图 7 中,Project 表示项目,描述了项目的基本信息,包含计算配置方案列表 ServiceOfferings、磁盘配置方案列表 DiskOfferings、虚拟机映像文件列表 Templates、虚拟机列表 VirtualMachines、网络子网列表 GuestNetworks 和磁盘卷列表 Volumes.Template 表示项目可以使用的虚拟机映像文件,描述其操作系统信息.ServiceOffering 表示虚拟机计算配置方案,描述其 CPU 核数、内存大小等计算资源配置信息.DiskOffering 表示虚拟机磁盘配置方案,描述其磁盘大小等存储资源配置信息.GuestNetwork 表示网络子网,描述该子网的基本信息,并包含该子网拥有的公共 IP 地址的列表 PublicIps;PublicIp 表示可供外部网络访问的 Ip 地址,为子网中拥有该 IP 地址的虚拟机提供外部网络的访问入口.Volume 表示磁盘卷,可为虚拟机增加额外的存储.VirtualMachine 表示虚拟机,描述虚拟机使用的映像文件、计算及存储配置方案等基本信息.同时,VirtualMachine 还包含虚拟机使用的虚拟网卡列表 Nics;Nic 表示虚拟网卡,描述其 ip 地址、mac 地址及所在的网络子网等网络配置信息.

Amazon EC2(Amazon Elastic Compute Cloud)公有云服务允许使用者按需租用虚拟机来搭建应用系统.图 7 所示模型图(下)描述了 Amazon EC2 软件体系结构模型(系统元模型)中的主要受管单元,包括资源配置类型(InstanceType)、虚拟机映像文件(Image)、虚拟机(Instance)、网络子网(Subnet)及磁盘卷(Volume)等.其中,EC2Client 是模型的根元素,包含资源配置类型列表 InstanceTypes、虚拟机映像列表 Images、虚拟机列表 Instances、网络子网列表 Subnets 及磁盘卷列表 Volumes.InstanceType 表示虚拟机资源配置类型,描述了 CPU 核数、内存大小、存储空间等资源配置信息.Image 表示虚拟机映像文件,是虚拟机软件系统的载体.Subnet 表示某个网络子网,为虚拟机提供基本的网络服务.Volume 表示可定制的用于持久性数据存储的磁盘卷,可为虚拟机增加额外的存储空间.Instance 表示虚拟机,描述了虚拟机使用的映像文件、资源配置类型等基本信息.同时,Instance 包含其使用的虚拟网卡列表 Nics;Nic 表示虚拟网卡,描述虚拟网卡的 ip 地址、mac 地址及所在的网络子网等网络配置信息.

给定 CloudStack 和 Amazon EC2 系统元模型,仍需定义其上的模型操作[21],即访问模型.CloudStack 和 Amazon EC2 拥有大量的管理接口,我们通过定义模型操作到这些管理接口的映射规则,来对它们建模.由于篇幅限制,访问模型的具体构建过程请查阅我们之前的工作[15,16,19].在系统元模型和访问模型的基础上,SM@RT 工具能够自动生成模型转换程序,以保障系统运行时模型与运行系统的双向同步.于是,能够在模型层对 CloudStack 和 Amazon EC2 进行使用和管理.

5.2 统一模型到云平台运行时模型的映射

如图 3 所示,统一模型(使用者模型)为云资源的使用提供了统一的全局视图.为了进一步使使用者能够通过统一模型对基于 CloudStack 的企业私有云及 Amazon EC2 公有云服务的计算、存储资源进行操作,需要实现统一模型到 CloudStack 和 Amazon EC2 运行时模型的映射.根据模型间的元素映射关系,我们定义了统一模型到云平台运行时模型的映射规则.其中,云平台虚拟机及网卡在模型间的映射关系是映射规则描述的关键点.下面以统一模型中 Server 到 CloudStack 运行时模型中 VirtualMachine 的“一对一”映射,以及统一模型中 Nic 和 PublicIp 到 CloudStack 运行时模型中 Nic 的“多对一”映射为例,详细介绍映射规则的描述方式.

1) 统一模型中 Server 与 CloudStack 运行时模型中 VirtualMachine 均表示虚拟机,它们存在“一对一”映射关系,图 7 所示映射规则描述片段中通过 helper 标签表示两个元素间的映射关系.Server 中 id、name、imageId、cpuNumber、cpuSpeed、cpuUsed 及 diskSize 等属性与 VirtualMachine 中 id、name、templateId、cpuNumber、cpuSpeed、cpuUsed 及 diskSize 等属性存在对应关系,描述片段中通过 mapper 标签表示两两属性间的映射关系.其中,当 mapper 标签的 type 为“basic”时,表示对应属性的属性值相等.例如,当 Server 的属性 cpuNumber 值为 3 时,VirtualMachine 对应的属性 cpuNumber 值也为 3.而当 mapper 标签的 type 为“advanced”时,表示对应属性的属性值需要进行转换.例如,当 Server 的属性 serverTypeId 值为“small”时,VirtualMachine 对应的属性 serviceOfferingId 值为“e3770982-e2f2-4349-b1c0-2d7dfb6df0fe”.

2) 统一模型中 Nic 表示虚拟网卡,描述了虚拟机所在网络子网、子网 IP 地址等网络配置信息,PublicIp 表示可供外部网络访问的 IP 地址,并描述了关联的 Nic 信息;而 CloudStack 运行时模型中 Nic 则描述了虚拟机所在网络子网、子网 IP 地址、外网 IP 地址等全部网络配置信息.因此,统一模型中 Nic 和 PublicIp 到 CloudStack

运行时模型中 Nic 是“多对一”映射关系.在图 7 所示的映射规则描述片段中,通过 helper 标签表示统一模型中 Nic 到 CloudStack 运行时模型中 Nic 的映射关系,通过 mapper 标签表示两两属性间的映射关系.特别地,通过 query 标签表示统一模型中与 Nic 关联的 PublicIp 属性 publicIPAddress 到 CloudStack 运行时模型中 Nic 属性 elasticIPAddress 的映射.

根据映射规则,作用在统一模型上的模型操作将转换为作用在 CloudStack 或 Amazon EC2 运行时模型运行时模型上对应的模型操作.图 8 展示了统一模型上的虚拟机创建操作转换为 CloudStack 运行时模型上对应的模型操作并加以执行的过程,虚拟机创建操作描述如下.(1) Query:查询一个 Servers 元素,其 projectId 为“2edc7933-0a09-46eb”.(2) Add:创建一个 Server 元素.(3) Set:为 Server 元素的属性赋值.

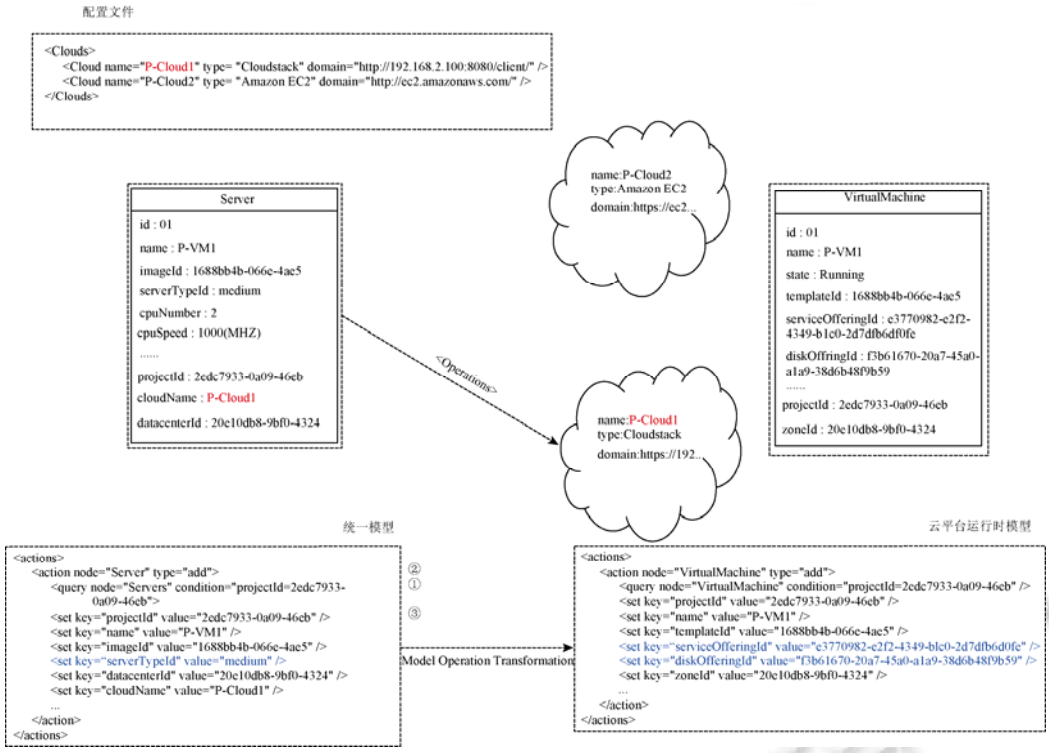

Fig.8 An example of model operation transformation

图 8 模型操作转换实例

统一模型中待创建的 Server 元素的 cloudName 属性值为“P-Cloud1”,根据配置文件可知,目标云平台类型是 CloudStack.因此,Server 创建任务将转换为 CloudStack 运行时模型上的 VirtualMachine 创建任务.根据映射规则,模型操作会逐条进行转换:统一模型中 Servers 元素的查询操作将转换为 CloudStack 运行时模型中 VirtualMachines 元素的查询操作,其 projectId 为“2edc7933-0a09-46eb”;统一模型中 Server 元素的创建操作将转换为 CloudStack 运行时模型中 VirtualMachine 元素的创建操作;统一模型中 Servers 元素属性的赋值操作将转换为 CloudStack 运行时模型中 VirtualMachine 元素对应属性的赋值操作.特别地,赋值操作的转换需要遵循属性值的映射规则.例如,统一模型中 Server 元素 serverTypeId 属性的值为 medium,那么,CloudStack 运行时模型中 VirtualMachine 元素 serviceOfferingId 属性的值为 e3770982-e2f2-4349-b1c0-2d7dfb6df0fe,diskOfferingId 属性的值为 f3b61670-20a7-45a0-a1a9-38d6b48f9b59.最后,生成的模型操作文件将传送到 CloudStack 运行时模型并执行,以实现预期的管理效果.

5.3 方法评估

我们从以下 3 方面对方法进行评估.

5.3.1 混合云运行时软件体系结构模型的构造工作评估

在单一云平台运行时模型的构造过程中,开发人员仅需使用 SM@RT 建模语言定义系统元模型和系统访问模型,SM@RT 代码生成器就能够自动生成同步引擎,支持运行时模型与运行系统的双向同步,实现在模型层对单一云平台进行管理.在前期工作中,我们研究了系统元模型的推理方法[18],通过分析调用管理 API 的客户端代码,实现系统元模型的自动构造,该工作能够进一步降低单一云平台运行时模型的构造难度.此外,单一云平台运行时模型的构造工作是一次性的,它与管理 API 一样可以在不同的云管场景中复用.因此,从管理功能复用的角度,构造单一云平台运行时模型的额外工作是可以接受的.

在单一云平台运行时模型的基础上,开发人员仅需定义统一模型到云平台运行时模型的映射规则,就能够面向统一模型进行混合云管理程序的开发.统一模型描述了常见云平台共有的资源类型及管理操作,能够对计算、存储、网络等资源进行统一管理.为了验证统一模型对云平台管理功能的覆盖程度,我们将统一模型提供的管理功能与多个云平台的资源管理 API 进行比较.如图 9 所示,云平台共有的资源管理 API 可以覆盖各云平台的核心管理功能,且其数量均超过各云平台资源管理 API 的 70%,满足对云平台进行统一管理的需求.在前期工作中,我们研究了基于模型的多样化云资源集成管理方法[20],通过模型转换,实现单一云平台中系统管理功能的复用和集成,该工作支持云平台个性化管理功能的集成,能够进一步增强统一模型在不同管理场景中的适应能力.因此,从管理程序开发的角度,混合云运行时模型的构造代价是可以接受的,其适用程度是满足要求的.

云平台共有资源管理API示例			
Compute	Server	CreateServer	创建虚拟机
		DestroyServer	销毁虚拟机
		UpdateServer	修改虚拟机
		ListServers	查询虚拟机
		StartServer	启动虚拟机
		StopServer	关闭虚拟机
		RebootServer	重启虚拟机
		MigrateServer	热迁移虚拟机
		ResizeServer	更改虚拟机硬件配置
	Image	CreateImage	创建虚拟机模板映像
		DeleteImage	删除虚拟机模板映像
		UpdateImage	修改虚拟机模板映像
		ListImages	查询虚拟机模板映像
	ServerType	ListServerTypes	查询虚拟机硬件配置方案
Storage	Volume	CreateVolume	创建磁盘卷
		DeleteVolume	删除磁盘卷
		ListVolumes	查询磁盘卷
		AttachVolume	附加磁盘卷到某个虚拟机
		DetachVolume	从某个虚拟机卸载磁盘卷
Network	Nic	AddNicToServer	添加网卡到某个虚拟机
		RemoveNicFromServer	从某个虚拟机卸载网卡
		ListNics	查询虚拟网卡
	Network	CreateNetwork	创建虚拟网络
		DeleteNetwork	删除虚拟网络
		UpdateNetwork	修改虚拟网络
		ListNetworks	查询虚拟网络
	PublicIp	ListPublicIps	查询公共IP
		AssociatePublicIpWithServer	绑定公共IP到某个虚拟机
		DisassociatePublicIpFromServer	从某个虚拟机撤销公共IP

云平台共有资源管理API的覆盖程度			
	OpenStack	CloudStack	Amazon EC2
Compute	81.50%	84.20%	73.90%
Storage	70%	71.40%	75%
Network	92.90%	76.90%	79.20%

Fig.9 Statistics of common APIs for cloud resource management

图 9 云平台共有资源管理 API 统计

5.3.2 基于模型语言与通用语言的管理程序开发难度比较

为了验证本文方法,我们针对一组常见的混合云管理任务,基于模型语言 QVT[21]和通用语言 Java 分别实现了其管理程序.图 10 展示了混合云中虚拟机 CPU 负载报警任务的 QVT 和 Java 管理程序,其中,Java 程序需要 200 多行,而 QVT 程序仅需要 3 行,与 Java 程序相比,QVT 程序的难度和复杂度都要小得多.一方面,混合云软件体系结构模型对云平台管理接口进行复用,开发人员不用处理管理接口调用及底层数据交互等编程工作;另一

方面,模型语言提供了一些模型层的复杂操作,例如,“select”用于选出符合某种条件的所有模型元素,这些复杂操作进一步降低了编程难度和复杂度.表 2 对完成相同一组混合云管理任务的 QVT 和 Java 程序进行比较.任务 1 是列出混合云中的所有虚拟机,Java 代码行数为 205 行,而 QVT 代码行数为 3 行;任务 2 是根据混合云负载,选择一个云平台创建虚拟机,Java 代码行数为 223 行,而 QVT 代码行数为 17 行;任务 3 是混合云中虚拟机 CPU 负载查询,Java 代码行数为 218 行,而 QVT 代码行数为 3 行;任务 4 与任务 5 分别是混合云中所有虚拟机磁盘使用量和网络使用量的计算.同样地,QVT 程序的代码行数也远小于 Java 程序.

Fig.10 Programs of VM monitoring in the languages of QVT and Java

图 10 分别用 QVT 与 Java 实现的虚拟机监测程序

Table 2 Comparison of LOC between the QVT and Java programs

表 2 QVT 与 Java 程序开发难度比较

管理任务		虚拟机的展示	虚拟机的创建	虚拟机 CPU 负载报警	虚拟机磁盘使用量计算	虚拟机网络使用量计算
代码量(行)	QVT	3	17	3	13	13
	Java	205	223	218	241	241

5.3.3 基于运行时模型与管理接口的管理程序执行性能比较

为了比较执行性能,我们在虚拟机使用数量为 10 台、20 台和 50 台的情况下,分别执行 QVT 和 Java 管理程序,完成相同一组混合云管理任务.实验过程中,虚拟机在 CloudStack 私有云和 Amazon EC2 公有云中占比分别为 80%和 20%.如表 3 所示,QVT 程序的执行时间均会略高于 Java 程序.任务 1、任务 2 均未对运行的虚拟机进行操作,其管理接口的调用次数不随虚拟机数量的增长而发生变化,它们的执行时间也基本不变.任务 3~任务 5 均需要获取每一台运行虚拟机的属性值,其执行时间与虚拟机数量呈线性增长,QVT 与 Java 程序的执行时间差也随之增大.其主要原因是,QVT 和 Java 程序从本质上均是通过调用云平台管理接口来实现特定的管理功能;而模型方法还需要额外的操作来维护统一模型与运行时模型,以及运行时模型与底层系统间的同步机制.因此,QVT 和 Java 程序执行时间的差异与管理接口的调用次数呈线性增长.然而,与管理程序的执行时间相比,它们的差异并不大.特别地,从系统管理的角度看,这种性能上的差异是可被接受的.

Table 3 Comparison of performance between the QVT and Java programs

表 3 QVT 与 Java 执行性能比较

	管理任务	虚拟机的展示	虚拟机的创建	虚拟机 CPU 负载报警	虚拟机磁盘使用量计算	虚拟机网络使用量计算
10 台	QVT 执行时间(ms)	117	118	1 148	1 183	1 098
	Java 执行时间(ms)	74	86	749	738	741
	API 调用次数	1	1	10	10	10
20 台	QVT 执行时间(ms)	160	128	2 386	2 413	2 395
	Java 执行时间(ms)	113	79	1 526	1 493	1 511
	API 调用次数	1	1	20	20	20

Table 3 Comparison of performance between the QVT and Java programs (Continued)
表 3 QVT 与 Java 执行性能比较(续)

	管理任务	虚拟机的展示	虚拟机的创建	虚拟机 CPU 负载报警	虚拟机磁盘使用量计算	虚拟机网络使用量计算
50 台	QVT 执行时间(ms)	247	131	5 876	6 129	5 812
	Java 执行时间(ms)	202	81	3 826	3 784	3 816
	API 调用次数	1	1	50	50	50

6 相关工作

目前存在许多云平台管理工具用于不同类型云资源的管理,如,OpenStack[4]、Eucalyptus[5]用于管理基础设施层的云资源,Tivoli[22]、Hyperic[23]用于管理平台层的云资源.然而,这些管理工具缺乏完善的混合云支撑机制.

近年来,存在许多混合云管理的研究工作,包括混合云构造[24−26]、混合云应用部署[27,28]及混合云资源调度[29−31].文献[24]提出一种混合云虚拟化基础设施的管理方法,通过 OpenNebula[32]实现单个云平台虚拟化基础设施的管理及外部云平台核心管理 API 的集成,并提供了一套混合云虚拟化资源的调度机制.文献[25]提出一种 IaaS(infrastructure as a service)云服务的抽象模型及一组核心管理 API 的抽象接口,并在多云环境中构造了统一管理实例,文献[26]在此基础上进一步实现了一组高级管理功能.上述工作对混合云构造方法进行了初步探索,然而,方法中的统一管理接口是通过直接封装云平台核心管理 API 获得,因此,其工作量大且可扩展性差.文献[27,28]在混合云虚拟化资源管理的基础上,增加了应用部署和配置功能,实现了混合云中的应用自动部署.文献[29−31]在混合云虚拟化资源管理的基础上,面向特定应用场景研究调度策略,实现了混合云中的资源管理.本文在常见云平台共有的资源类型及管理操作的基础上,提出一种 IaaS 云服务的抽象模型,通过模型转换实现多个云平台核心管理功能的快速集成,其工作量小;基于前期工作[20],还能够支持云平台个性化管理功能的集成,可扩展性好.此外,本文方法为混合云应用部署及资源调度提供了虚拟化资源的统一管理能力,而以模型为中心的分析方法与支撑机制[33]能够进一步为混合云管理程序的开发提供帮助.

运行时模型被广泛应用在不同类型的软件系统中,以支持数据操作[34]、系统自修复[35]和动态自适应[36−39]等管理功能.前期工作中,我们在运行时模型理论及构造方法方面进行了研究:给定系统元模型与一组管理接口,SM@RT 工具[15]就能自动生成代码,在保证性能的前提下实现模型到管理接口的映射;当系统元模型发生变化时,SM@RT 可以自动生成新的映射代码;文献[16]对以上内容进行了详细的论述.我们还研究了系统元模型的推理方法,通过分析调用管理 API 的客户端代码,实现系统元模型的自动构造[18].同时,为了弥补建模语言本身的非完全形式化问题,我们在模型分析及模型容错方面也进行了研究:提出一种 MOF 元模型扩展机制[40]以支持元模型的向上兼容,从而实现在模型集成过程中模型的自动转换.该方法在体系结构级别的系统容错实践[41]中进一步得到了验证.我们还构建了云平台运行时体系结构模型[19],对运行时模型的性能进行了验证,并尝试基于模型语言实现系统自适应管理.进一步地,提出一种基于模型的多样化云资源集成管理方法[20],通过模型转换实现单一云平台中系统管理功能的复用和集成.本文方法建立在以上前期工作的基础上.

7 结束语

云平台管理接口和管理机制的异构性,给混合云管理系统的开发带来了极大的难度和复杂度.本文提出一种基于运行时模型的混合云管理方法:开发人员仅需定义统一模型与云平台运行时模型间的元素映射关系,任何统一模型上的管理操作就能够自动转换为云平台运行时模型上对应的管理操作,并最终作用到云计算系统上.于是,开发人员能够面向统一模型进行管理程序的开发,而不用处理管理接口调用及底层数据交互等繁杂、琐碎的编程工作.本文方法能够降低混合云管理系统开发的难度和复杂度.

未来工作的重点主要包含两个方面:一方面,将方法运用到遗产系统整合和动态资源扩展等混合云实际管理场景中,并完善特定场景下的支撑机制;另一方面,在方法基础上进行管理风格的研究,基于模型分析、推理等技术实现系统容错、安全监控等高级管理功能.

References:

[1] Mell P, Grance T. The NIST definition of cloud computing. Communications of the ACM, 2011,53(6):50.

[2] Amazon Web Services (AWS). Cloud Computing Services. 2017. http://aws.amazon.com/

[3] Kumar R, Jain K, Maharwal H, Jain N, Dadhich A. Apache CloudStack: Open source infrastructure as a service cloud computing platform. Int'l Journal of Advancement in Engineering Technology, Management and Applied Science, 2014,681−684. [doi: 10.13140/2.1.1290.0483]

[4] OpenStack. The open source cloud operating system. 2013. http://openstack.org/projects/

[5] Nurmi D, Wolski R, Grzegorczyk C, Obertelli G, Soman S, Youseff L, Zagorodnov Z. The eucalyptus open-source cloud-computing system. In: Proc, of the 9th IEEE/ACM Int'l Symp. on Cluster Computing and the Grid. Shanghai: IEEE Press, 2009. 124−131. [doi: 10.1109/CCGRID.2009.93]

[6] Garlan D. Software architecture: A roadmap. In: Proc. of the 22nd Int'l Conf. on Software Engineering, Future of Software Engineering Track. New York: ACM Press, 2000. 91−101. [doi: 10.1145/336512.336537]

[7] Mei H, Shen JR. Progress of research on software architecture. Ruan Jian Xue Bao/Journal of Software, 2006,17(6):1257−1275 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/1257.htm [doi: 10.1360/jos171257]

[8] France R, Rumpe B. Model-Driven development of complex software: A research roadmap. In: Proc. of the 29th Int'l Conf. on Software Engineering. Minneapolis: IEEE Press, 2007. 37−54. [doi: 10.1109/FOSE.2007.14]

[9] Huang G, Ma XX, Tsai WT. A new software paradigm for Internet computing. National Science Review, 2014,1(2):168−169. [doi: 10.1093/nsr/nwt014]

[10] Bencomo N, Blair G, France R. Summary of the workshop models@run.time at MoDELS 2006. Models in Software Engineering, 2006,4364:227−231. [doi: 10.1007/978-3-540-69489-2_28]

[11] Blair G, Bencomo N, France R. Models@ run.time. Computer, 2009,42(10):22−27. [doi: 10.1109/MC.2009.326]

[12] Huang G, Mei H, Yang FQ. Runtime recovery and manipulation of software architecture of component-based systems. Automated Software Engineering, 2006,13(2):257−281. [doi: 10.1007/s10515-006-7738-4]

[13] Occello A, Dery-Pinna A, Riveill M. A runtime model for monitoring software adaptation safety and its concretisation as a service. In: Models@ runtime, Toulouse, 2008,8:67−76.

[14] Wu YH, Huang G, Song H, Zhang Y. Model driven configuration of fault tolerance solutions for component-based software system. In: Proc. of the 15th Int'l Conf. on Model Driven Engineering Languages and Systems. Innsbruck: Springer-Verlag, 2012. 514−530. [doi: 10.1007/978-3-642-33666-9_33]

[15] Huang G, Song H, Mei H. SM@RT: Applying architecture-based runtime management of internetware systems. Int'l Journal of Software and Informatics, 2009,3(4):439−464. [doi: 10.1145/1640206.1640215]

[16] Song H, Huang G, Chauvel F, Xiong YF, Hu ZJ, Sun YC, Mei H. Supporting runtime software architecture: A bidirectional-transformation-based approach. Journal of Systems and Software, 2011,84(5):711−723. [doi: 10.1016/j.jss.2010.12.009]

[17] Song H, Xiong YF, Chauvel F, Huang G, Hu ZJ, Mei H. Generating synchronization engines between running systems and their model-based views. In: Models in Software Engineering (the MoDELS Workshops). Denver: Springer-Verlag, 2009. 140−154. [doi: 10.1007/978-3-642-12261-3_14]

[18] Song H, Huang G, Xiong YF, Chauvel F, Sun YC, Mei H. Inferring meta-models for runtime system data from the clients of management APIs. In: Proc. of the 13rd Int'l Conf. on Model Driven Engineering Languages and Systems. Oslo: Springer-Verlag, 2010. 168−182. [doi: 10.1007/978-3-642-16129-2_13]

[19] Huang G, Chen X, Zhang Y, Zhang XD. Towards architecture-based management of platforms in the cloud. Frontiers of Computer Science, 2012,6(4):388−397. [doi: 10.1007/s11704-012-2100-4]

[20] Chen X, Zhang Y, Huang G, Zheng XH, Guo WZ, Rong CM. Architecture-Based integrated management of diverse cloud resources. Journal of Cloud Computing: Advances, Systems and Applications, 2014,3:11. [doi: 10.1186/s13677-014-0011-7]

[21] Object Management Group. Meta object facility (MOF) 2.0 query/view/transformation (QVT). 2011. http://www.omg.org/spec/QVT

[22] IBM. IBM Tivoli Software. 2011. http://www-01.ibm.com/software/tivoli/

[23] SpringSource. Hyperic. 2013. http://www.hyperic.com/

[24] Sotomayor B, Montero RS, Llorente IM, Foster I. Virtual infrastructure management in private and hybrid clouds. IEEE Network Computing, 2009,13(5):14−22. [doi: 10.1109/MIC.2009.119]

[25] Lee BS, Yan SX, Ma D, Zhao GP. Aggregating IaaS service. In: Proc. of the 2011 Annual SRII Global Conf. San Jose: IEEE Computer Society Press, 2011,14:335−338. [doi: 10.1109/SRII.2011.44]

[26] Yan SX, Lee BS, Zhao GP, Ma D, Mohamed P. Infrastructure management of hybrid cloud for enterprise users. In: Proc. of the 5th Int'l DMTF Academic Alliance Workshop on Systems and Virtualization Management (SVM). Paris: IEEE Press, 2011. 1−6. [doi: 10.1109/SVM.2011.6096463]

[27] Suzuki J, H.Phan D, Higuchi M, Yamano Y, Oba K. Model-Driven integration for a service placement optimizer in a sustainable cloud of clouds. In: Proc. of the Joint, Int'l Conf. on Soft Computing and Intelligent Systems. Kobe: IEEE Press, 2012. 301–306. [doi: 10.1109/SCIS-ISIS.2012.6505344]

[28] Juve G, Deelman E. Automating application deployment in infrastructure clouds. In: Proc. of the 3rd IEEE Int'l Conf. on Coud Computing Technology and Science. Athens: IEEE Computer Society Press, 2011. 658–665. [doi: 10.1109/CloudCom.2011.102]

[29] Yue JY, Zhang Z, Fu JH, Lu SQ, Li XM, Shen YF. Extensible architecture for high-throughput task processing based on hybrid cloud infrastructure. In: Proc. of the 2011 Int'l Conf. on Electronics, Communications and Control (ICECC). IEEE Press, 2011. 1452–1455. [doi: 10.1109/ICECC.2011.6067604]

[30] Ruben VDB, Vanmechelen K, Broeckhove J. Cost-Optimal scheduling in hybrid IaaS clouds for deadline constrained workloads. In: Proc. of the 3rd IEEE Int'l Conf. on Cloud Computing (CLOUD). Miami: IEEE, 2010. 228–235. [doi: 10.1109/CLOUD.2010.58]

[31] Wang WJ, Chang YS, Lo WT, Lee YK. Adaptive scheduling for parallel tasks with QoS satisfaction for hybrid cloud environments. Journal of Supercomputing, 2013,66(2):783–811. [doi: 10.1007/s11227-013-0890-2]

[32] Fontán J, Vázquez T, Gonzalez L, *et al*. OpenNEbula: The open source virtual machine manager for cluster computing. In: Proc. of the Open Source Grid and Cluster Software Conf. San Francisco, 2008.

[33] Rushby J. Model Checking and Other Ways of Automating Formal Methods. Software Quality Week, 1995. 1–12.

[34] Bruneliere H, Cabot J, Jouault F, Madiot F. MoDisco: A generic and extensible framework for model driven reverse engineering. In: Proc. of the 25th IEEE/ACM Int'l Conf. on Automated Software Engineering. Antwerp: ACM Press, 2010. 173–174. [doi: 10.1145/1858996.1859032]

[35] Sicard S, Boyer F, Palma ND. Using components for architecture-based management: The self-repair case. In: Proc. of the 30th Int'l Conf. on Software Engineering. New York: ACM Press, 2008. 101–110. [doi: 10.1145/1368088.1368103]

[36] Morin B, Barais O, Nain G, Jezequel J. Taming dynamically adaptive systems using models and aspects. In: Proc. of the 31st Int'l Conf. on Software Engineering. Washington: IEEE Computer Society Press, 2009. 122–132. [doi: 10.1109/ICSE.2009.5070514]

[37] Li Y, Sun KW, Yang J, Liu TC, Zeng LZ. Model-Based system configuration approach for internetware. Science China Information Sciences, 2013,56(8):1–20. [doi: 10.1007/s11432-013-4917-3]

[38] Chen XP, Huang G, Chauvel F, Sun YC, Mei H. Integrating MOF-compliant analysis results. Int'l Journal of Software and Informatics, 2010,4(4):383–400.

[39] Li JG, Chen XP, Huang G, Mei H, Chauvel F. Selecting fault tolerant styles for third-party components with model checking support. In: Proc. of the 12th Int'l Symp. on Component-Based Software Engineering. East Stroudsburg: Berlin, Heidelberg: Springer-Verlag, 2009. 69–86. [doi: 10.1007/978-3-642-02414-6_5]

附中文参考文献:

[7] 梅宏,申峻嵘.软件体系结构研究进展.软件学报,2006,17(6):1257–1275. http://www.jos.org.cn/1000-9825/17/1257.htm [doi: 10.1360/jos171257]

陈星(1985－),男,福建永春人,博士,副教授,CCF 专业会员,主要研究领域为分布式系统,软件中间件,软件工程.

兰兴土(1992－),男,学士,主要研究领域为分布式系统,软件中间件,软件工程.

李隘鹏(1992－),男,学士,主要研究领域为分布式系统,软件中间件,软件工程.

郭文忠(1979－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为计算智能及其在计算机网络中的应用研究.

黄罡(1975－),男,博士,教授,博士生导师,CCF 专业会员,主要研究领域为分布式系统,软件中间件,软件工程.