Journal of Cloud Computing
a SpringerOpen Journal

RESEARCH Open Access

Architecture-based integrated management of diverse cloud resources

Xing Chen[1,2], Ying Zhang[3,4], Gang Huang[3,4], Xianghan Zheng[1,2*], Wenzhong Guo[1,2] and Chunming Rong[5]

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 systems 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. In this paper, we present an architecture-based approach to managing diverse Cloud resources. First, manageability of Cloud resources is abstracted as runtime models, which could automatically and immediately propagate any observable runtime changes of target resources to corresponding architecture models, and vice versa. Second, a customized model is constructed according to the personalized management requirement and the synchronization between the customized model and Cloud resource runtime models is ensured through model transformation. Thus, all the management tasks could be carried out through executing programs 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

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]. 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:

First, the virtualization not only makes the physical resources easier to share and control but also increases the complexity of management [2]. 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. All these resources have to be managed together.

Second, there are kinds of personalized management requirements. In some scenarios, administrators need to manage different kinds of resources together; while in other scenarios, administrators have to manage Cloud resources in appropriate styles [2]. 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 could usually sustain running of several applications simultaneously. What's more, all of the platforms are in a resource sharing and competing environment [3].

* Correspondence: xianghan.zheng@fzu.edu.cn
[1]College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350116, China
[2]Fujian Provincial Key Laboratory of Networking Computing and Intelligent Information Processing (Fuzhou University), Fuzhou 350116, China
Full list of author information is available at the end of the article

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, management tasks of different Clouds are not the same. For instance, Amazon EC2 [4] mainly manages infrastructure level Cloud resources such as virtual machines, while Google App Engine [5] manages platform-level Cloud resources such as operating systems execution environment. To satisfy the personalized management requirements, Cloud administrators usually conduct redevelopment based on the existing management systems. 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 efforts and costs. 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 code. Administrators first have to be familiar with these APIs and then build programs upon them. Such a work is not easy due to diverse resources and personalized requirements. 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 management logics compared with analyzing and planning related code, 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, the programs are built on the code that directly connects with runtime systems, so they are not easy for reuse. Administrators have to write many different programs to manage different cloud applications and their platforms, even their 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 [6]. 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 [7]. It is a natural idea to understand management tasks through modeling the architecture of the system. Current researches in the area of model driven engineering (MDE) also support systematic transformation of problem-level abstractions to software implementations [8].

To address the issues above, we try to leverage architecture-based runtime model for the management of diverse Cloud resources. An architecture-based runtime model is a causally connected self-representation of the associated system that emphasizes the structure, behavior, and goals of the system from a problem space perspective [9,10]. It has been broadly adopted in the runtime management of software systems [11-13]. With the help of runtime models, administrators can obtain a better understanding of their systems and write model-level programs for management. We have developed a model-based runtime management tool called SM@RT (Supporting Model AT Run Time [14-16]), which provides the synchronization engine between a runtime model and its corresponding running system. SM@RT makes any state of the running system reflected to the runtime model, as well as any change to the runtime model applied to the running system in an on-the-fly fashion.

In this paper, we present an architecture-based approach to the integrated management of diverse Cloud resources. First, we construct the architecture-based runtime model of each kind of Cloud resource (Cloud resource runtime model) automatically based on its architecture meta-model and management interfaces. Second, we define a customized model which satisfies the specific management requirement, and describe mapping relationships between the customized model and Cloud resource runtime models. Then any operation on the customized model is transformed to one on Cloud resource runtime models automatically. Finally, management tasks are carried out through executing operating programs on the customized model, which could benefit from many model-centric analyzing or planning methods and mechanisms such as model checkers [17]. 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 apply the runtime model to a real Cloud system, which is a practical evaluation on architecture-based integrated management of diverse Cloud resources.

The rest of this paper is organized as follows: Section II gives a motivating example of the architecture-based approach to managing diverse Cloud resources. Section III presents the construction of Cloud resource runtime models. Section IV describes the construction of the customized model. 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.

Motivating example

In order to satisfy personalized requirements, Cloud administrators conduct redevelopment based on existing management systems. However, it may result in several difficulties of integrated management in general approach. For instance, management scenarios may consist of different types of resources which need to be managed collaboratively. Administrators have to be familiar with the APIs and then build programs upon them. While conducting redevelopment, they have to choose proper APIs for use and make different types of APIs interpretable with each other, as shown in Figure 1.

Such code fragments are not the core of management logics, but it has to be developed to make the whole management program run effectively. Many similar code fragments are required for a simple task. As shown in Figure 1, the code fragment for fetching the value of the "maxThreads" attribute in a JOnAS (a popular open source Java application server) through JMX API is more than 20 LOC (Line of Code). During this procedure, the irrelevant APIs as well as the collected low-level data can sometimes make administrators exhausted and frustrated. Furthermore, as programs are built on the code that directly connect with the running systems, they are not easy for reuse. Administrators have to write different programs to satisfy similar requirements even their management objectives are the same.

When using our approach, the procedure becomes much simpler and shorter. Figure 2 shows an overview of the runtime model based approach to integrated management of Cloud resources. The architecture-based runtime models can shield administrators from the relatively low-level details of redevelopment.

There are two steps in our approach. First, we construct runtime models of Cloud resources. The Cloud resource runtime model is abstracted from the software

```
1.  /*
2.   * JAVA: To get the value of the maxThreads of a JOnAS
3.   * through the JMX.
4.   */
5.  public int getMaxThreads(String port)
6.  {
7.      //To prepare to invoke the interface
8.      String objName = "jonas:type=Connector,port=" + port;
9.      String attributeName = "maxThreads";
10.     MBeanServerConnection mbeanServerConn = null;
11.     try
12.     {
13.         JMXServiceURL connURL = new JMXServiceURL(
14.         "service:jmx:rmi://localhost/jndi/rmi://localhost
15.         :1099/jrmpconnector_jonas");
16.         JMXConnector connector = JMXConnectorFactory.
17.         newJMXConnector(connURL, null);
18.         connector.connect(null);
19.         mbeanServerConn = connector.getMBeanServerConnection();
20.     }
        ...
41.     //To invoke the specific management interface
42.     try
43.     {
44.         attributeValue = (Integer) mbeanServerConn.
45.         getAttribute(obj, attributeName);
46.     }
47.     catch (AttributeNotFoundException e)
48.     {
        ...
```

Figure 1 Example of invoking management interfaces.

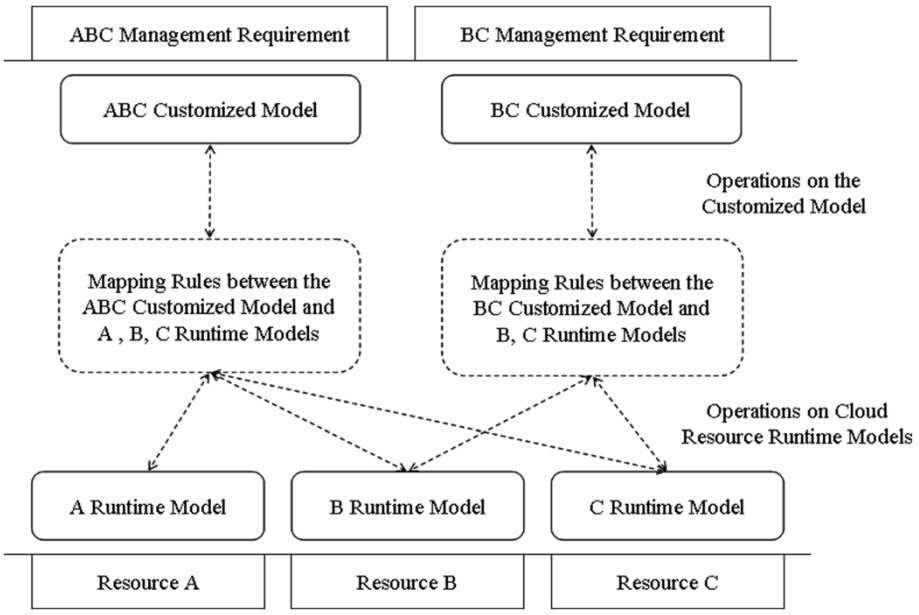

Figure 2 Overview of the architecture-based approach to managing diverse Cloud resources.

architecture of this kind of resources and the correct synchronization between the runtime model and the running system is ensured, which shields the heterogeneity of management interfaces. Second, according to the specific management style, we construct a customized model and ensure its correct synchronization with Cloud resource runtime models through model transformation. In our approach, we only need to define a group of meta-models, mapping rules and model-level programs, so the workload of hand coding can be greatly reduced.

Construction of cloud resource runtime models

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 [18] and Nginx [19], database servers such as MySQL, SQL server and Oracle. We construct their runtime models in order to manage them in a unified manner. The runtime model is abstracted from their software architecture. It is done easily with the help of SM@RT (The source code of SM@RT can be downloaded from [16]), which is proposed in our previous work [14,15].

SM@RT consists of a domain-specific modeling language (called SM@RT language) and a code generator (called SM@RT generator) to support model-based runtime system management. The SM@RT language allows developers to specify: (1) the structure of the running system by a UML-compliant meta-model; (2) how to manipulate the system's elements by an access model. With these two models, the SM@RT generator can automatically generate the synchronization engine to reflect the running system to the runtime model. The synchronization engine not only enables any states of the system to be monitored by the runtime model, but also any changes to the runtime model to be applied on the running system. Thus we can manage the resources through operations on the runtime models, and these operations will finally propagate to the underlying cloud resources. For instance, in Figure 3, 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 [20-22].

Construction of the customized model

There are different management requirements in Cloud environment due to diverse Cloud resources and management styles. Different types of resources usually need to run collaboratively to support the Cloud application and the resources should be managed in an appropriate management style. In our approach, administrators just need to construct a customized model and define a set of mapping rules, in order to satisfy a specific management requirement. The customized model is abstracted from the software architecture of the required management system. The correct synchronization between the customized model and Cloud resource runtime models

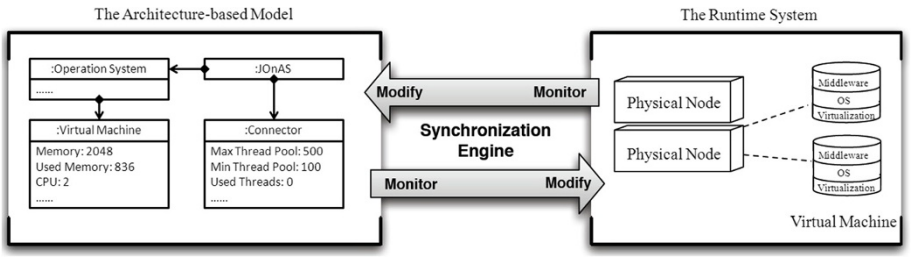

Figure 3 Synchronization between the runtime model and the running system.

is ensured through model operation transformation, which is completed automatically according to the mapping rules.

Mapping rules definition

Mapping rules are used to describe mapping relationships between the customized model and Cloud resource runtime models. Every element in the customized model is related with one in Cloud resource runtime models. As shown in Figure 4, there are three types of basic mapping relationships between model elements. Any other mapping relationship can be demonstrated as a combination of them.

One-to-one mapping relationship

One element in the customized model is related to a certain element in Cloud resource runtime models. Particularly, the attributes of elements in the customized model are also corresponding to the ones of related elements in Cloud resource runtime models. For instance, the *Flavor* element in the customized model and the *MachineType* element in Cloud resource runtime models both represent the configuration of virtual machine. The *id, name, memoryMb, imageSpaceGb* and *guestCpus* attributes of the *Flavor* element are related to the *id, name, ram, disk* and *vcpus* attributes of the *MachineType* element.

Many-to-one mapping relationship

One type of element in Cloud resource runtime models is related to two or more types of elements in the customized model. Particularly, the attributes of a certain type of the element in Cloud resource runtime models are related to the attributes of two or more types of elements in the customized model. For instance, *Image* and *Kernel* elements in the customized model are both used to describe the information about the type of virtual machine. In Cloud resource runtime models, all the related information is described in the *Image* element. However, in the customized model, there is not any attribute of the *Image* element, related to the *kernelDescription* attribute of the *Image* element in Cloud resource runtime

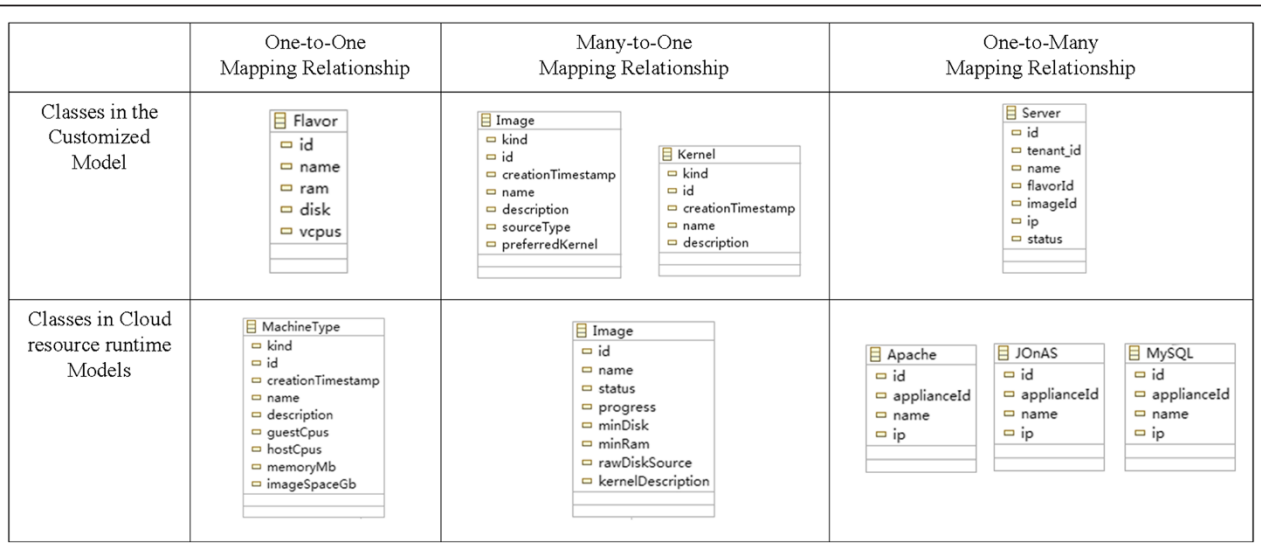

Figure 4 Three types of basic mapping relationships between model elements.

models. The related attribute is in the *Kernel* element, which is indicated by the *preferredKernel* attribute of the *Image* element in the customized model.

One-to-many mapping relationship

One type of elements in the customized model is related to two or more types of elements in Cloud resource runtime models. For instance, *Server* elements in the customized model represent virtual machines, and *Apache*, *JOnAS* and *MySQL* elements in Cloud resource runtime models represent virtual machines with software deployed. During model transformation, any Server element is mapped to one of *Apache*, *JOnAS* and *MySQL* elements, according to its *imageId* attribute.

As shown in Table 1, we have defined some keywords and presented the method to describe mapping relationships between the customized model and Cloud resource runtime models.

1. Helper: The "helper" tag 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 target element in the customized model and the *key* attribute describes the target element in Cloud resource runtime models. The *type* attribute describes the type of the mapping relationship. When its value is "basic", it is a one-to-one mapping relationship or a many-to-one mapping relationship. When its value is "multi", it is a one-to-many mapping relationship. The "helper" tag is used to describe the mapping relationship between elements. Elements often have attributes or other elements, so the "helper" tag usually nests "helper" tags, "mapper" tags and "query" tags.
2. Mapper: The "mapper" tag is used to describe the mapping relationship between attributes of elements. There are usually two attributes in the "mapper" tag, the *key* and *value* attributes. The *value* attribute describes the target attribute in the customized model and the *key* attribute describes the target attribute in Cloud resource runtime models. The element, which the attributes belong to, is defined in the outer "helper" tag.
3. Query: The "query" tag is used to describe the mapping relationship between attributes of elements. There are usually four attributes in the "query" tag. The *key* and *value* attributes in the "query" tag are similar with the ones in "mapper" tag. But the element, which the attribute belongs to, is defined by the node and condition attributes; the *node* attribute describes the type of the target element and the *condition* attribute describes the constraint that the target element should follow. The "query" tag is usually used in the descriptions of many-to-one mapping relationships between elements.

Based on the key words above, we could define the mapping rules between elements, according to their mapping relationships. As shown in Figure 5, there are three cases of basic mapping relationships between elements.

One-to-one mapping relationship

The first case is to describe the One-to-One mapping relationship between *Flavor* elements in the customized model and *MachineType* elements in Cloud resource runtime models. The "helper" tag is used to describe this mapping relationship. The value of the *key* attribute is "machineType" and the value of the *value* attribute is "flavor". The "mapper" tags are used to describe the mapping relationships between the attributes of *Flavor* and *MachineType* elements.

Many-to-one mapping relationship

The second case is to describe the Many-to-One mapping relationship between *Image*, *Kernel* elements in the customized model and *Image* elements in Cloud resource runtime models. The "helper" tag is used to describe this mapping relationship. The "mapper" tag is for describing the mapping relationships between the attributes of *Image* elements in the models above. The "query" tag is to describe the mapping relationship between *description* attributes of *Kernel* elements in the customized model and *kernelDescription* attributes of *Image* elements in Cloud resource runtime models. The related *Kernel* element is indicated by the *preferredKernel* attribute of *Image* element in the customized model.

Table 1 Keywords used to describe mapping relationships

Keywords	Descriptions	Keywords	Descriptions
Helper	Mapping Rules between Elements	**Type**	Types of Mapping Relationships
Mapper	Mapping Rules between Attributes	**Query**	Mapping Rules between Attributes
Key	Elements or Attributes in the Objective Model	**Value**	Elements or Attributes in the Source Model
Condition	Preconditions	**Node**	Types of Elements

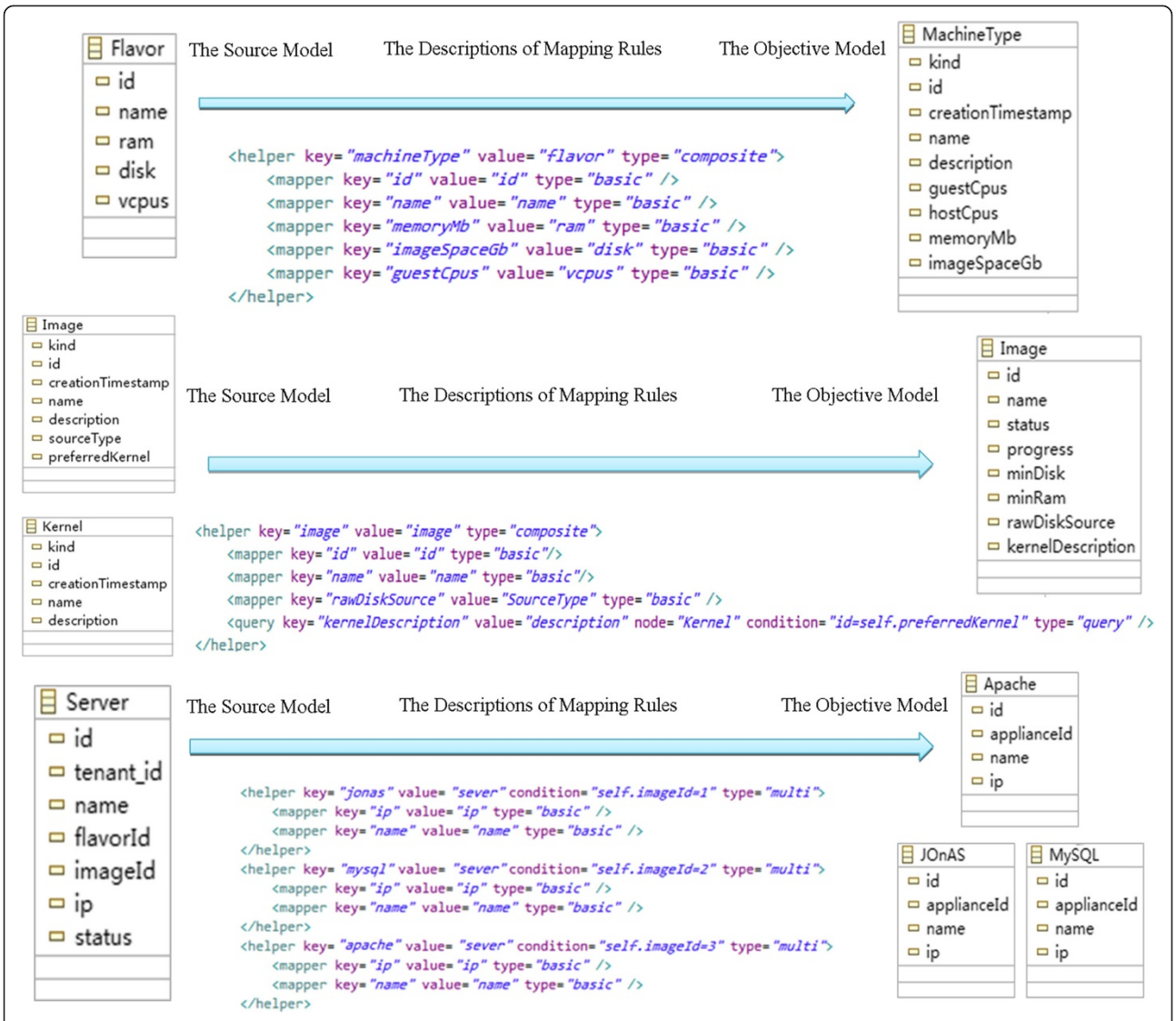

Figure 5 Descriptions of basic mapping relationships between elements.

One-to-many mapping relationship

The third case is to describe the One-to-Many mapping relationship between *Server* elements in the customized model and *Apache*, *JOnAS*, *MySQL* elements in Cloud resource runtime models. The "helper" tag is used to describe the One-to-Many mapping relationship, so the value of its *type* attribute is "multi". The *condition* attribute in the "helper" tag is to describe the mapping precondition. For instance, if the value of the *imageId* attribute of the Server element is "1", the *Server* element is mapped to the *JOnAS* element.

Model operation transformation

Model operations are aimed to monitor some system parameters or execute some management tasks. There are five basic types of model operations, including "Get", "Set", "List", "Add" and "Remove". In order to ensure the correct synchronization between the models, operations on the customized model need to be transformed to ones on Cloud resource models, as shown in Figure 2.

We define the description and execution effect of each type of basic model operation, as shown in Figure 6. When Cloud administrators operate on the customized model, an operation file will be generated automatically, which is described in the form of "action" tag. If the type of operation is "Get" or "List", the result file is required, which is describe in the form of "return" tag. Particularly, the *condition* attribute of the operation or result tag usually describes the identification of objective element, such as "id = f9764071".

List	Description	<action node="TypeS" type= " List"> <query node="TypeF" condition="Constraint" /> </action> <ruturn> <node="TypeS" condition="Constraint1" /> <node="TypeS" condition="Constraint2" /> …… </return>
	Post Condition	∃ TypeS s1, s2… ∃ TypeF f, s1, s2… ∈ f ∧ f in condition of Constraint
	Illustration	Find the "TypeF" element which satisfies the constraints and list "TypeS" elements which are its child nodes.
Get	Description	<action key="KEY" type="get"> <query node="Type" condition="Constraint" /> </action> <return key="KEY" value="VALUE" />
	Post Condition	∃ Type n, n in condition of Constraint ∧ prop ∈ n.properties
	Illustration	Find the "Type" element which satisfies the constraints and get the value of its "KEY".
Add	Description	<action node="TypeS" type="add"> <query node="TypeF" condition="Constraint" /> <set key="" value="" /> <set key="" value="" /> </action>
	Post Condition	∃ TypeS s, ∃ TypeF f, s ∈ f ∧ f in condition of Constraint ∧ props ⊆ s.properties
	Illustration	Find the "TypeF" element which satisfies the constraints and add a "TypeS" element as its child node.
Set	Description	<action key="KEY" value="VALUE" type="set"> <query node="Type" condition="Constraint" /> </action>
	Post Condition	∃ Type n, n in condition of Constraint ∧ prop ∈ n.properties
	Illustration	Find the "Type" element which satisfies the constraints and set the value of its "KEY" attribute to "VALUE".
Remove	Description	<action node="Type" condition="Constraint" type="remove" />
	Post Condition	∀ Type n, n not in condition of Constraint
	Illustration	Find the "Type" element which satisfies the constraints and remove it.

Figure 6 Basic Types of Model Operations.

As shown in Figure 5, there are three types of basic mapping relationships between model elements. We have defined the model operation transformation rules, as shown in Table 2. Then the operations on the element in the customized model can be transformed to the operations on the related element in Cloud resource runtime models automatically, according to the mapping relationships.

One-to-one mapping relationship

For instance, *A* elements in the customized model are mapped to *B* elements in Cloud resource runtime model. Thus, operations to add, remove and list *A* elements are mapped to the same operations on related *B* elements. The operation to get or set the value of *A*'s attribute is mapped to the same operation on the related attribute too.

Many-to-one mapping relationship

For instance, *A* elements in the customized model are mapped to *B* elements in Cloud resource runtime model, but some attributes of *B* element are related to ones of *C* element in the customized model too. Thus, the operation to get or set the value of the attribute of *A* or *C* element is mapped to the same operation on the related attribute of *B* element. The operation to add, remove or list *A* elements is also mapped to the same operation on related *B* elements. In addition, when a

Table 2 Mapping rules of model operation transformation

	One-to-one	Many-to-one	One-to-many
	Mapping rule	**Mapping rule**	**Mapping rule**
Example	A -> B	A -> B	A -> B or C
	A1.a1 -> B1.b1	A1.a1 -> B1.b1	A1.a1 -> B1.b1
		C1.c1 -> B1.b2	A2.a1 -> C1.c1
Get	Get A1.a1 -> Get B1.b1	Get A1.a1 -> Get B1.b1	Get A1.a1 -> Get B1.b1
		Get C1.c1 -> Get B1.b2	Get A2.a1 -> Get C1.c1
Set	Set A1.a1 -> Set B1.b1	Set A1.a1 -> Set B1.b1	Set A1.a1 -> Set B1.b1
		Set C1.c1 -> Set B1.b2	Set A2.a1 -> Set C1.c1
List	List*A -> List*B	List*A -> List*B	List*A -> List*B and List*C
	Get A.properties -> Get B.properties	Get A.properties -> Get B.properties	Get A.properties -> Get B.properties or Get C.properties
Add	Add*A -> Add*B	Add*A -> Add*B	Add*A -> Add*B or Add*C
	Set A.properties -> Set B.properties	Set A.properties -> Get C.properties and Set B.properties	Set A.properties -> Set B.properties or Set C.properties
Remove	Remove*A -> Remove*B	Remove*A -> Remove*B	Remove*A -> Remove*B or Remove*C

B element is created, the initial values of properties come from both of *A* and *C* elements.

One-to-many mapping relationship

For instance, *A* elements in the customized model are mapped to *B* or *C* elements in Cloud resource runtime models. Thus, the operations are mapped to the same ones on the related elements or attributes. Particularly, the operation to list *A* elements is mapped to the operation to list all of related *B* and *C* elements.

Case study

In a Cloud environment, the hardware and software resources of virtual machines need to be managed together in order to optimize allocation of resources. However, to the best of our knowledge, there is currently no open source product to satisfy the requirement above. There are many Cloud management systems provide solutions to manage different kinds of Cloud resources. For instance, OpenStack [23] is an open source product which is used to manage Cloud infrastructure. Hyperic [24] is an open source product which is used to manage 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 integrated 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.

Construction of cloud resource runtime models

OpenStack is used 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 Figure 7. The virtual machine (the *Server* element) is the basic unit of resource allocation, each of which is included in a project. The resources of infrastructure are divided into several projects. The configuration of virtual machine contains the image, which describes the file system of virtual machine, and the flavor, which describes the hardware resource of virtual machine. The *Images* element contains a list of images which are related to 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 are related to the project. The *Flavor* element describes one type of hardware resource configuration (such as tiny-flavor: CPU 1G, Memory 512 M; 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 only contains the main management interfaces of Apache, JOnAS and MySQL, as shown in Figure 7. The attributes of *Apache*, *JOnAS* and *MySQL* elements represent the metrics and configurations of middleware platforms.

Given the architecture-based meta-models, we also need to identify the changes enabled by the models [22]. There are hundreds of management interfaces in OpenStack and Hyperic, so we can model them into the Access Models [14] through specifying how to invoke the APIs to manipulate each type of elements in the models. For instance, Figure 8 provides several management operations about the virtual machine. For each operation we detail the management operation names, the

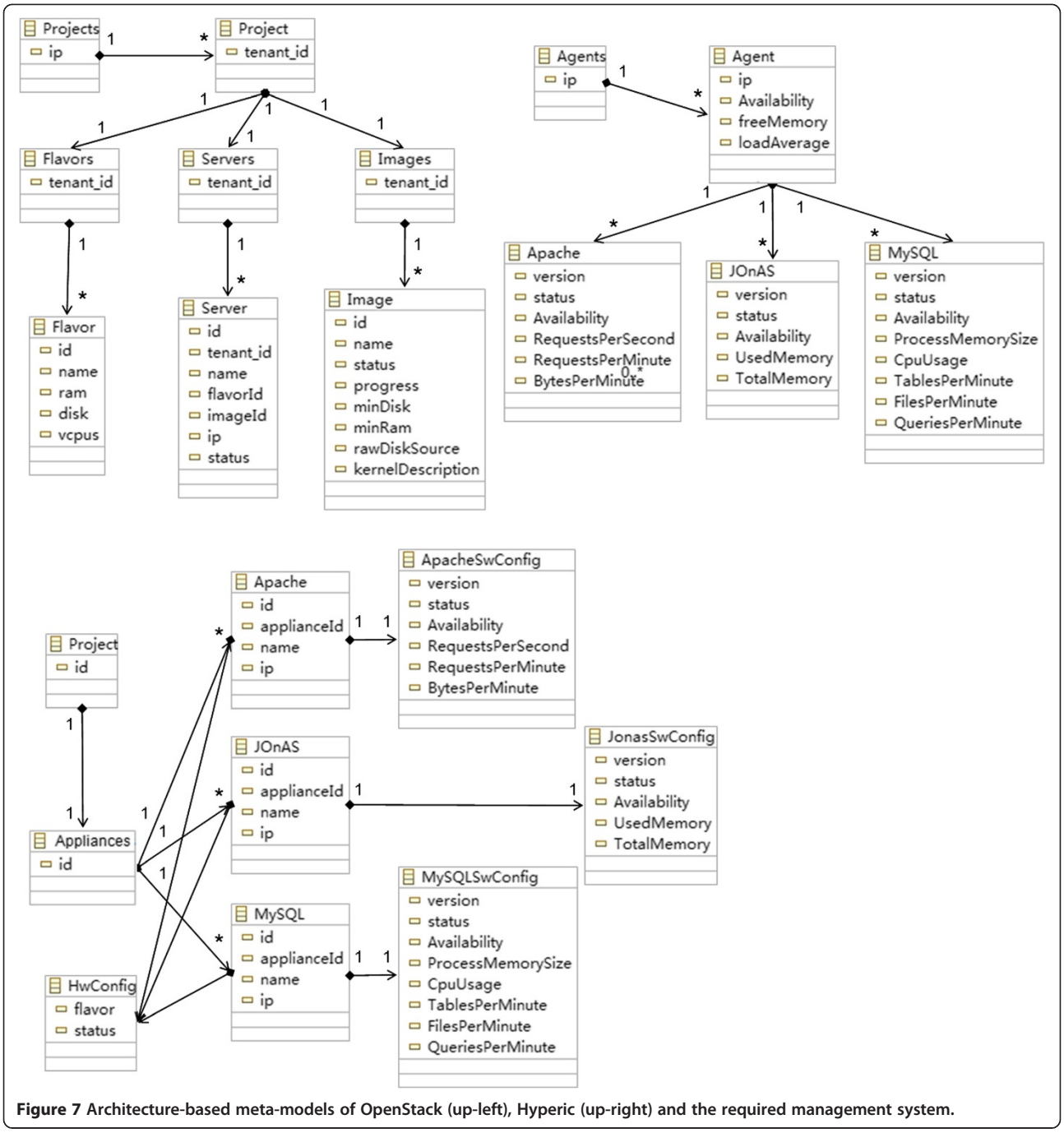

Figure 7 Architecture-based meta-models of OpenStack (up-left), Hyperic (up-right) and the required management system.

required arguments and the changes enabled by the operation. As shown in Figure 8, all types of manipulations of model elements are summarized and management operations are also classified.

Based on architecture-based meta-models and Access Models, the correct synchronization between runtime models and management systems can be guaranteed by the SM@RT tool. Thus, administrators are capable to manage the hardware and software resource at an architecture level separately.

Construction of the customized model

According to the management requirement above, the virtual machine and the software deployed can be regarded as an appliance [25], which is the basic managed unit. Several appliances compose a project that provides the infrastructure and software resources to a distributed application system. We construct a customized model, according to this management style. Figure 7 shows the main elements in the customized model. The *Project* element contains an *Appliances* element, which is regarded as a list of appliances.

Name	Argument	Post Condition
CreateAVM	Node rn, Image ri, Property[] props	∃ VM rv, rv ∈ rn.vms ∧ rv instanceof ri ∧ props ⊆ rv.properties
ShutdownAVM	Node rn, VM rv	rv ∉ rn.vms
MigrateAVM	Node rnI, VM rv, Node rnO	rv ∉ rnI.vms ∧ rv ∈ rnO.vms
PauseAVM	Node rn, VM rv	rv ∈ rn.vms ∧ rv.state = STOPED
UnpauseAVM	Node rn, VM rv	rv ∈ rn.vms ∧ rv.state = STARTED
ConfPProp	RuntimeUnit ru, Property[] props	props ⊆ ru.properties

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
Lookfor	Class	Condition	Find an element according to condition
Identify	Class	Other	Check if this element equals to other
Auxiliary	Package	-	User-defined auxiliary operations

Internal Changes	Manipulation
CreateAVM	Add
ShutdownAVM	Remove
MigrateAVM	Auxiliary
PauseAVM	Set
UnpauseAVM	Set
ConfPProp	Set

Figure 8 Definitions of runtime changes.

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. Therefore, management tasks could be described as the sequences of operations on the customized model.

In order to ensure the correct synchronization between the customized model and Cloud resource runtime models, we define the mapping rules between them according to their mapping relationships, as shown in Figure 9.

The key challenge is to describe the mapping from *Apache*, *JOnAS* and *MySQL* elements in the customized model to *Server* and *Agent* elements in runtime models of *OpenStack* and *Hyperic*. We take the *Apache* element for an example.

1. The *Apache* element is mapped to the *Server* element in the OpenStack model. It is a one-to-one mapping relationship. The *id, applicanceId, name* and *ip* attributes of *Apache* element are mapped to the *id, tenant_id, name* and *ip* attributes of *Server* element. The *flavor* and *status* attributes of *HwConfig* element, are mapped to the *flavorId* and *status* attributes of *Server* element. In addition, the *Apache* element in the customized model represents the appliance with Apache platform deployed and the id of the certain type of virtual machine image is "6ebf952c". So the *imageId* attribute of the related *Server* element in the OpenStack runtime model should be "6ebf952c" too.
2. The *Apache* element in the customized model is mapped to the *Agent* element in the Hyperic model. It is a one-to-one mapping relationship. The *Apache* element in the customized model contains the *ApacheSwConfig* element, which describes the configurations of Apache instance, as the same as the *Apache* element in the Hyperic model. So the *ApacheSwConfig* element in the customized model is mapped to the *Apache* element in the Hyperic model. The attributes of *ApacheSwConfig* element are also mapped to the ones of *Apache* element.

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 runtime models. Figure 10 shows an example of model operation transformation. The original operation is to create an *Apache* element and it is described as follows:

1. Query: Find the *Appliances* element whose id is "f9764071".
2. Add: Create an *Apache* element.
3. Set: Assign property values of the *Apache* element.

The original operation is mapped to the operation to create a *Server* element in the OpenStack runtime model. Model operation transformation is executed instruction-by-instruction. For instance, the action to query the *Appliances* node is mapped to the action to query the *Servers* node, whose tenant_id is "f9764071". The action to add an *Apache* node is mapped to the action to add a *Server* node. The actions to set the property values are mapped to the actions to set the values of related attributes too. According to the mapping relationships, the *imageId* attribute of related *Server* element should be "6ebf952c", so there is an extra action to assign the property value.

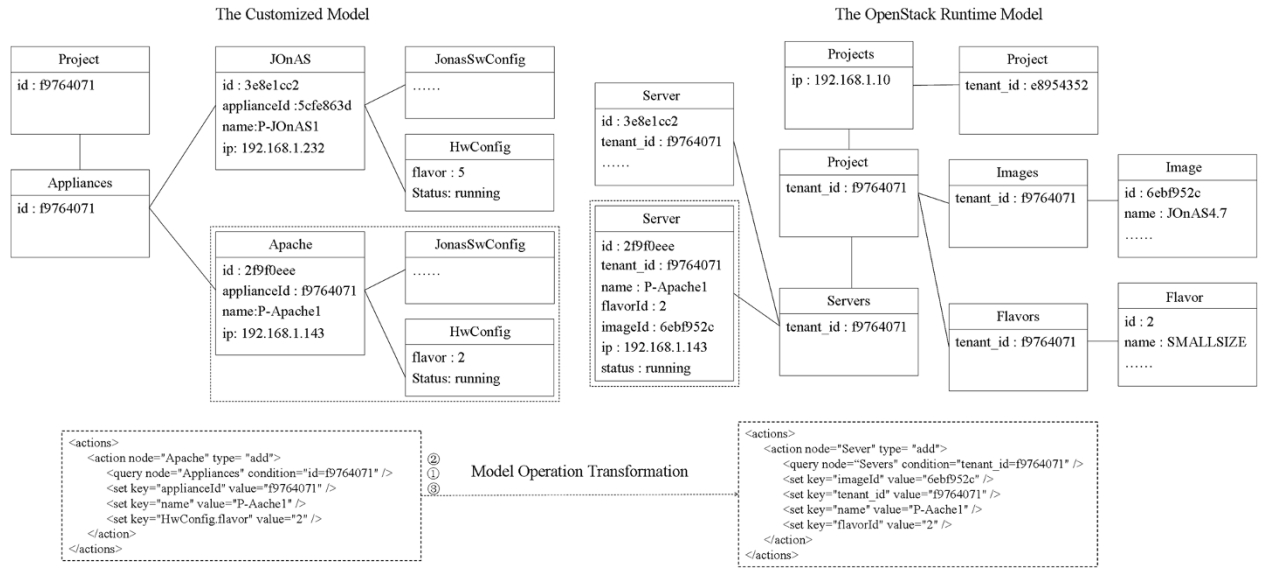

Figure 9 Snippets of mapping rules.

The generated operation file is transferred to the OpenStack runtime model. When the operation is executed, changes of the runtime model will be applied on the running system.

Evaluation

We evaluate our approach from three aspects.

1) Development of the architecture-based tool for integrated management of the hardware and software resources

For constructing Cloud resource runtime models, we just need to define the architecture-based meta-models and the Access Models on the Eclipse Modeling Framework (EMF) [26]. The runtime model will be generated automatically by our SM@RT tool. Construction of Cloud resource runtime models is one-off work, so it is acceptable for Cloud administrators. The existing forms of models and model operations are documents in XML format, and the process of model operation transformation is fulfilled in XML format based on mapping rules.

We have developed an architecture-based tool for integrated management of the hardware and software resources of virtual machines, as shown in Figure 11. 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 management interfaces provided by OpenStack and Hyperic, instead of modifying underlying systems. We manage a cloud infrastructure, which consists of 15 physical servers and supports about 100 appliances, through our runtime model based tool. It has been proved that management tasks could be fulfilled exactly by the tool. Furthermore, complex management tasks could be carried out through executing operating programs on the customized model, which may benefit from existing model-centric analyzing or planning methods and mechanisms.

2) Comparison of programming difficulty between general languages and model languages

According to our previous work [22], for the same management tasks, the programs are simpler to write in model languages like QVT [27], compared with in general languages like Java. With the help of the architecture-based model, Cloud administrators can focus on the logics of management tasks without handling different types of low-level management interfaces. In addition, model languages usually provide model operations such as “select” and “sum”, which makes it simpler to do programming.

3) Comparison of performance between management interfaces and the runtime model

To evaluate the performance of our approach, we develop Java and QVT 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 Table 3. The execution time of Java programs is less than QVT ones. The main reason is that the two sets of programs are based on the same APIs and there are some extra

```
<helper key="Project" value="Project" type="composite">
    <mapper key="tenant_id" value="id" type="basic" />
    <helper key="Servers" value="Appliances" type="composite">
        <mapper key="tenant_id" value="id" type="basic" />
        <helper key="Server" value="Apache" type="composite">                    ①
            <mapper key="imageId" value="" 6ebf952c"" type="basic" />
            <mapper key="id" value="id" type="basic" />
            <mapper key="tenant_id" value="applianceId" type="basic" />
            <mapper key="name" value="name" type="basic" />
            <mapper key="ip" value="ip" type="basic" />
            <mapper key="flavorId" value="HwConfig.flavor" type="basic" />
            <mapper key="status" value="HwConfig.status" type="basic" />
        </helper>
        <helper key="Server" value="JOnAS" type="composite">
        ……
        </helper>
        ……
    </helper>
    ……
</helper>
<helper key="Agent" value="Apache" type="composite">                             ②
    <mapper key="ip" value="ip" type="basic" />
    <helper key="Apache" value="ApacheSwConfig" 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="RequestsPerSecond" value="RequestsPerSecond" type="basic" />
        <mapper key="RequestsPerMinute" value="RequestsPerMinute" type="basic" />
        <mapper key="BytesPerMinute" value=" BytesPerMinute" type="basic" />
    </helper>
</helper>
<helper key="Agent" value="JOnAS" type="composite">
……
</helper>
……
```

Figure 10 Example of model operation transformation.

operations in the runtime model based approach, which are aimed to ensure synchronization between the architecture-based models and the underlying systems. However, the difference is small and completely acceptable for Cloud management.

Related work

There are many management systems, which are used to manage different types of Cloud resources. For instance, Eucalyptus [28] and OpenStack help administrators manage infrastructure level Cloud resources, while Tivoli [29] and Hyperic help administrators manage platform-level Cloud resources. However, most of these systems lack of efficient mechanisms to adjust or extend their management interfaces for personalized requirements.

There are some research works which try to integrate existing management functions based on service-oriented architecture. A solution to system management in a distributed environment is proposed in the work [30], which encapsulates management functions into RESTful services and makes them subscribed by administrators. In our previous work [31,32], a "Management as a Service (MaaS)" solution is proposed from the reuse point of view. However, management services are not so 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 Cloud management.

Runtime models have been widely used in different systems to support self-repair [33], dynamic adaption [34], data manipulation [35], etc. We have made lots of research in the area of model driven engineering. For a given meta-model and a given set of management interfaces, SM@RT [14,15] can automatically generate the code for mapping models to interfaces with good enough runtime performance. In addition, for the situation of incomplete formalized of modeling languages, our previous work [36] has provided an MOF meta-model extension mechanism with support for upward compatibility and automatically generates a model transformation for model integration, and the work

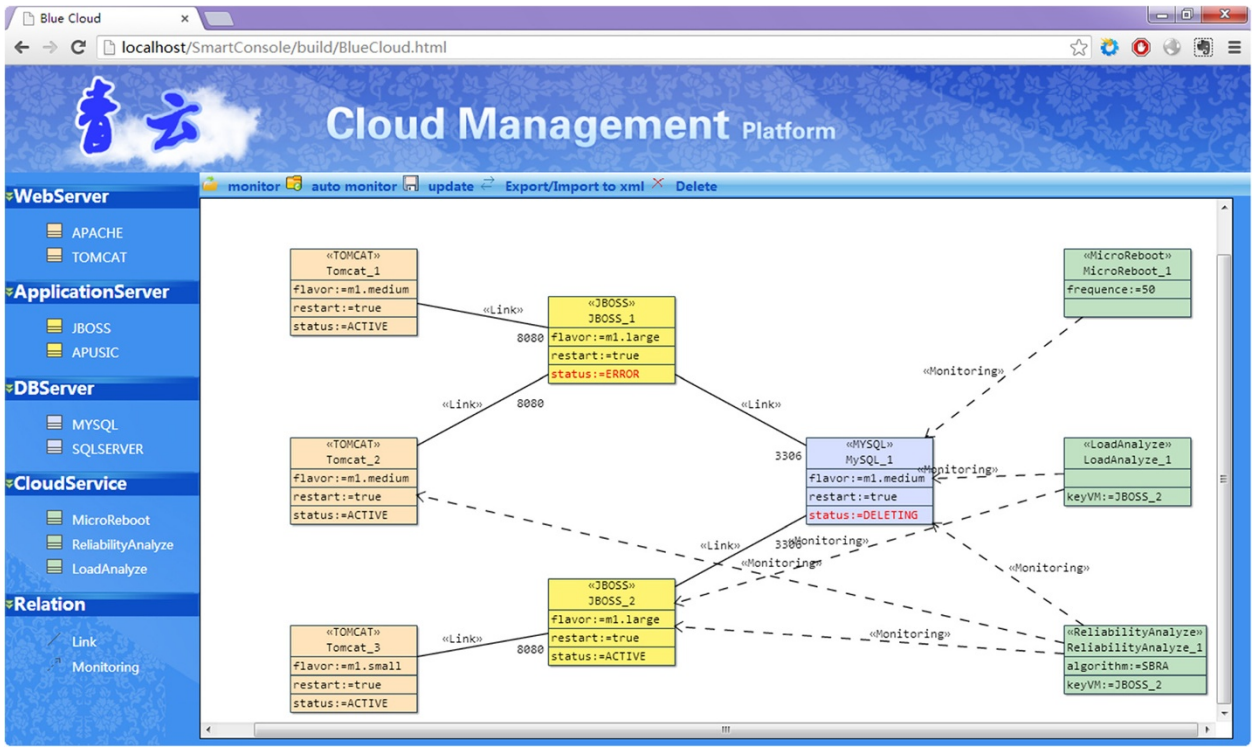

Figure 11 Architecture based integrated management of the hardware and software resources of virtual machines.

implemented on architecture-level fault tolerance [37] can also compensate for this to a degree. We have tried to construct the runtime model of a real-world Cloud and develop management programs in a modeling language [22,38]. The approach in this paper is built on our previous works. In addition, the approach is not intrusive, that is, neither instructs non-manageable systems nor extends inadequate APIs. Therefore, it is a general-purpose approach and is capable to interwork with other similar works like Pi-ADL [39].

Table 3 Comparison of performance between management interfaces and the runtime model

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

Conclusion and future work

Due to the diversity of Cloud resources and personalized management requirements, Cloud management is faced with huge challenges. To satisfy a new management requirement, the most common way is conducting redevelopment based on existing management systems. However, the difficulty and workload of redevelopment are very high. This paper proposed an architecture-based approach to integrated management of diverse Cloud resources. For a new management requirement , we construct runtime models of Cloud resources and the customized model which satisfies the requirement. The operations on the customized model are mapped to the ones on Cloud resource runtime models through model operation transformation. Thus, administrators could focus on the core of management logics and all the management tasks could be carried out through executing operating programs on the customized model, which greatly reduces the workload of hand coding.

As future work, we plan to give more support for administrators to manage Cloud resources. On one hand, we plan to perform further analysis such as model checking to ensure a deeper correctness and completeness of the generated causal link between the runtime model and underlying systems. On the other hand, we also plan to add some more advanced management functions with

the help of model techniques to ease management tasks of Cloud.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
XC and YZ carried out the Cloud management studies, participated in the study and drafted the manuscript. GH participated in the design of the study and performed the statistical analysis. XZ and CR conceived of the study, and participated in its design and coordination and helped to draft the manuscript. WG participated in paper revision and made many suggestions. All authors read and approved the final manuscript.

Acknowledgements
This work was supported by the National Basic Research Program (973) of China under Grant No. 2011CB302604, the National Natural Science Foundation of China under Grant No. 61121063, the Technology Innovation Platform Project of Fujian Province under Grant No. 2009 J1007 and the Research Program of Fuzhou University under Grant No. 022543.

Author details
[1]College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350116, China. [2]Fujian Provincial Key Laboratory of Networking Computing and Intelligent Information Processing (Fuzhou University), Fuzhou 350116, China. [3]Key Laboratory of High Confidence Software Technologies (Ministry of Education), Beijing 100871, China. [4]School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China. [5]Department of Computer Science and Electronic Engineering, University of Stavanger, Stavanger 4036, Norway.

Received: 11 November 2013 Accepted: 20 June 2014
Published: 29 July 2014

References
1. Mell P, Grance T (2009) The NIST definition of cloud computing. Special publication 800-145. U.S. Department of Commerce. In: National Institute of Standards and Technology.
2. Kotsovinos E, Stanley M (2010) Virtualization: Blessing or Curse? Managing Virtualization at a large scale is fraught with hidden challenges. Comm ACM 54(1):61–65
3. Zhang Y, Huang G, Liu X, Mei H (2010) Integrating Resource Consumption and Allocation for Infrastructure Resources on-Demand. In: Proc. of the 3rd International Conference on Cloud Computing. IEEE Computer Society, Washington, pp 75–82
4. Amazon Amazon EC2. http://aws.amazon.com/ec2/
5. Google Google App Engine. https://appengine.google.com/
6. Garlan D (2000) Software Architecture: A Roadmap. In: Proc. of the 22nd International Conference on Software Engineering, Future of Software Engineering Track. ACM, New York, pp 91–101
7. Hong M, Junrong S (2006) Progress of research on software architecture. J Software 17(6):1257–1275, in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/1257.htm
8. France R, Rumpe B (2007) Model-driven Development of Complex Software: A Research Roadmap. In: Proc. of the 29th International Conference on Software Engineering, Future of Software Engineering Trac. IEEE Computer Society, Washington, pp 37–54
9. Bencomo N, Blair G, France R (2006) Summary of the Workshop Models@run.time at MoDELS 2006. In: Lecture Notes in Computer Science, Satellite Events at the MoDELS 2006 Conference. Springer, Heidelberg, pp 226–230
10. Blair G, Bencomo N, France R (2009) Models@ run.time. Comput 42(10):22–27
11. Huang G, Mei H, Yang F (2006) Runtime recovery and manipulation of software architecture of component-based systems. Automated Software Eng 13(2):257–281
12. Occello A, AM DP, Riveill M (2008) A Runtime Model for Monitoring Software Adaptation Safety and its Concretisation as a Service. Models@ runtime 8:67–76
13. Wu Y, Huang G, Song H, Zhang Y (2012) Model driven configuration of fault tolerance solutions for component-based software system. In: Proc. of the 15th International Conference on Model Driven Engineering Languages and Systems. Springer, Heidelberg, pp 514–530
14. Huang G, Song H, Mei H (2009) SM@RT: Applying Architecture-based Runtime Management of Internetware Systems. Int J Software Informa 3(4):439–464
15. Song H, Huang G, Chauvel F, Xiong Y, Hu Z, Sun Y, Mei H (2011) Supporting Runtime Software Architecture: A Bidirectional-Transformation-Based Approach. J Syst Software 84(5):711–723
16. Peking University SM@RT: Supporting Models at Run-Time. http://code.google.com/p/smatrt/
17. Rushby JM (1995) Model Checking and Other Ways of Automating Formal Methods. In: Position paper for panel on Model Checking for Concurrent Programs. Software Quality Week, San Francisco
18. Microsoft Internet Information Services. http://www.iis.net/
19. Igor Sysoev Nginx. http://nginx.org/en/index.html
20. Song H, Xiong Y, Chauvel F, Huang G, Hu Z, Mei H (2009) Generating Synchronization Engines between Running Systems and Their Model-Based Views. In: Models in Software Engineering (the MoDELS Workshops). Springer, Heidelberg, pp 140–154
21. Song H, Huang G, Xiong Y, Chauvel F, Sun Y, Hong M (2010) Inferring Meta-Models for Runtime System Data from the Clients of Management APIs. In: Proc. of the 13rd International Conference on Model Driven Engineering Languages and Systems. Springer, Heidelberg, pp 168–182
22. Huang G, Chen X, Zhang Y, Zhang X (2012) Towards architecture-based management of platforms in the cloud. Frontiers Comput Sci 6(4):388–397
23. OpenStack The Open Source Cloud Operating System. http://www.openstack.org/
24. SpringSource Hyperic. http://www.hyperic.com/
25. Kecskemeti G, Terstyanszky G, Kacsuk P, Neméth Z (2011) An approach for virtual appliance distribution for service deployment. Future Generation Comput Syst 27(3):280–289
26. Eclipse Eclipse Modeling Framework. http://www.eclipse.org/modeling/emf/
27. Object Management Group Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT). http://www.omg.org/spec/QVT
28. Nurmi D, Wolski R, Grzegorczyk C, Obertelli G, Soman S, Youseff L, Zagorodnov Z (2009) The Eucalyptus Open-Source Cloud-Computing System. In: Proc. of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid. IEEE Computer Society, Washington, pp 124–131
29. IBM IBM Tivoli Software. http://www-01.ibm.com/software/tivoli/
30. Ludwig H, Laredo J, Bhattacharya K (2009) Rest-based management of loosely coupled services. In: Proc. of the 18th International Conference on World Wide Web. ACM, New York, pp 931–940
31. Chen X, Liu X, Zhang X, Liu Z, Huang G (2010) Service Encapsulation for Middleware Management Interfaces. In: Proc. of the 5th International Symposium on Service Oriented System Engineering. IEEE Computer Society, Washington, pp 272–279
32. Chen X, Liu X, Fang F, Zhang X, Huang G (2010) Management as a Service: An Empirical Case Study in the Internetware Cloud. In: Proc. of the 7th IEEE International Conference on E-Business Engineering. IEEE Computer Society, Washington, pp 470–473
33. Sicard S, Boyer F, de Palma N (2008) Using components for architecture-based management: the self-repair case. In: Proc. of the 30th International Conference on Software Engineering. ACM, New York, pp 101–110
34. Morin B, Barais O, Nain G, Jezequel JM (2009) Taming dynamically adaptive systems using models and aspects. In: Proc. of the 31st International Conference on Software Engineering. IEEE Computer Society, Washington, pp 122–132
35. MoDisco Project. http://www.eclipse.org/gmt/modisco/
36. Chen X, Huang G, Chauvel F, Sun Y, Mei H (2010) Integrating MOF-Compliant Analysis Results. Int J Software Informat 4(4):383–400
37. Junguo L, Xiangping C, Gang H, Hong M, Franck C (2009) Selecting Fault Tolerant Styles for Third-Party Components with Model Checking Support. In: Proc. of the 12th International Symposium on Component-Based Software Engineering. Springer, Heidelberg, pp 69–86
38. Xiaodong Z, Xing C, Ying Z, Yihan W, Wei Y, Gang H, Qiang L (2013) Runtime Model Based Management of Diverse Cloud Resources. In: Proc. of the 16th International Conference on Model Driven Engineering Languages and Systems. Springer, Heidelberg, pp 572–588
39. Flavio O (2008) Dynamic Software Architectures: Formally Modelling Structure and Behaviour with Pi-ADL. In: Proc. Of the 3rd International Conference on Software Engineering Advances. IEEE Computer Society, Washington, pp 352–359

doi:10.1186/s13677-014-0011-7
Cite this article as: Chen *et al.*: **Architecture-based integrated management of diverse cloud resources.** *Journal of Cloud Computing: Advances, Systems and Applications* 2014 **3**:11.