Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems

Nicolas Ferry, Alessandro Rossini, Franck Chauvel, Brice Morin, Arnor Solberg
Department of Networked Systems and Services
SINTEF, Oslo, Norway
{name.surname}@sintef.no

***Abstract*—In the landscape of cloud computing, the competition between providers has led to an ever growing number of cloud solutions offered to consumers. The ability to run and manage multi-cloud systems (*i.e.*, applications on multiple clouds) allows exploiting the peculiarities of each cloud solution and hence optimising the performance, availability, and cost of the applications. However, these cloud solutions are typically heterogeneous and the provided features are often incompatible. This diversity hinders the proper exploitation of the full potential of cloud computing, since it prevents interoperability and promotes vendor lock-in, as well as it increases the complexity of development and administration of multi-cloud systems. This problem needs to be addressed promptly. In this paper, we provide a classification of the state-of-the-art of cloud solutions, and argue for the need for model-driven engineering techniques and methods facilitating the specification of provisioning, deployment, monitoring, and adaptation concerns of multi-cloud systems at design-time and their enactment at run-time.**

***Keywords*—cloud computing, provisioning, deployment, monitoring, adaptation, multi-cloud, model-driven engineering, domain-specific modelling language, models@run-time, CloudML**

I. Introduction

Cloud computing is a computing model enabling ubiquitous network access to a shared and virtualised pool of computing capabilities (*e.g.*, network, storage, processing, and memory) that can be rapidly provisioned with minimal management effort [1]. The landscape of cloud computing encompasses a multitude of cloud providers, as well as several infrastructure-as-a-service (IaaS) [1] and platform-as-a-service (PaaS) [1] solutions. The ability to run and manage multi-cloud systems (*i.e.*, applications targeting multiple private, public, or hybrid clouds) allows exploiting the peculiarities of each cloud solution and hence optimising performance, availability, and cost of the applications. However, these cloud solutions are typically heterogeneous and the provided features are often incompatible. This diversity hinders the proper exploitation of the full potential of cloud computing, since it prevents interoperability and promotes vendor lock-in, as well as it increases the complexity of development and administration of multi-cloud systems. This challenge needs to be addressed promptly.

There are several projects that aim at addressing this challenge by providing solutions for provisioning, deployment, monitoring and adaptation of cloud systems. The results from these projects are paramountly important to promote interoperability and prevent vendor lock-in, but they are not sufficient to properly manage the complexity of development and administration of multi-cloud systems [2], [3].

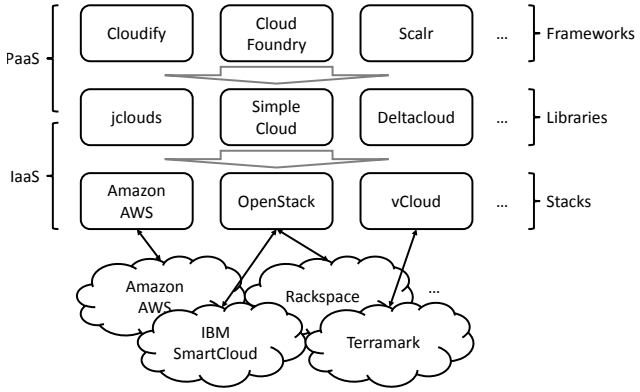

Figure 1. The stack of cloud solutions

In this paper, we provide a classification of the state-of-the-art of cloud solutions, and discuss why these solutions are not sufficient to properly manage the complexity of development and administration of multi-cloud systems. Then, we argue that the application of advanced model-driven engineering (MDE) methods and techniques would be appropriate to tame this complexity, *i.e.*, enabling the specification of provisioning, deployment, monitoring, and adaptation concerns at design-time and their automatic enactment at run-time. Finally, we present our vision for advancing the software engineering of multi-cloud systems, as well as our realisation of this vision which we call Cloud Modelling Language (CloudML) [2].

The remainder of the paper is organised as follows. In Section II, we outline a classification of the state-of-the-art of cloud stacks, libraries, and frameworks. In Section III, we present CloudML, a domain-specific modelling language along with a run-time environment that facilitate the specification of provisioning, deployment, monitoring, and adaptation concerns of multi-cloud systems at design-time and their enactment at run-time.

II. Classification of cloud solutions

The cloud market counts numerous cloud solutions at different levels of the cloud stack, such as IaaS providers, IaaS/PaaS libraries, as well as PaaS frameworks. As mentioned, this diversity prevents interoperability and promotes vendor lock-in. In the following, we present each of these solutions and explain how they build upon each other to form a stack (see Figure 1).

A. Providers

The cloud computing market counts numerous providers. The literature encompasses several taxonomies and surveys of providers [4], [5], but the cloud computing market has been constantly evolving during the latest years, and the data collected just few years ago is already outdated.

Table I shows a classification of the current major public IaaS providers. This classification is based on headquarters, data centres' location, and uptime service level agreement (SLA). Note that the list of providers is by no means exhaustive, but it includes those that we believe are the current major players at least in the European and North American markets.

The headquarters column shows that 15 providers are based in the USA while only six are based in the Europe. However, the data centres' location column shows that 17 providers have data centres in the USA while 16 have data centres in Europe. This information is particularly relevant with respect to data protection laws and regulations, such as the EU data protection directive (Directive 95/46/EC) and the upcoming data protection regulation (to be adopted in 2014), which restrict the geographical locations where the data of EU residents can be stored and processed.

The uptime SLAs column shows that all the providers promise at least 99.9% uptime. This indicates that the difference in terms of uptime SLAs across providers is not significant. However, the uptime SLA does not reflect the actual uptime, but rather a contract between the provider and the clients, and the latest years have witnessed several severe outages at major providers [6]. The interested reader may refer to the CloudSleuth's Global Provider View [7] to have an intuition of the reliability of the current major providers.

Public providers have traditionally been offering a set of proprietary APIs for the provisioning, deployment, monitoring, and (partially) adaptation of cloud capabilities. Some minor providers have been implementing APIs which are compatible with the ones from leading providers such as the Amazon AWS [8] APIs. This solution may increase the interoperability across some providers. However, it does not solve the vendor lock-in problem.

B. Stacks

A first step towards solving this problem is provided by IaaS stacks, such as OpenStack [9] or VMWare vCloud [10], for creating and managing infrastructure cloud services in private, public, and hybrid clouds.

Table II shows a classification of these stacks. This classification is based on the license, implementation languages, hypervisors supported, main contributors, and adopters of each stack.

Apache CloudStack [11] is free software included in the Apache Incubator project since 2012. It was originally developed by Citrix and is currently maintained by the Apache Software Foundation. CloudStack provides features such as resource management, user management, API, and graphical user interface (GUI).

Eucalyptus [12] is a free software project initiated in 2008. It is developed and maintained by Eucalyputs Systems. Eucalyptus allows building Amazon AWS-compatible private and hybrid clouds.

OpenNebula [13] is a free software project initiated in 2008. It is sponsored by C12G, a cloud computing company associated with the Scientific Park of Madrid, and maintained by the OpenNebula Community. OpenNebula aims at providing an industry standard solution for creating and managing virtualised enterprise data centres and IaaS clouds.

OpenStack [9] is a free software project launched in 2010. It was originally developed by Rackspace and NASA and is currently maintained by the OpenStack Foundation with contributions from the major players in cloud computing. OpenStack provides an API and a dashboard to manage pools of computing, storage, and networking resources.

vCloud [10] is commercial integrated cloud infrastructure solution launched in 2008 and developed by VMWare.

As depicted by the IaaS providers column of Table II, the cloud market seems to be consolidating at the IaaS level towards a few IaaS stacks. As shown for the 21 public providers listed in Table I, seven providers adopt OpenStack (four fully, and three partially), four providers adopt VMWare vCloud, one provider adopts CloudStack and the remaining nine adopt proprietary stacks (although one is compatible with Amazon AWS APIs). This indicates that OpenStack has gained relatively wide acceptance across public providers. This trend may increase the interoperability across providers adopting the same stack. However, it does not support the development and administration of multi-clouds systems.

C. Libraries

A second step towards supporting multi-cloud systems is provided by some IaaS/PaaS libraries such as jclouds [14], Deltacloud [15], or Simple Cloud [16]. These libraries provide abstraction layers facilitating the provisioning and deployment of multi-cloud systems through a single interface. They support numerous IaaS providers as well as IaaS stacks. These libraries are at the border between IaaS and PaaS levels since they allow, for instance, a developer to run scripts on a virtual machine or to deploy a load balancer that may rely on platform services.

Table III shows a classification of these libraries. This classification is based on license, implementation languages, and supported providers/stacks of each library.

fog [17] is a Ruby API providing access to computing and storage facilities on multiple clouds. It helps developers in testing and simulating their deployment by providing an in-memory representation of cloud resources.

jclouds [14] is a Java and Clojure API delivering an abstraction layer over the APIs of IaaS providers and stacks. It facilitates developers in describing generic virtual machines by means of templates. It also allows deploying multiple virtual machines and managing them as a group.

libCloud [18] is a Python API providing solutions for managing multiple clouds that are akin to the ones of jclouds.

Simple Cloud [16] is a PHP API delivering mechanisms for managing the life-cycle of a virtual machine on multiple clouds. It offers services for data storage, message queue, and monitoring.

Table I. PROVIDERS

Provider	Headquarters	Data centres location	Uptime SLA
Amazon AWS	USA	USA, Brazil, Ireland, Japan, Singapore, Australia	99.95%
AT&T Cloud	USA	USA	100.00%
Bit Refinery	USA	USA, UK	100.00%
GoGrid	USA	USA, Netherlands	100.00%
Google Compute Engine	USA	USA, EU (Unspecified)	99.95%
Hosting.com	USA	USA	100.00%
HP Cloud	USA	USA	99.95%
IBM SmartCloud Enterprise	USA	USA, Germany, Japan	99.90%
Microsoft Windows Azure	USA	USA, Ireland, Netherlands, Hong Kong, Singapore	99.95%
Nephoscale	USA	USA	99.90%
OpSource	USA	USA, France, UK	100.00%
Rackspace	USA	USA, UK, Hong Kong	100.00%
ReliaCloud	USA	USA	100.00%
Softlayer	USA	USA, Netherlands, Singapore	100.00%
Terramark	USA	USA, Canada, Brazil, Colombia, Dominican Republic, Belgium, France, Germany, Ireland, Italy, Luxembourg, Netherlands, Spain, Sweden, Turkey, UK, China, Japan, Singapore, Australia	100.00%
Aruba Cloud	Italy	Italy	99.95%
CloudSigma	Switzerland	Switzerland, USA	100.00%
Gandi	France	France, USA	99.95%
GreenQloud	Iceland	Iceland	100.00%
Lunacloud	UK	France, Germany, Latvia, Portugal	99.99%
Memset	UK	UK	99.99%

Table II. STACKS

Stack	License	Implementation languages	Supported hypervisors	Main contributors	Adopters
CloudStack	Apache License 2.0	Java	KVM, Citrix Xen, VMWare vSphere	Citrix, Apache Software Foundation	GreenQloud
Eucalyptus	GPL v3	Java, C	KVM, Citrix Xen, VMWare vSphere	Eucalyptus Systems	
OpenNebula	Apache License 2.0	C++, C, Ruby, Java, Shell script, lex, yacc	KVM, Citrix Xen, Oracle VM, VMWare vSphere	OpenNebula Community	
OpenStack	Apache License 2.0	Python	KVM, Citrix Xen, VMWare vSphere	Rackspace, NASA	AT&T Cloud Architect, HP Cloud, IBM SmartCloud Enterprise, Nephoscale (storage), Rackspace, Softlayer (storage), Memset (storage)
VMWare vCloud	Commercial		VMWare vSphere	VMWare	Bit Refinery, Hosting.com, ReliaCloud, Terramark

Table III. LIBRARIES

Stack	License	Implementation languages	Supported providers/stacks
Deltacloud	Apache License 2.0	Ruby	http://deltacloud.apache.org/supported-providers.html
fog	MIT License	Ruby	http://fog.io/about/supported_services.html
jclouds	Apache License 2.0	Java	http://www.jclouds.org/documentation/reference/supported-providers/
libCloud	Apache License 2.0	Python	http://libcloud.apache.org/supported_providers.html
Simplecloud	BSD license	PHP	

Most of these libraries are language-dependent since they are designed to interface with programming language like Ruby, Java, and PHP. However, this is not the case of Deltacloud [15], another API providing drivers for computing and storage facilities. It consists of a REST interface where clients send requests to a Deltacloud server (on a local or remote machine) wrapping the drivers to the various cloud providers. This approach is language-independent but introduces a single point of failure.

These libraries support the development and administration of multi-cloud systems by providing abstraction layers to multiple clouds. However, they do not provide mechanisms for *automatic* provisioning and deployment of multi-cloud systems.

D. Frameworks

The latest step towards supporting multi-cloud systems is provided by some specific PaaS frameworks. These frameworks aim at reducing the complexity of managing multi-clouds systems. They provide capabilities for the provisioning, deployment, monitoring, and adaptation of multi-cloud systems without being language-dependent. They partially reuse the IaaS/Paas libraries (see Figure 1). As claimed in [19], two main types of PaaS can be distinguished. One type of PaaS such as OpenShift [20] considers the underlying IaaS as a black box; *i.e.*, it does not provide visibility and control over the underlying infrastructure. Another type of PaaS considers the same IaaS as a white box, *i.e.*, it provides full visibility and control over the underling infrastructure. Without visibility and control on the underlying infrastructure, developers can not explicitly adapt the infrastructure to optimise performance, availability, and cost.

In the following, we consider the latter type of frameworks. Some of these frameworks rely on so-called "DevOps" tools such as Chef [21] and Puppet [22] that automate the deployment of applications, as well as the management of cloud capabilities. With visibility and control on both IaaS and PaaS levels, developers can exploit the peculiarities of cloud solutions at each level of the cloud stack. These frameworks embed mechanisms to monitor cloud resources and their

consumption (*e.g.*, computing, memory, storage, networking), as well as the applications and their status. They also offer cloud-specific adaptation mechanisms such as load balancing, auto scaling, and automatic failure recovery.

Table IV shows a classification of these frameworks. This classification is based on license, implementation languages, interfaces, supported providers/stacks, monitoring support, and adaptation support.

These frameworks provide mechanisms for optimising performance, availability, and cost of multi-cloud systems. However, they do not provide any methodology supporting the engineering of multi-cloud systems at a high level of abstraction. Thus, the developer would still hack at the source code level, which is challenging to maintain, rather than engineering at the modelling level.

E. EU projects

Several on-going European projects are providing stacks, libraries or frameworks for the provisioning, deployment, monitoring and adaptation of cloud-based systems at IaaS or PaaS levels. In the following, we present these projects with a focus on their ability to target multi-clouds systems.

4CaaST [23] delivers a solution for elastic and optimised hosting of Internet-scale multi-tier applications. This solution is based on Chef to monitor the execution and manage the life-cycle of applications.

ARTIST [24] aims at providing model-based techniques for representing applications as well as cloud infrastructures and platforms. The expected outcomes of the project are a vendor- and platform-independent methodology and an automation-oriented toolset for re-engineering, migration, maintenance and evolution of cloud-based applications.

CELAR (Cloud ELAsticity pRovisining) [25] aims at delivering an automated and customisable system for elastic provisioning of resources in cloud computing platforms. The expected outcomes of the project are a middleware for elastic provisioning that automatically manages and adapts cloud resources, an information system describing cloud resources and providing a search mechanism, and a scalable monitoring tool.

Cloud4SOA [26], [27] supports cloud application developers with multi-platform management, monitoring and migration by semantically interconnecting heterogeneous PaaS offerings. The solution currently supports CloudFoundry, Heroku, OpenShift, and Amazon Elastic Beanstalk.

CloudScale [28] aims at supporting scalable service engineering. The expected outcomes of the project are tools and methods for the modelling of design alternatives and the analysis of their effect on scalability and cost, as well as for detecting scalability problems by analysing code.

Contrail [29], [30] aims at solving the vendor lock-in problem by allowing the seamless switch of cloud provider. The solution requires an agreement in the adoption of a common technology stack among cloud providers.

CumuloNimbo [31] provides a solution for high scalability without sacrificing data consistency and ease of programming. The solution provides self-healing subsystems that automatically repair themselves in the event of failures without disrupting service provisioning and without introducing data inconsistencies during recovery.

MODAClouds [32], [3] aims at delivering methods, a Decision Support System (DDS), an Integrated Development Environment (IDE), and a run-time environment for the high-level design, early prototyping, semiautomatic code generation, and automatic deployment of applications on multiple clouds with guaranteed QoS. The work presented in this paper is partially financed by this project (see Section V).

mOSAIC [33], [34] tackles the vendor lock-in problem by providing an open-source platform including an API for provisioning and deployment of applications on multiple clouds.

OPTIMIS [35], [36] aims at enabling organisations to automatically externalise services and applications to trustworthy and auditable cloud providers in the hybrid model. The solution provides some continuous monitoring mechanisms that can be used to check SLA violations.

PaaSage [37] aims at delivering an open and integrated platform to support both design and deployment of cloud applications, together with an accompanying methodology that allows model-based development, configuration, optimisation, and deployment of existing and new applications independently of the existing underlying cloud infrastructures. The work presented in this paper is partially financed by this project (see Section V).

REMICS [38] supports legacy systems migration to clouds by providing model-driven methodology and tools, which significantly improve the baseline OMG's Architecture Driven Modernization (ADM) initiative. The work presented in this paper represents a continuation of the work done in this project [2] (see Section V).

Reservoir [39], [40] provides solutions for managing the provisioning of IaaS resources on demand. The expected outcome of the project is to enable providers of cloud infrastructure to dynamically partner with each other.

F. Discussion

The stacks, libraries and frameworks presented in this Section provide mechanisms to automate the provisioning and deployment of application on multiple clouds. However, as explained in [3], there is a *"... need for developers to be able to design their software systems for multiple Clouds and for operators to be able to deploy and re-deploy these systems on various Clouds depending on the convenience. The current Cloud literature, however, does not seem to pose attention to this issue as it is focused on considering the perspective of the Cloud providers, by offering mechanisms for auto-scaling of Clouds and for interoperability and federation between Clouds."*

MDE is a branch of software engineering which aims at improving the productivity, quality and cost-effectiveness of software development by shifting the paradigm from code-centric to model-centric. This approach, which is commonly summarised as "model once, generate anywhere", is particularly relevant to tame the complexity of developing complex systems such as multi-cloud systems. Models and modelling

Table IV. FRAMEWORKS

Framework	License	Implementation languages	Interface	Supported providers/stacks	Monitoring support	Adaptation support
Cloudify	Apache License 2.0	Java, Groovy, JavaScript	CLI, web-based monitoring interface, REST API to Cloudify service	Amazon, OpenStack, Azure, HP cloud, Rackspace, your own local provider	Application status and logs, Deployment status and logs, Resource metrics	Auto-scaling based on metrics on resources and number of virtual machines, Automatic failure recovery
Cloud Foundry	Apache License 2.0	Ruby, Java, JavaScript	REST API, CLI, Eclipse plug-in	Amazon, OpenStack, Rackspace, Eucalyptus, your own local provider	Application status and logs, Environment variables, Resources metrics	Change the number of virtual machines associated to an application, Automatic failure recovery
Scalr	Apache License 2.0	Python, PHP, JavaScript	REST API, Web-based user interface	Amazon, OpenStack, Rackspace, Nimbula, Eucalyptus, IDC Frontier, CloudStack, Cloud Foundry	Application status and logs, Load statistics, Notification system	Auto-scaling of the infrastructure and database when overloaded or scheduled, Automatic failure recovery

languages as the main artefacts of the development process enable developers to work at a high level of abstraction by focusing on cloud concerns rather than implementation details. Model transformation as the primary technique to generate (parts of) software systems restrains developers from repetitive and error-prone tasks such as coding.

Domain-specific modelling languages (DSMLs) provide abstractions and notations that allow direct and understandable expression of domain concepts instead of encoding these in a lower level programming language. DSMLs are particularly relevant for facilitating the specification of provisioning and deployment concerns of multi-cloud systems at design-time.

The frameworks presented in this Section offer some cloud-specific techniques and methods for adaptation and self-adaptation [41], such as load balancing, auto scaling, and failure recovery. These adaptations are triggered when some of the requirements specified at design-time are not fulfilled any more. These requirements are related either to computing resources (*e.g.*, the compute load should be below 75%) or to desired topologies (*e.g.*, the application should be deployed and running on at least two virtual machines). Self-adaptive systems are generally based on a control loop like the well-known Monitor–Analyse–Plan–Execute from autonomic computing [41]. The input of the reasoning systems consists of observables describing the running system and its context. The output consists of a set of adaptation actions. The implementation of this loop in the context of multi-cloud systems can be particularly complex.

Models@run-time [42] extend the adoption of models to the run-time environment. This approach is particularly relevant for facilitating the dynamic adaptation of multi-cloud systems at run-time. It has already been shown that models@run-time facilitate reasoning about- and dynamic adaptation of running systems [43] by providing an abstract representation of the system causally connected to the running system; *i.e.*, a change in the model of the system is reflected on-demand in the running system, whereas a change in the running system is automatically reflected in the model of the system. This enables the continuous evolution of the system with no strict boundaries between design-time and run-time activities.

As far as we know, the frameworks presented in this section do not provide such abstraction. This deficiency is compensated by CloudML.

III. CLOUDML

CloudML aims at facilitating the provisioning, deployment, monitoring, and adaptation of multi-cloud systems. CloudML is built upon MDE techniques and methods, and provides: (i) a DSML for modelling the provisioning and deployment of multi-cloud systems at design-time; (ii) a models@run-time environment for enacting the provisioning, deployment, and adaptation of these systems, as well as monitor their status at run-time. This run-time environment can be accessed by a reasoning system through a model-based interface. CloudML is agnostic to any development paradigm and technology, meaning that the developers can design and implement the applications based on their preferred paradigms and technologies.

A. Roles and work-flow

CloudML considers two possible roles in the deployment work-flow: a cloud application developer (hereafter called *cloud-app developer*) and a cloud application vendor (hereafter called *cloud-app vendor*).

The cloud-app developer develops the applications to be deployed on the cloud. She knows the internals of these systems, so she can model their topologies together with requirements, constraints, and dependencies. This information is collected into one or more templates of the provisioning and deployment model.

The cloud-app vendor delivers the applications as a service to consumers. She does not necessarily know about the internals of these systems, but she can specify a set of additional requirements, constraints, and dependencies related to her business (*e.g.*, budget constraints).

At design-time, the typical deployment work-flow will consist of two steps. First, the cloud-app developer specifies one or more templates of the provisioning and deployment model. Second, the cloud-app vendor adjusts and combines these templates into the actual provisioning and deployment model.

B. Design-time

Figure 2 presents the architecture of CloudML, which reflects part of the OMG Model-Driven Architecture [44].

The DSML of CloudML is specified by a metamodel which, inspired by component-based approaches, implements the *type-instance* pattern [45]. At design-time, this DSML is

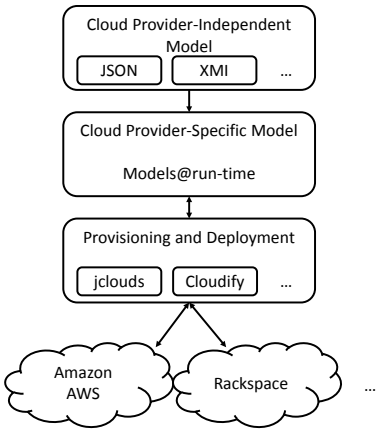

Figure 2. The architecture of CloudML

used to specify the provisioning and deployment models. These models encompass the topology of the *nodes* of the cloud infrastructure, as well as the topology of the software *artefacts* deployed on these nodes.

CloudML considers provisioning and deployment models at two levels of abstraction, namely Cloud Provider-Independent Model (CPIM), and Cloud Provider-Specific Model (CPSM) (see Figure 2).

The CPIM represents a generic provisioning and deployment model that is independent of the cloud provider. This model consists of two main kinds of elements, namely the *node types* and the *artefacts types*.

A node type represents a generic virtual machine (*e.g.*, a virtual machine running GNU/Linux). This element can be parameterised by provisioning requirements (*e.g.*, 2 cores $\leq$ compute $\leq$ 4 cores, 2 GiB $\leq$ memory $\leq$ 4 GiB, storage $\geq$ 10 GiB, location = Europe).

An artefact type represents a generic component of the application (*e.g.*, a Java servlet of an application for document collaboration, a Jetty container, or a MongoDB database). This element can be annotated with *deployment commands* (*e.g.*, retrieve the Java servlet from `http://cloudml.org/`, configure it, and run it), and constraints. Artefact types may be grouped together and reused in the form of composites.

As depicted in Figure 3, artefact types can expose two kinds of ports: *requirement ports* (*e.g.*, the Java servlet requires an artefact type providing the JettyCapability), and *communication ports* (e.g., the Java servlet is accessible on port 443). Moreover, two kinds of bindings can be specified between artefacts types: *deployment dependencies* (*e.g.*, the Jetty container and the MongoDB database have to be deployed before the Java servlet), and *communication channels* (*e.g.*, a Java servlet communicates with another Java servlet through Hypertext Transfer Protocol Secure (HTTPS) on port 443). These bindings can be annotated with requirements and constraints.

Currently, the CPIM can be serialised using two formats, namely the JavaScript Object Notation (JSON) and the XML Metadata Interchange (XMI).

Listing 1 shows an excerpt of a CPIM in JSON format.

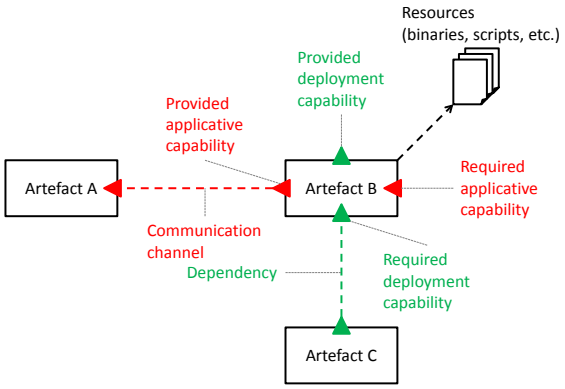

Figure 3. The ports and bindings in a CPIM

Listing 1. An excerpt of a CPIM in JSON format

```
{
    "id": "DocsDeployment",
    "nodeTypes": [
        {
            "id": "SmallGNULinux",
            "os": "GNULinux",
            "compute": [ 2, 4 ],
            "memory": [ 2048, 4096 ],
            "storage": [ 10240 ],
            "location": "eu",
            "provides": [
                { "id": "SSHCapability" }
            ]
        }
    ],
    "artefactTypes": [
        {
            "id": "MongoDB",
            "retrieval": "wget http://cloudml.org/services/
                mongodb.sh",
            "deployment": "sudo mongodb.sh",
            "provides": [
                { "id": "MongoDBCapability" }
            ]
        },
        {
            "id": "Jetty",
            "retrieval": "wget http://cloudml.org/services/
                jetty.sh",
            "deployment": "sudo jetty.sh",
            "provides": [
                { "id": "JettyCapability" }
            ]
        },
        {
            "id": "Docs",
            "retrieval": "wget http://cloudml.org/apps/docs.
                war; wget http://cloudml.org/apps/
                docs_configure.sh; wget http://cloudml.org/
                apps/docs_deploy.sh",
            "configuration": "sudo docs_configure.sh",
            "deployment": "sudo docs_deploy.sh",
            "requires": [
                { "id": "JettyCapability" },
                { "id": "MongoDBCapability" }
            ]
        }
    ]
}
```

The CPIM is transformed semi-automatically into a CPSM, which represents a specific provisioning and deployment model that is dependent on the cloud provider. Hence, similar to the CPIM, this model considers two main concepts, namely *node instances* and *artefact instances*.

A node instance represents an instance of a virtual machine on a specific cloud provider (*e.g.*, a virtual machine running

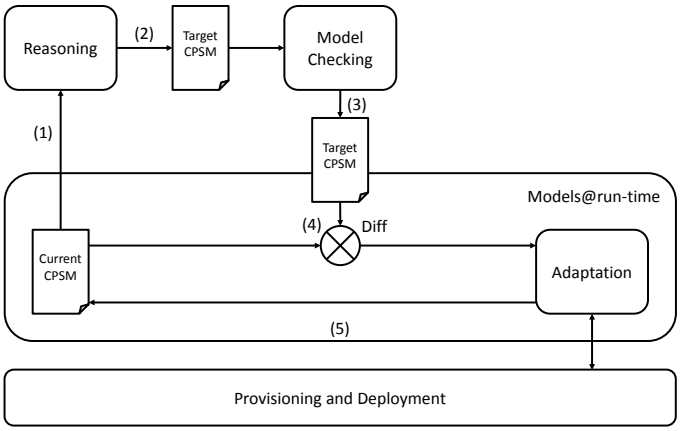

Figure 4. The models@run-time approach

GNU/Linux on Amazon EC2). An artefact instance represents an instance of a component of the application on a specific virtual machine (*e.g.*, an instance of the Java servlet on the virtual machine above).

Finally, the CPSM is transformed to some text-based input to the libraries and frameworks presented in Section II, which will enact the provisioning and deployment of the system.

C. Run-time

At run-time, the CPSM is causally connected to the running system; *i.e.*, a change in the CPSM is reflected on-demand in the running system, whereas a change in the running system is automatically reflected in the CPSM. As mentioned, it has already been shown that models@run-time facilitate reasoning about- and dynamic adaptation of running systems [43].

Figure 4 (adapted from [46]) depicts the architecture of the models@run-time environment. The reasoning system reads the current CPSM (step 1) and produces a target CPSM (step 2). Then, the model checker validates the target CPSM (step 3). If the validation is passed, the run-time environment calculates the difference between the current CPSM and the target CPSM (step 4). Then, the adaptation system enacts the adaptation on the parts of the running system which are included in the difference. Finally, the target CPSM becomes the current CPSM (step 5).

IV. Future work and Challenges

CloudML is at the early stage of development. However, we have already identified some challenges that we intend to address in a future work. In the following, we present some of these challenges.

Modelling QoS constraints. QoS requirements are the requirements defining the service-level agreement for a system. Cloud-app developers and vendors may not always be able to specify these requirements at design-time in a cloud-agnostic way; *e.g.*, a cloud-app provider may not be able to specify the location of the systems without knowing the location of the consumers. The challenge is to model QoS requirements at design-time in a cloud-agnostic way and to bind these requirements to the run-time so that modifying them leads to the adaptation of the running system.

Enacting adaptation in a timely fashion. The enactment of an adaptation can be time-consuming, and this has already been identified as a challenge for adaptive systems [47]. This is because if the enactment of an adaptation is too time-consuming, a change in the environment may require another adaptation while the system is still being adapted. In this case, the result of the adaptation is a system that needs to be adapted again. On the one hand, if the enactment of an adaptation is interrupted every time a change requires another adaptation, the system may never be adapted. On the other hand, if the frequency of changes requiring an adaptation is higher than the frequency of enactments of an adaptation, the system may fall in a continuous loop of adaptation, also referred to as *oscillation*. The challenge is to find the right balance between length and frequency of enactments of an adaptation, which can be challenging in cloud-based systems, where adaptation actions can be particularly time-consuming; *e.g.*, provisioning a virtual machine can take several minutes.

Handling failure during adaptation. The enactment of an adaptation may be subject to failures. This is because the adaptation of multi-cloud systems involves complex actions that consist of several sub-actions; *e.g.*, provisioning a virtual machine consists of authenticating with the cloud provider, finding the right image, and finally provisioning it. The challenge is to find techniques and methods to prevent and handle failures.

Location of datacentres. The location of datacentres may have legal implications for the data stored and processed by multi-cloud systems, aside from obeying to data protection laws and regulations; *e.g.*, the data held in a particular region may be accessed by local authorities without the notification to the consumer. The challenge is to seamlessly move data from one region to another without legal consequences.

V. Conclusion

In this paper, we provide a classification of the state-of-the-art of cloud solutions, and explain how MDE techniques and methods facilitating the specification of provisioning, deployment, monitoring, and adaptation concerns of multi-cloud systems at design-time and their enactment at run-time.

The solution outlined in this paper is CloudML, built upon MDE techniques and methods. The DSML facilitates the specification of provisioning and deployment concerns of multi-cloud systems at design-time. Moreover, the models@run-time environment facilitates reasoning about- and dynamic adaptation of running systems [43] by providing an abstract representation of the system causally connected to the running system. This enables the continuous evolution of the system with no strict boundaries between design-time and run-time activities.

Acknowledgements

This work is partially funded by the EU commission in the FP7 programme through the MODAClouds project [32], contract number 318484, the PaaSage project [37], contract number 317715, and the REMICS project [38], contract number 257793.

References

[1] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," National Institute of Standards and Technology, Special Publication 800-145, September 2001.

[2] E. Brandtzæg, M. Parastoo, and S. Mosser, "Towards a Domain-Specific Language to Deploy Applications in the Clouds," in *CLOUD COMPUTING 2012: 3rd International Conference on Cloud Computing, GRIDs, and Virtualization*. IARIA, 2012, pp. 213–218.

[3] D. Ardagna, E. Di Nitto, G. Casale, D. Petcu, P. Mohagheghi, S. Mosser, P. Matthews, A. Gericke, C. Balligny, F. D'Andria, C.-S. Nechifor, and C. Sheridan, "MODACLOUDS, A Model-Driven Approach for the Design and Execution of Applications on Multiple Clouds," in *ICSE MiSE: International Workshop on Modelling in Software Engineering*. IEEE/ACM, 2012, pp. 50–56.

[4] R. Prodan and S. Ostermann, "A survey and taxonomy of infrastructure as a service and web hosting cloud providers," in *GRID 2009: 10th IEEE/ACM International Conference on Grid Computing*. IEEE, 2009, pp. 17–25.

[5] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: comparing public cloud providers," in *IMC 2010: 10th ACM SIGCOMM Conference on Internet Measurement*, M. Allman, Ed. ACM, 2010, pp. 1–14.

[6] W. Jansen, "Cloud Hooks: Security and Privacy Issues in Cloud Computing," in *HICSS 2011: 44th Hawaii International Conference on Systems Science*. IEEE Computer Society, 2011, pp. 1–10.

[7] "CloudSleuth," 2015. [Online]. Available: https://cloudsleuth.net/

[8] "Amazon Web Services," 2015. [Online]. Available: http://aws.amazon.com/

[9] "OpenStack," 2015. [Online]. Available: http://www.openstack.org/

[10] "vCloud," 2015. [Online]. Available: http://vcloud.vmware.com/

[11] "CloudStack," 2015. [Online]. Available: http://incubator.apache.org/cloudstack/

[12] "Eucalyptus," 2015. [Online]. Available: http://www.eucalyptus.com/

[13] "OpenNebula," 2015. [Online]. Available: http://opennebula.org/

[14] "jclouds," 2015. [Online]. Available: http://www.jclouds.org/

[15] "Deltacloud," 2015. [Online]. Available: http://deltacloud.apache.org/

[16] "Simple Cloud," 2015. [Online]. Available: http://simplecloud.org/

[17] "fog," 2015. [Online]. Available: http://fog.io/

[18] "Libcloud," 2015. [Online]. Available: http://libcloud.apache.org/

[19] "The blurring line between PaaS and IaaS," 2015. [Online]. Available: http://natishalom.typepad.com/nati_shaloms_blog/2012/10/paas-as-an-infrastructure.html

[20] "OpenShift," 2015. [Online]. Available: https://openshift.redhat.com/app/

[21] "Chef," 2015. [Online]. Available: http://www.opscode.com/chef/

[22] "Puppet," 2015. [Online]. Available: https://puppetlabs.com/

[23] "4CaaST EU project," 2015. [Online]. Available: http://4caast.morfeo-project.org/

[24] "ARTIST EU project," 2015. [Online]. Available: http://www.artist-project.eu/

[25] "CELAR EU project," 2015. [Online]. Available: http://www.celarcloud.eu/

[26] "Cloud4SOA EU project," 2015. [Online]. Available: http://www.cloud4soa.eu/

[27] N. Loutas, V. Peristeras, T. Bouras, E. Kamateri, D. Zeginis, and K. A. Tarabanis, "Towards a Reference Architecture for Semantically Interoperable Clouds," in *CloudCom 2010: 2nd IEEE International Conference on Cloud Computing Technology and Science*, 2010, pp. 143–150.

[28] "CloudScale EU project," 2015. [Online]. Available: http://www.cloudscale-project.eu/

[29] "Contrail EU project," 2015. [Online]. Available: http://contrail-project.eu/

[30] E. Carlini, M. Coppola, P. Dazzi, L. Ricci, and G. Righetti, "Cloud Federations in Contrail," in *Euro-Par Workshops*, 2011, pp. 159–168.

[31] "CumuloNimbo EU project," 2015. [Online]. Available: http://www.cumulonimbo.eu/

[32] "MODAClouds EU project," 2015. [Online]. Available: http://www.modaclouds.eu/

[33] "mOSAIC EU project," 2015. [Online]. Available: http://www.mosaic-project.eu/

[34] C. Sandru, D. Petcu, and V. I. Munteanu, "Building an Open-Source Platform-as-a-Service with Intelligent Management of Multiple Cloud Resources," in *UCC 2012: 5th IEEE International Conference on Utility and Cloud Computing*. IEEE Computer Society, 2012, pp. 333–338.

[35] "Optimis EU project," 2015. [Online]. Available: http://www.optimis-project.eu/

[36] A. J. Ferrer, F. Hernández, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R. M. Badia, K. Djemame, W. Ziegler, T. Dimitrakos, S. K. Nair, G. Kousiouris, K. Konstanteli, T. A. Varvarigou, B. Hudzia, A. Kipp, S. Wesner, M. Corrales, N. Forgó, T. Sharif, and C. Sheridan, "OPTIMIS: A holistic approach to cloud service provisioning," *Future Generation Comp. Syst.*, vol. 28, no. 1, pp. 66–77, 2012.

[37] "PaaSage EU project," 2015. [Online]. Available: http://www.paasage.eu/

[38] "REMICS EU project," 2015. [Online]. Available: http://remics.eu/

[39] "Reservoir EU project," 2015. [Online]. Available: http://www.reservoir-fp7.eu/

[40] L. Rodero-Merino, L. M. V. Gonzalez, V. Gil, F. Galán, J. Fontán, R. S. Montero, and I. M. Llorente, "From infrastructure delivery to service management in clouds," *Future Generation Comp. Syst.*, vol. 26, no. 8, pp. 1226–1240, 2010.

[41] J. Kephar and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, January 2003.

[42] G. Blair, N. Bencomo, and R. France, "Models@run.time," *IEEE Computer*, vol. 42, no. 10, pp. 22–27, 2009.

[43] B. Morin, O. Barais, J.-M. Jézéquel, F. Fleurey, and A. Solberg, "Models@Run.time to Support Dynamic Adaptation," *IEEE Computer*, vol. 42, no. 10, pp. 44–51, 2009.

[44] "OMG Model-Driven Architecture," 2015. [Online]. Available: http://www.omg.org/mda/

[45] C. Atkinson and T. Kühne, "Rearchitecting the UML infrastructure," *ACM Transactions on Modeling and Computer Simulation*, vol. 12, no. 4, pp. 290–321, 2002.

[46] F. Fouquet, E. Daubert, N. Plouzeau, O. Barais, J. Bourcier, and J.-M. Jézéquel, "Dissemination of Reconfiguration Policies on Mesh Networks," in *DAIS 2012: 12th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems*, ser. Lecture Notes in Computer Science, K. M. Göschka and S. Haridi, Eds., vol. 7272. Springer, 2012, pp. 16–30.

[47] J.-Y. Tigli, S. Lavirotte, G. Rey, V. Hourdin, N. Ferry, C. Vergoni, and M. Riveill, "Low response time context awareness through extensible parameter adaptation with ORCA," *Annales des Télécommunications*, vol. 67, no. 7-8, pp. 313–327, 2012.