEEE Transactions on Software Engineering, 2012, 38(3): 520-536.
- [14] Gang Huang, Xing Chen, Ying Zhang, Xiaodong Zhang: Towards architecture-based management of platforms in the cloud. Frontiers of Computer Science 6(4): 388-397 (2012)
- [15] Xiaodong Zhang, Xing Chen, Ying Zhang .etc. Runtime Model Based Management of Diverse Cloud Resources. In: Proc. of the 16th International Conference on Model Driven Engineering Languages and Systems. 2013. Accepted.