Journal of Cloud Computing
a SpringerOpen Journal

RESEARCH Open Access

Multi-cloud resource management: cloud service interfacing

Victor Ion Munteanu[1,2*], Călin Şandru[1,2] and Dana Petcu[1,2]

Abstract

Cloud service abstractions are currently used to hide the underlying complexity given by existing technologies and services, in hope of facilitating the enacting of Cloud Federations and Marketplaces. In particular, resource management systems dealing with multiple Cloud providers need to expose an uniform interface for various services and to build wrappers for the Cloud service APIs. In this paper we discuss the solution adopted by a recent developed open-source and vendor agnostic platform-as-a-service for Multi-Cloud application deployment. The middleware includes a multi-agent system for automatic Cloud resource management. With a modular design, the solution provides a flexible approach to encompass new Cloud service offers as well as new resource types. This paper focuses on the modules which enable resource abstraction and automatized management.

Introduction

Offering rapid access to a large pool of available hardware and software resources to a large variety of users, the Cloud computing has been rapidly adopted by business and academic communities. The most preferred services are the ones from the Infrastructure-as-a-Service (IaaS) category and a large number of such services are available all over the world.

Currently, there are many reasons for the use of services from multiple Clouds. We can name here few scenarios: optimize costs or improve quality of services; react to changes in existing provider offers; follow constraints, like new locations or laws; avoid dependency on only one external provider; ensure backup-ups to deal with disasters or scheduled downtime; deal with the peaks in service and resource load by offloading on external ones, on demand basis; replicate applications/services by consuming services from different Clouds to ensure their high availability; act as intermediary; enhance own Cloud resource and service offers, based on agreements with other providers; consume different services for their particularities not provided elsewhere.

According to [1], a Multi-Cloud denotes the usage of multiple and independent Clouds by a client or a service. It does not imply interconnection and sharing between Clouds. The clients or their software representatives are responsible for managing resource provisioning. The selection of the best fitted place to deploy a Cloud application is a complex technical issue in a Multi-Cloud that requires the introduction of a Cloud resource management layer based on vendor-independent brokers and semi-automated tools (including knowledge-based selection methods for Cloud services). Such a resource management system should be able to hide the complexity of service selection procedures and to control the lifecycle of the resources and services allocated to a certain application.

The mOSAIC project consortium (http://www.mosaic-cloud.eu) has recently proposed and developed an open-source Platform-as-a-Service focusing on ensuring the portability of applications consuming Cloud resources from Private or Public Clouds (the acronym stands for Open-source API and Platform for multiple Clouds). It complies with the requirements of a Multi-Cloud resource management system. In order to achieve its goal to serve application developers, the PaaS relies upon artificial intelligence methods in the different procedures, like in the selection of the Cloud resources to be consumed.

mOSAIC, as a whole, is of modular design, allowing modules to be used as a whole or individually, individual modules servicing specific purposes. Previous papers about mOSAIC's platform have reported the

*Correspondence: vmunteanu@info.uvt.ro
[1]West University of Timişoara, Timişoara, Romania
[2]Institute e-Austria Timişoara, Timişoara, Romania

design and functionality of different architectural modules. In this paper we put a special focus on the platform modules which are allowing the connection to different Cloud resources to be consumed. The following sections describe them in detail as well as their integration within the whole platform.

The work carried out in this paper is of direct consequence of a general lack of support for common standards coming from Cloud Vendors, each of them having proprietary, closed source, implementations with custom interfaces and APIs. This in turn makes it difficult for cloud application developers to create provider independent cloud applications and forces them to spend time away from working on their applications to work on integrating various Cloud Vendor technologies.

This paper is an extension of the authors' paper presented at ITAAC 2012 [2]. The extension consists in the description of the interaction with the Cloud resources, as well as a the functionality of the tool designed as proof-of-concept.

The remainder of the paper is organized as follows. Section 'Short overview of mOSAIC's approach for interaction with Cloud services' gives a brief introduction to the mOSAIC's approach to the interaction with Cloud services. The main results are presented in Section 'Vendor module role and functionality' where our approach to Vendor Modules is detailed along with use cases for which the solution was designed and in Section 'Proof-of-concept implementation' a proof-of-concept implementation is presented. Finally, conclusions and future work are presented in Section 'Conclusions'.

Short overview of mOSAIC's approach for interaction with Cloud services

Triggered by the need of a solution for the portability problem, mOSAIC was designed to be an open-source and deployable middleware able to support applications which are consuming Private or Public Cloud services. An overview of the entire solution can be found in the recent paper [3].

Figure 1 captures mOSAIC's architecture as a series of grouped components. The top part represents proof of concept applications that were developed on top of mOSAIC. The bottom part is the mOSAIC PaaS which is composed of: an application support layer made up of APIs, tools and semantic support; software platform support which is behind the high level APIs and handles execution; infrastructure support which handles the management of the infrastructure; cloud adaptors which for the basis of the PaaS and communicate directly to various Cloud services and providers.

mOSAIC offers integration of Cloud services which is achieved through an interface that is instantiated in three forms:

1. an abstract entity (e.g. an object) in a programming language, mainly used at the design stage of the application;
2. a wrapper that allows the service to be integrated in the platform, mainly used at the run time by the platform as an intermediary for the application;
3. a representation in the service acquisition and SLA management processes, mainly at the deployment time of the support platform and the application.

In the next sections we will focus on the last case. The other two cases we present shortly in what follows.

Design time: language-dependent and vendor-independent abstraction of the Cloud services

The developer of a new application intended to run in Cloud environment is invited to describe the application in Java, Python and Erlang following the mOSAIC's API recommendations so that the application is not depending on a certain implementation of a Cloud service. An application can profit from the elasticity at the level of components (instead at a larger granularity level, as usual, at virtual machine level), if the component is able to scale. An event-driven programming style has been adopted to reduce the network traffic.

The first level of interface with the Cloud service (the abstraction layer) is done to the level of the language-dependent APIs through the so-called Cloudlets and Connectors. The first ones are expressing the reaction of the application to the events related to Cloud resource consumption. The second ones are generic in terms of operations allowed for a certain type of Cloud resource (e.g. key-value store, distributed file system, http gateway, or message passing system).

The latest detailed description of the mOSAIC's API can be found in [4].

Run time: wrappers of cloud service interfaces

An interoperability service of the mOSAIC's platform acts at run-time as a proxy between a vendor-agnostic and language-dependent Connector used by a certain application and a Driver of a certain type of Cloud resource (e.g. message queuing system, key value store, distributed file system). The Driver is wrapping the native API of the Cloud service in order to enable the service to interact with the other components of the platform or application. Deployable open-source services (like RabbitMQ as message queuing system, Jetty as web server, or Riak as key-value store) are used as Cloud resources available on the provisioned virtual machines on which the mOSAIC platform is deployed. The open-source code of the latest stable version includes more then ten drivers for various deployable or hosted services and is provided at https://

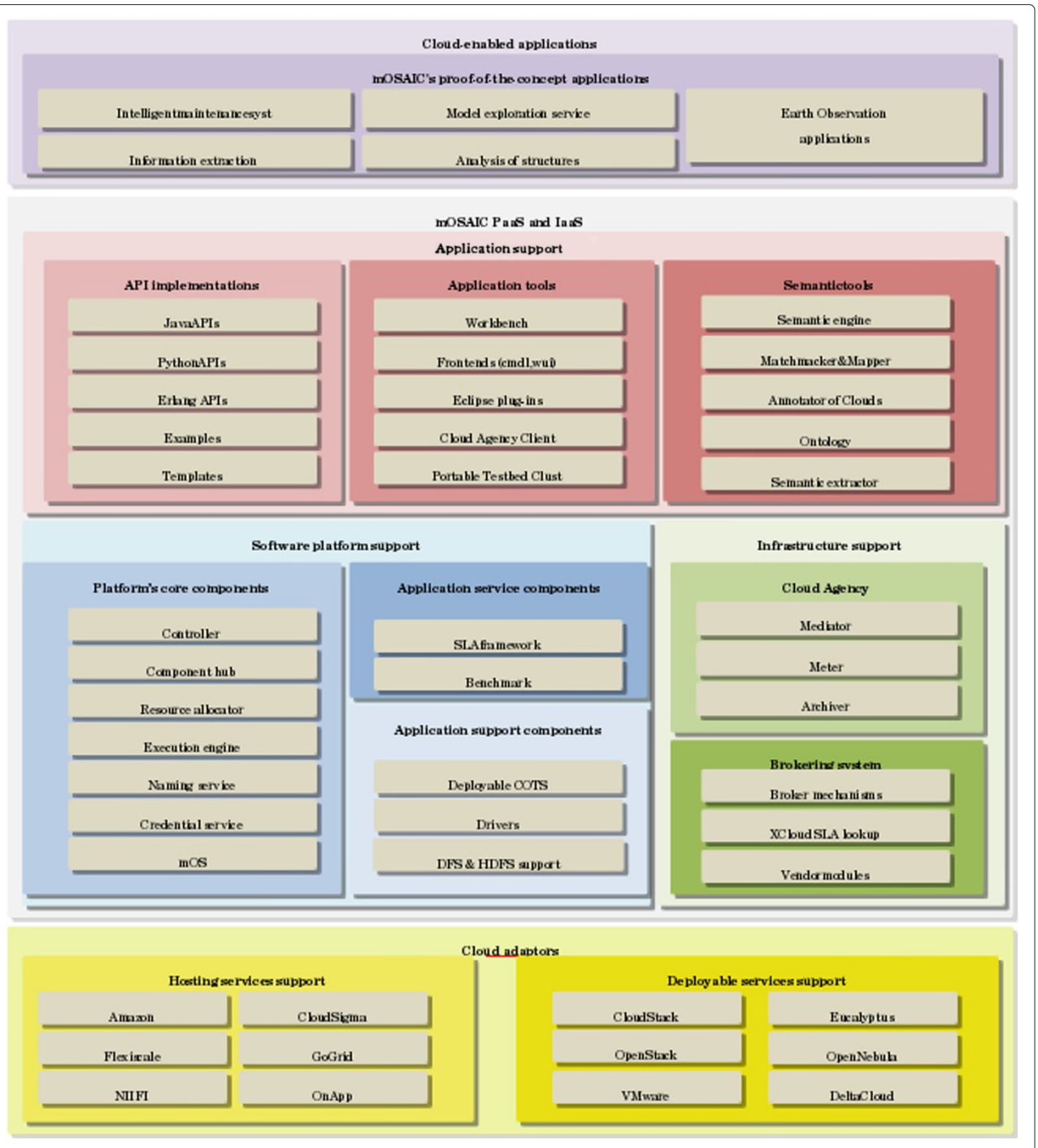

Figure 1 mOSAIC's architecture presented in [3].

bitbucket.org/mosaic. Some of them are mentioned in the next section (see the tables).

The platform that is deployable on virtual machines acquired from Private or Public Clouds includes core modules that ensure the deployment of the application, the control of the deployed components, the registration and discovery of new components. A web interface allows manual start, destruction, replication or replacement of application components without stopping and restarting the application or the platform.

The latest detailed description of the platform functionality at run-time can be found in [5]. The platform is further developed to include monitoring facilities in the frame of the EC-FP7 project MODAClouds [6] and to

improve security features and SLA compliance checks in the frame of EC-FP7 project SPECS [7].

Deployment time: cloud service representation for acquisition

The Application Tools developed throughout the project were designed to assist the developers in the deployment process by enabling the editing of the application descriptor stating the basic requirements in terms of the relationships between the application components. Another component, the Portable Testbed Cluster (PTC), allows the development and debugging of application on a desktop and then assist in its porting on a Private or Public Cloud. The Resource Allocator is able to acquire resources as described in the deployment descriptor.

The Semantic Engine and Service Discoverer support the application developer in finding the proper functionality for his or her application or the proper type of Cloud service (details in [8]). In order to tackle with the variety of terms and relationships between of them, a Cloud Resource Ontology was build (details in [9]).

The application developer is assisted in the process of Cloud service acquisition by the Cloud Agency (shortly, CA), a multi-agent system designed to support brokering and provisioning of Cloud services. The multi-agent system includes Vendor Agents representing the providers in the brokerage process. Details about the CA concept can be found in [10], while the full workflow is detailed in [11]. An overview of its architecture can be seen in Figure 2. The brokering process is based on service level agreements (details in [12]).

The Cloud Agency focuses on acquiring services for computing, storage or networking (i.e related to resources).

A Cloud resource can be present in the inputs and outputs of the CA:

1. in an abstract representation in the call for proposals for services to be acquired for an application deployment;
2. in an abstract representation in the response of the Cloud providers agents available in the platform leading to a proposal for a service level agreement;
3. in an abstract representation in the application deployment descriptor, after the approval of the service level agreement.

The interface with the Cloud service for resource acquisition is achieved through a so-called Vendor Module that is discussed in what follows.

Vendor module role and functionality

This section focuses on the Vendor Modules, as being relevant for the management system of multiple Cloud resources. The design requirements are related to the need of vendor agnosticism, the integration in the Cloud Agency, the compliance with the Cloud providers offers. While the answers to the last two requirements are specific for the solution that is build or the provider that is connected, the first one leads to an abstract level that can be of general interest, and therefore is described in details in what follows.

Supported applications

Taking a step back, we should first note that there are two business processes which are relevant in relation with the vendor agents: the resource provisioning and the resource management (Figures 3 and 4).

The Cloud Agency supports at least three kinds of applications:

1. ones that run on top of the Platform having CA as a resource provider for the Platform itself;
2. ones that run on the Cloud without Platform and CA provides provisioning and resource management as well as scaling up and down of the resources;
3. ones that include both mOSAIC API compliant components intended to run on top of the Platform and non-mOSAIC components to be serviced by the CA.

The Vendor Agents should be able to read and interpret application descriptions prepared using the Application Tools for applications addressing any of the three situations above. They are able to prepare the Cloud resources according to the application description rules. The application description together with other elements coming in place based on the user preferences and tools interactions during the resource provisioning process (e.g. SLA mechanisms) allows for the generation of a deployment descriptor. This descriptor includes all the needed information to prepare and create resources at deployment time (it is basically an artifact which is available at the end of the resource provisioning phase and is the base for the resource management phase).

A sample deployment descriptor is included in Listing 1. It mainly includes a set of descriptions for all the resource classes involved in the application to deploy. The example presents an application which needs:

1. the CA support, thus a CA Virtual Machine has to be available;
2. the Platform support, thus the Platform Control VM and Platform Execution VM should be available;
3. a storage for a platform Driver implementing a key-value storage;
4. the Web Server functionality presented as a distinct tier with all the resources at this tier being load balanced.

Listing 1 Deployment descriptor

```
<deployment_descriptor>
  <application>
    mOSAIC Application
  </application>
  <description>
    Requiring the CA,  PaaS and a Web Server
  </description>
  <tiers>
   <tier>
    <id> CA </id>
    <resource_classes>
      <resource_class  resource_type="compute" name="CA_VM">
                <provider>AMAZON</provider>
                <sla>SLA_1001</sla>
                <description>CA Virtual Machine</description>
                <url>deployed_image://CA.ami</url>
                <files>
                <file>*/*</file>
            </files>
                <run>
                <program>startCA.sh</program>
            </run>
                <max_instances>1</max_instances>
      </resource_class>
    </resource_classes>
   </tier>
   <tier>
    <id>mOSAIC Platform </id>
    <resource_classes>
      <resource_class resource_type="compute" name="PLATFORM_VM">
            <provider>AMAZON</provider>
            <sla>SLA_1002</sla>
            <description>PaaS Control VM</description>
            <url>deployed_image://PLATFORM.ami</url>
            <files>
                <file>PLATFORM/*</file>
            </files>
            <run>
                <program>init_paas.sh</program>
            </run>
            <max_instances>1</max_instances>
      </resource_class>
      <resource_class resource_type="storage" name="PLATFORM_KV_STORAGE">
            <provider>AMAZON</provider>
            <sla>SLA_1002</sla>
            <description>Platform KV storage</description>
            <url>storage://kv_bucket_name</url>
      </resource_class>
      <resource_class resource_type="compute" name="PLATFORM_EXEC_VM">
            <provider>AMAZON</provider>
            <sla>SLA_1002</sla>
            <description>PaaS Execution VM</description>
            <url>deployed_image://PLATFORM_EXEC.ami</url>
            <run>
                <program>init_exec_platform.sh</program>
                </run>
            <max_instances>5</max_instances>
```

```
      </resource_class>
    </resource_classes>
  </tier>
  <tier>
    <id>Web server</id>
    <resource_classes>
      <resource_class resource_type="compute" name="APP_WS_VM">
            <provider>AMAZON</provider>
            <sla>SLA_1003</sla>
            <description>App WS Tier VM</description>
            <url>deployed_{i}mage://WebService.ami</url>
            <files>
                <file>APP/WebService/*</file>
            </files>
            <run>
                <program>ca_monitor.sh</program>
                <program>tier1.sh</program>
            </run>
            <max_instances>3</max_instances>
            <load_balancing>
                <port>8080</port>
                <protocol>HTTP</protocol>
            </load_balancing>
      </resource_class>
    </resource_classes>
  </tier>
 </tiers>
</deployment_descriptor>
```

Use case

In their white paper "Architecture for Managing Clouds" [13], Distributed Management Task Force (DMTF) identify a series of cloud management use cases and depict these in close relation with the cloud service lifecycle, starting from NIST's definition of Cloud Computing [14], at the same time identifying relations between various actors.

Of the use cases presented by DMTF, the provisioning use case is the most important one as it essentially defines the work carried out in this paper. The provisioning use cases defines "the process of selecting, reserving, or creating an instance of a service offering" [13].

The normal steps for this use case are:

- Authentication – establishing the identity and permissions with the cloud provider;

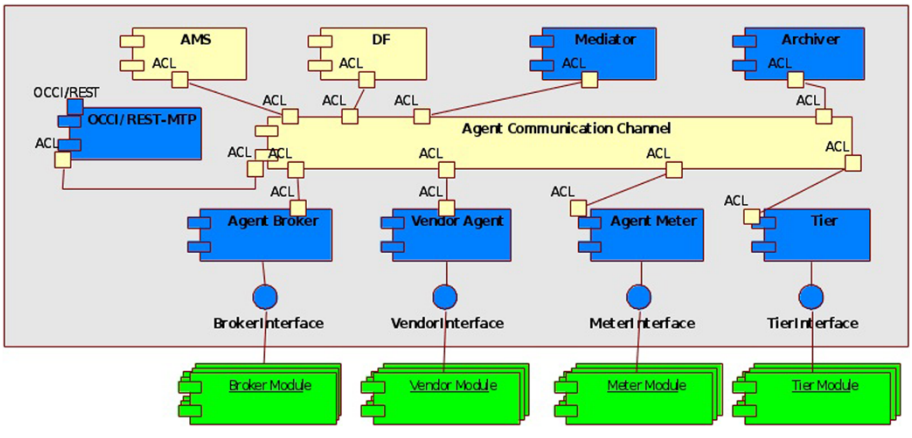

Figure 2 Cloud agency's architecture presented in [10].

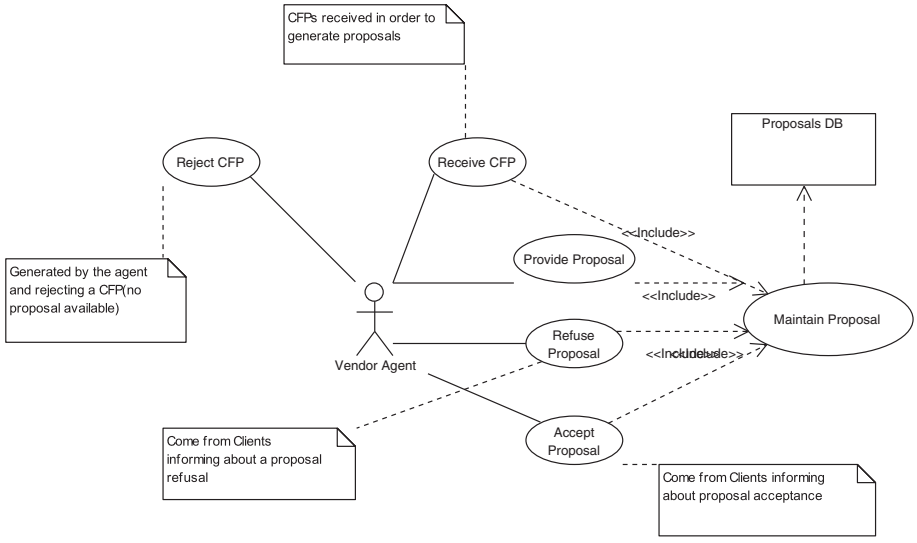

Figure 3 The resource provisioning use case.

- Offerings – evaluating and selecting from existing offerings;
- Provisioning – the actual provisioning of the desired resources;
- Provisioning monitoring – monitoring the activity of the provisioning process;
- Information retrieval – retrieving meta information about the provisioned resources;

The best example for this use case would be a company that has a cloud application composed of two components: one that runs on a public cloud and one that runs on a private cloud. When provisioning, the company must choose two cloud providers to match the components. The ease of use provided by the CA enables the company to provision the resource for its application on desired cloud vendors as long as these vendors are supported through specific Vendor Modules.

Thus, having an unified interface provides a vendor agnostic approach to resource provisioning enables provisioning without knowing the intricacies of cloud vendor APIs, as well as it allows the addition of other cloud vendors with great ease.

Describing resources

A resource class is uniquely identified by a name and its type. After resource provisioning, it is tied to a specific provider and a specific SLA. Because of the nature

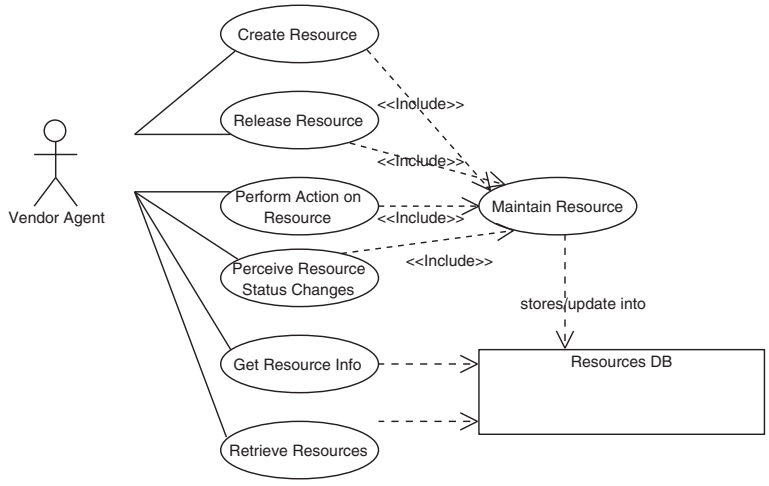

Figure 4 The resource management use case.

of SLAs, that of being cloud vendor specific, it is hard to find common ground between different vendors, and moreover it is hard to find common relationships between SLA metrics and provisioned resources. A clear example of this issue would be Amazon, the largest cloud provider, which only has 1 SLA metric: Monthly Uptime. The SLAs do not have to be the same for different resource classes. Depending of the class resources type, there are description tags which need to be specified in the class description. For example in the case of compute resources, the location of the image to use in order to create that VM is specified. In the same situation, there could be a set of files to be uploaded into that VM before using it. Also, one or more programs have to be executed at startup. All this information is present in the descriptor.

Listing 2 presents an example for the resource classes for the Platform part. There are three resource classes in the above descriptor extract:

1. Compute resources in order to run the VMs for the core Platform. Specific files have to be installed on the VMs and some programs to be run at the initialization time;
2. Compute resources in order to host the components of the Application which runs on top of the Platform. These VMs have to be properly initialized as well;
3. A storage to be used by the Platform core. It is to be attached to each core Platform VM, indication specified accordingly in the core VM descriptor.

Listing 2 Cloud Application Resource Descriptor

```
<tier>
      <id>mOSAIC Platform</id>
      <name>CA Part 2</name>
      <description>mOSAICPlatform</description>
      <resourceClasses>
        <resourceClass>
          <id>Platform_VM</id>
          <name>Platform VM</name>
          <description>Platform VM</description>
          <type>COMPUTE</type>
          <vendorId>AMAZON</vendorId>
          <slaId>
            <id>AMAZON_SLA_1342887364004_2</id>
            <cfpId>
              <id>CFP_1</id>
              <appDescriptorId>App_Dscr_Id</appDescriptorId>
            </cfpId>
          </slaId>
          <image_url>
          deployed_image://AMI_WITH_LINUX.ami?user-data=#!pkg:mosaic-node-boot
          </image_url>
          <filesToDeploy>
            <file>
              <localURL>
http://ftp.info.uvt.ro/mosaic/mos/bundle-installer/mosaic-tools-bundle-0.2.sh
              </localURL>
              <remotePath>mosaic-tools-bundle-0.2.sh</remotePath>
            </file>
          </filesToDeploy>
          <programsToRun>
            <program>sh mosaic-tools-bundle-0.2.sh standalone</program>
            <program>chroot /opt/mosaic/os</program>
            <program>/etc/init.d/mosaic start</program>
          </programsToRun>
          <firewall>
            <openPorts>
             <range>
              <min>0</min>
              <max>65535</max>
```

```
            </range>
          </openPorts>
        </firewall>
        <storage>Platform_Storage_Id</storage>
        <max_instances>3</max_instances>
      </resourceClass>
      <resourceClass>
        <id>Platform_Exec_VM</id>
        <name>Platform Execution VM</name>
        <description>Platform Execution VM</description>
        <type>COMPUTE</type>
        <vendorId>AMAZON</vendorId>
        <slaId>
          <id>AMAZON_SLA_1342887364004_2</id>
          <cfpId>
            <id>CFP_1</id>
            <appDescriptorId>App_Dscr_Id</appDescriptorId>
          </cfpId>
        </slaId>
        <image_url>deployed_image://AMI_WITH_LINUX.ami</image_url>
        <filesToDeploy>
          <file>
            <localURL>
http://ftp.info.uvt.ro/mosaic/mos/bundle-installer/mosaic-tools-bundle-0.2.sh
            </localURL>
            <remotePath>mosaic-tools-bundle-0.2.sh</remotePath>
          </file>
        </filesToDeploy>
        <programsToRun>
          <program>sh mosaic-tools-bundle-0.2.sh standalone</program>
      <program>chroot /opt/mosaic/os</program>
      <program>/etc/init.d/mosaic_exec_machine start</program>
        </programsToRun>
        <firewall>
         <openPorts>
          <range>
            <min>0</min>
            <max>65535</max>
          </range>
         </openPorts>
        </firewall>
        <max_instances>5</max_instances>
      </resourceClass>
      <resourceClass>
        <id>Platform_Storage_Id</id>
        <name>Platform KV Storage</name>
        <description>Platform KV Storage</description>
        <type>STORAGE</type>
        <vendorId>AMAZON</vendorId>
        <slaId>
          <id>AMAZON_SLA_1342887364004_2</id>
          <cfpId>
        <id>CFP_1</id>
        <appDescriptorId>App_Dscr_Id</appDescriptorId>
          </cfpId>
        </slaId>
      </resourceClass>
    </resourceClasses>
</tier>
```

To summarize it, Listing 2 contains a list of resource types that need to be provisioned (two compute resources and one storage). For the compute resources, information related to the vendor, sla, vm image, deployment information, security information are specified. Storage resource has only information related to the vendor and sla defined.

For compute resources, the maximum number of instances, open ports can be specified. Additionally, one can specify if they are load balanced using some load balancers, like in the case of the Web server tier described in Listing 1.

Generally speaking, deployment descriptors (e.g. Listing 1) contain information like: files to upload, commands to run, number of instances to start, and are mostly used in conjunction with compute resources.

The first distinction between Cloud resources is made by their traditional classification in the Cloud. We are interested in the following resource types:

1. COMPUTE : a Virtual Machine;
2. STORAGE : a volume to be attached (mounted) to a Virtual Machine;
3. NETWORK: networking for a compute resource;
4. LOAD BALANCER: a balancer associated with Virtual Machines;
5. MAP-REDUCE : a resource implementing the map reduce protocol;
6. CLUSTER: a set of COMPUTE resources being subject of auto-scaling.

A second distinction came from the fact that two compute resources (or, in general, resources of the same type) may require different credentials or are subject of different provisioning restrictions. We consider the concept of a resource class in order to cope with this distinction. In particular, resource types are important when asking for credentials and when creating resources as, for example, compute resources might require additional credentials (e.g. key pairs, ssh credentials) as opposed to storage resources. A resource identifier includes information about the resource class.

Cloud provider specifics

Different Vendor Agents are expected to have some common behavior which integrates with specific behavior. It is important to share common functions and their implementation between different Vendor Agents in order to minimize the development effort and to provide an uniform approach in the CA. The concept of Vendor Module was introduced therefore to encapsulate the specifics of Cloud providers. Such a module, pluggable into the Vendor Agent, is based on an Abstract Vendor Module entity intended to address all the common functionality of the vendor agents and their integration in the CA. The Figure 5 presents the relationship between the Abstract Vendor Module and the Vendor Module.

As the Vendor Module is a component which is intended to address the specifics of a Cloud provider in terms of resource provisioning and resource management, Vendor Modules are necessary to be developed for each Cloud service provider. An API was designed therefore to support the fast development of Vendor Modules. The API includes an abstract behavior of the Vendor Modules, definitions and implementations of all the important concepts related to resources, resources provisioning and resources management. Also the API was concerned about the deployment of the applications on the Cloud infrastructures.

Because of the specifics of the Cloud providers affect Cloud resource management, common elements had to be identified and currently reflect in the design and implementation as they take into account important elements like the provided resources by a specific vendor, the operations available on resources (like COMPUTE, VOLUME and STORAGE), the way the credentials are managed, the available APIs or administrative operations. Such common elements are discussed in what follows, for twelve providers of hosted or deployable services (the ones connected with the mOSAIC's platform).

Provider resources

As can be seen in Table 1, COMPUTE and VOLUME resources are available to all providers.

Compute operations

Table 2 basically reflects the availability of the major operations on virtual machine for all the considered Cloud providers. However, there are some variations between providers as some of them directly start the virtual machine on creation or can only attach volumes after the virtual machine creation.

Storage operations

The volume operations basically involve the ability to create and delete drives (see Table 3). Then, the drives are attached to the compute resources either at the creation time or later.

Admin operations

Table 4 includes some common administrative operations performed on virtual machine images and on the credentials as they can be made using some API. In some cases there is an ability to upload virtual machine images, but most of the providers cannot do that. This is why the current API of the Vendor Module is avoiding this step and is relying on the users to upload the appropriate images on the providers.

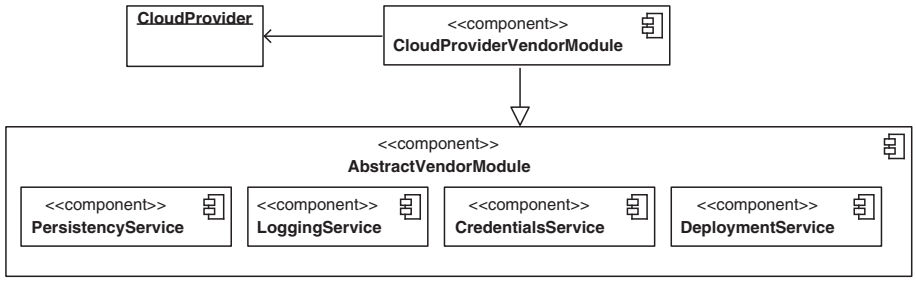

Figure 5 Vendor module.

Credentials management

The used credentials in order to manage resources vary across the Cloud providers (Table 5). In general, username/password authentication is possible. Amazon and Eucalyptus put this into the form of Access/Secret keys.

API

There is a wide range of options related to the API to use when accessing the services of a Cloud provider. The Table 6 provides a reference. The current vendor modules rely on Java libraries whenever available. In some cases there are Java libraries wrapping REST or SOAP communication.

Resource provisioning and management

The Abstract Vendor Module operations cover two important goals of the Vendor Agent: resource provisioning and resource management. The operations specific to the Vendor Module are presented in the Figure 6.

The operations outlined in the Abstract Vendor Module description falls into few categories which are not necessarily distinct:

1. public operations: are intended to be called in order to address client requests. They can be implemented at the level of the Abstract Vendor Module or Vendor Module (those also being abstract);
2. protected operations: are intended to be implemented by the Abstract Vendor Module (most of them) or by the Vendor Module (the ones which are also abstract: in italics);
3. abstract operations: are intended to be implemented by the specific Vendor Modules.

Provisioning

The resource provisioning is projected in Vendor Agents provisioning of proposals as answer to a Call for Proposal (CFP). Once a proposal is accepted, the resource classes involved in that proposal are prepared in order for resource services to be created/activated. The creation/activation of brokered SLA resources is not yet possible, as resources are intended to be created and destroyed when needed; however, for certain resource classes, the preparation step may be required, for example, to make sure some image is available on the right place on the Cloud provider environment. Resource class preparation also involves ensuring the right credentials are in place before actually managing resources (as instances of resource classes).

Management

The resource management refers here to all the operations on resources once they are provisioned. The resource management starts with the accepting of a proposal. Then, the resource classes have to be prepared by obtaining the needed credentials and by performing any required step in order for the resources of that class to be created.

Vendor module services

There are four services in relation with the Vendor Module coping with persistency, credentials, logging and serialization. These services are passed to the Vendor Modules by the Vendor Agent at the module creation time, and they can be potentially reused across Vendor Modules.

Credentials management

Different Cloud providers use different authentication policies. Also, different resource types require specific credentials in order to be created and managed. The credentials are not directly available to the Vendor Agents and therefore the agents should query them from the application deployer at the deployment time. The credentials may no longer be requested at the execution time as the agents are deployed in the Cloud and the CA is decoupled by the deployment tools. A Credentials Service is therefore intended to manage all the needed credentials of the vendor agent in relation with the resource classes and SLAs. The Vendor Module can query the credentials for a resource class defined in relation with a specific SLA in order to perform operations on that resource class or its instances.

Persistence management

There are two kinds of entities which are identified to be subject of persistence: vendor proposals (SLAs) and

Table 1 Cloud resources

	Amazon	Flexiscale	CloudSigma	Eucalyptus	OpenNebula	NIIFI Cloud	OpenStack	VMware	OnApp	GoGrid	CloudStack	DeltaCloud
Compute	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Storage	✓			✓			✓		✓	✓	✓	✓
Volume	✓	✓	✓	✓	✓	✓*	✓	✓		✓	✓	
MapReduce	✓											
Databases	✓											
Load balancing	✓						✓	✓	✓	✓	✓	
Firewall	✓	✓		✓			✓	✓		✓	✓	
Clusters	✓			✓								

*Not persistent (created and destroyed with the appropriate VM).

Table 2 Compute operations

	Amazon	Flexiscale	CloudSigma	Eucalyptus	OpenNebula	NIIFICloud	OpenStack	VMware	OnApp	GoGrid	CloudStack	DeltaCloud
Create	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Destroy	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Start	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓*
Stop	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓*
Reboot	✓	✓	✓		✓		✓	✓	✓	✓	✓	✓*
Attach Volume	✓	o	o	✓	o		✓	✓	✓	✓	✓	
Detach Volume	✓	✓	✓	✓			✓	✓	✓	✓	✓	
List details	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓

o: onCreate; *depends on particular driver load.

Table 3 Storage operations

	Amazon	Flexiscale	CloudSigma	Eucalyptus	OpenNebula	NIIFI Cloud	OpenStack	VMware	OnApp	GoGrid	CloudStack	DeltaCloud
Create	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Delete	✓	✓	✓	✓			✓	✓	✓	✓	✓	✓

Table 4 Administrative operations

	Amazon	Flexiscale	CloudSigma	Eucalyptus	OpenNebula	NIIFI Cloud	OpenStack	VMware	OnApp	GoGrid	CloudStack	DeltaCloud
Upload Image	✓	✓	✓	✓			✓	✓	✓		✓	✓
Bundle Image	✓			✓					✓			
Download Image	✓		✓	✓			✓	✓	✓		✓	
Delete Image	✓	✓	✓	✓			✓	✓	✓		✓	✓
Manage Keys	✓	✓	✓	✓			✓	✓	✓		✓	✓

Table 5 Credentials management

	Amazon	Flexiscale	CloudSigma	Eucalyptus	OpenNebula	NIIFI Cloud	OpenStack	VMware	OnApp	GoGrid	CloudStack	DeltaCloud
Private/Public Key	✓						✓		✓		✓	✓*
Username/Password		✓	✓		✓	✓	✓	✓	✓	✓	✓	✓*
Access/Secret key	✓		✓	✓						✓	✓	✓*
External Authentication					✓						L	✓*

L - based on LDAP; *depends on the driver load and provider configuration.

Table 6 API

	Amazon	Flexiscale	CloudSigma	Eucalyptus	OpenNebula	NIIFI Cloud	OpenStack	VMware	OnApp	GoGrid	CloudStack	DeltaCloud
REST	✓		✓	✓	✓	✓	✓		✓	✓	✓	✓
Java	✓			E	✓		✓	✓		✓	E	E
SOAP	✓	✓		✓								
Other	R,H,N				R,X	R,L						R,C
OCCI compliant					✓							✓
AWS compliant	✓			✓	P		✓				✓	P

E - external libs; P - EC2; R - Ruby; H - PHP: N - .NET; L - CLI; C - C/C++; X - XML-RPC.

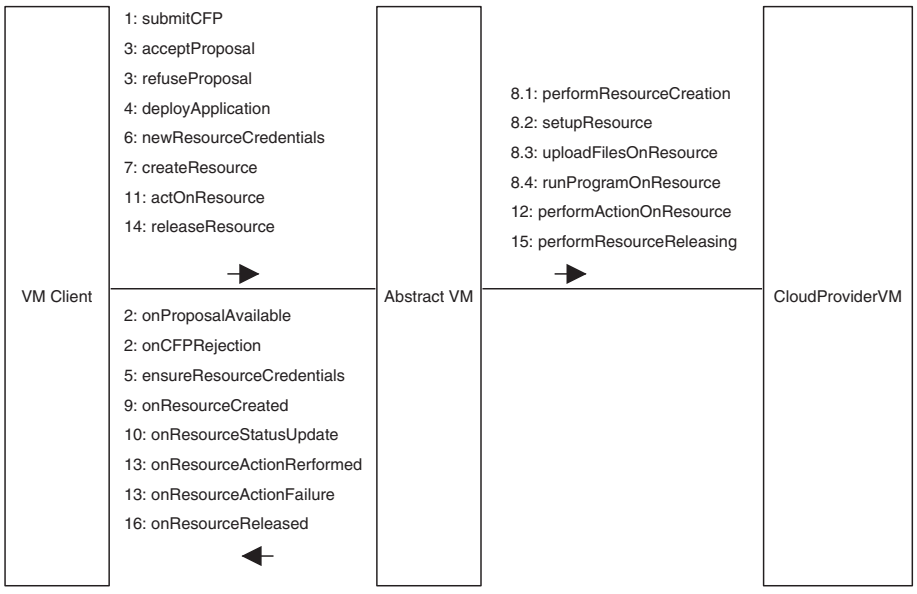

Figure 6 Vendor module communications.

Cloud resources. The Vendor Agents can refer to vendor proposals both when provisioning resources and when using resources. Therefore the proposals have to be stored and their status maintained in some persistent area. The proposals are related to the submitted CFP which may be stored as well. Once an SLA was agreed, the Vendor Agents maintain information about the resources classes and how they can be referred by the Cloud provider. For example, the AMAZON's Vendor Agent may keep a mapping between an AMI image and the resource class it is related to as this information comes from the deployment descriptor or as the Vendor agent itself may infer. When starting to create resources, the identifiers of these resources are also stored and mapped to resource descriptions and resource classes in order for future requests to be satisfied. The status of the resources is maintained as well based on the received requests from the clients and the received updates from the Cloud provider.

Logging

Logging events is sometimes useful and necessary. The Vendor Agent provides the Vendor Module with a logging service in order for the significant events to be logged at the level of the agency. There are different logging levels, in a similar way the Java language itself provide logging support.

Serialization

The CA uses its own protocol and serialization elements in order to transport messages and their parameters between agents. Once a message arrives to the Vendor Agent, it has to be split and its components transferred into an object oriented form in order for the agents' services' methods to be called. The serialization service is therefore tasked with message composition/decomposition as the objects are transmitted or received from the CA API.

Note that currently in the open-source repository of mOSAIC, on bitbucket.org, the codes of the Vendor Agents and Modules are available in `mosaic-vendor-vendors` sub-repository and are being maintained by their authors.

Support for brokering

Within the mOSAIC's Cloud Agency, it is the responsibility of the Vendor Agents to create an (SLA) offer in reply to a Call-for-proposal received from the user. These offers should contain a service description (hardware parameters of the virtual machines, storage, network, and/or any additional provider specific SLA related information) and a price (e.g. hourly fee of the offered infrastructure). The possible hardware configurations, and their price change from provider to provider. In the current implementation of the Vendor Modules, this information is either hardcoded in the corresponding module, or provided as configuration files which are read by the Vendor Modules. The main disadvantage of this solution is that whenever a provider changes its offered services, or its prices, the corresponding vendor module, or its configuration file has to be updated.

A more seamless integration could have been achieved, if the Vendor Modules could directly query the available hardware configurations and their prices from the

providers. This method could be applied, if the providers exposed public Web services to publish such information. Unfortunately, none of the providers support this functionality entirely. The most complete functionality is provided by Flexiscale and CloudSigma. These providers publish Web service calls to query the available hardware configurations and the prices of certain services. For instance, one can query the price of the available VM configurations from Flexiscale, and the price of some network and software (license) resources from CloudSigma. The problem with these providers is that they do not expose these calls as a public service, the user has to have a username and a password to make such requests. Since brokering happens before deployment, the user will typically not have a password to these providers at brokering time.

An even lower level of service is provided by RackSpace (using the OpenStack API) and Arctur-1 (using the VMWare API). These providers do expose the available hardware configurations, but there is no way of querying the corresponding prices. Unfortunately, we expect the prices to change more frequently than the offered hardware configurations, therefore the configurations of the corresponding vendor modules have to be changed anyway. Finally, among the integrated vendors, Amazon and GoGrid was found to lack any support for automated brokering. Neither the offered hardware configurations, nor their prices can be queried. Until this situation changes, these vendor modules have to rely on the current off-line approach.

Migration

The CA deployment procedure involves an initial situation when the CA agents are deployed into a local environment, on the user's machine and a follow-up situation when the agents are migrated into a Cloud environment as part of the application deployment procedure.

There are few elements which concerns the Vendor Agents as part of this process. In the initial phase, the Vendor Agents only contribute to resource provisioning. The resource creation and the resource management is subject of the second phase when the agents are deployed in the Cloud. During the first phase the Vendor Agents acquire credentials for different resource classes and create content in the Proposals DB as it results from the provisioning process. The Resources DB may also be populated with resource class details. Apart of the information the Vendor Agents themselves directly manage, there are a set of application specific elements which have to be moved into the Cloud context in order to be accessible to the Vendor Agents at the resource creation time. Such elements include files as described in the Application Types to Support section.

The Vendor Agents are directly interested about such considerations as they are actually responsible to move the application and all the needed elements from the local context to the Cloud environment.

Proof-of-concept implementation

In order to make a preliminary validation of the design and to have a reference implementation, a SampleVendor Module was developed. Additional development was made in order to interact with the PTC (Portable Testbed Cluster) infrastructure by writing a specific Vendor Module. After the initial validation on PTC, the remainder of the vendor specific modules were developed, current implementation of the modules covering several Cloud providers including Amazon, CloudSigma, Eucalyptus, Flexiscale, GoGrid, Niifi Cloud, OpenNebula, OpenStack, PTC and VMWare and can be found on the BitBucket repository (https://bitbucket.org/mosaic/mosaic-agency-vendors/).

In order to support the development and testing of Vendor Modules, an Eclipse based tool was developed. This tool facilitates the creation and editing of the descriptors that are used throughout the provisioning and management of Cloud resources as well as enable the workflow patterns under which the Vendor Modules have been designed as follows:

- Cloud Application – enables creation and editing of the description of the Cloud application with its tiers and resources (Figure 7);
- Call for Proposal – enables generation of CfP from an existing Cloud Application description and allows customization of each resource attributes (Figure 8);
- Service Level Agreements – enables the brokering (simple resource brokering) of the CfP by sending it to enabled Cloud vendor modules and receiving from each the suitable SLAs and enables their visualization, acceptance and rejection (Figure 9);
- Resource Deployment Descriptor – upon SLA acceptance, this file is automatically created and its editing enables the preparation and creation of resources at the execution time.
- Resource management – based on the deployment descriptor, the tool allows the creation of one or several instances of brokered Cloud resources as well as their management (resource information, starting and stopping resources etc.).

Related work

As identified in [1] the Multi-Cloud middleware can be library-based or service-based. In the first case, a library facilitates a uniform way to access multiple services and resources, as well as the provisioning of services and

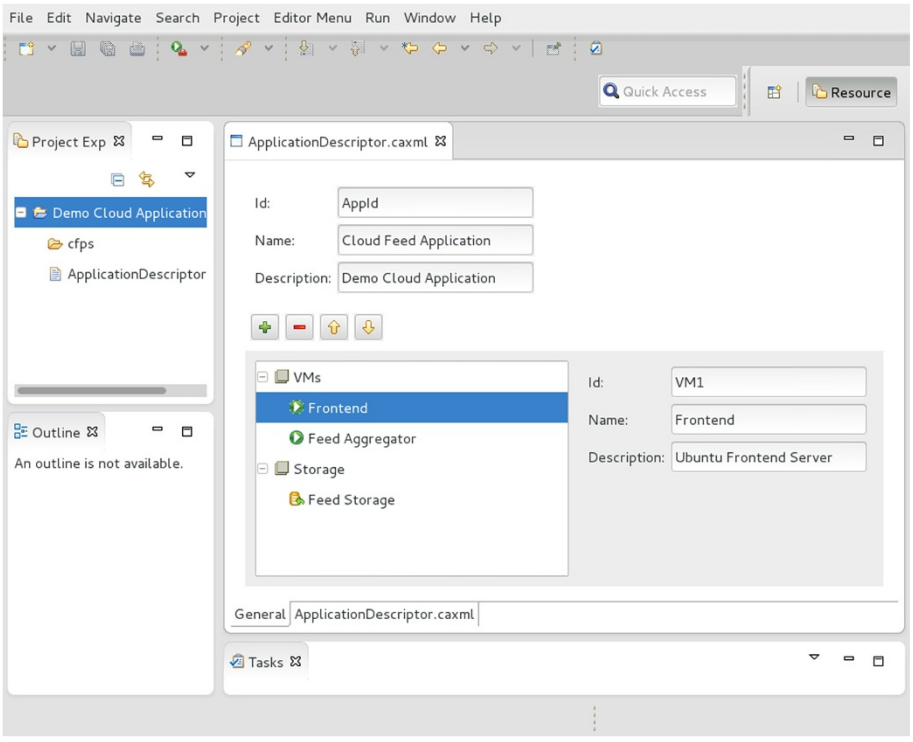

Figure 7 Application descriptor.

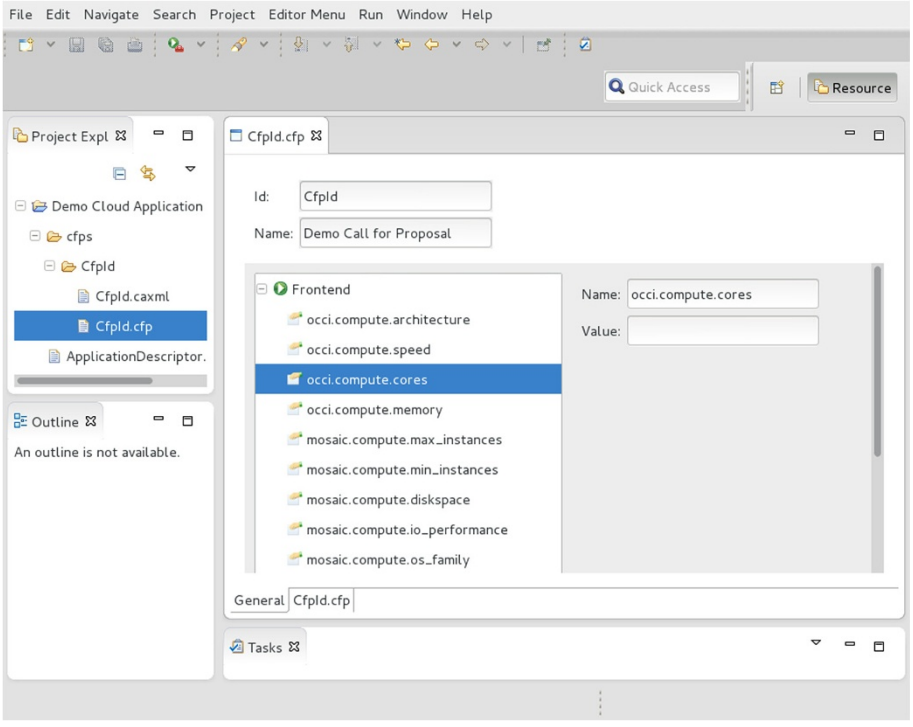

Figure 8 Call for proposal.

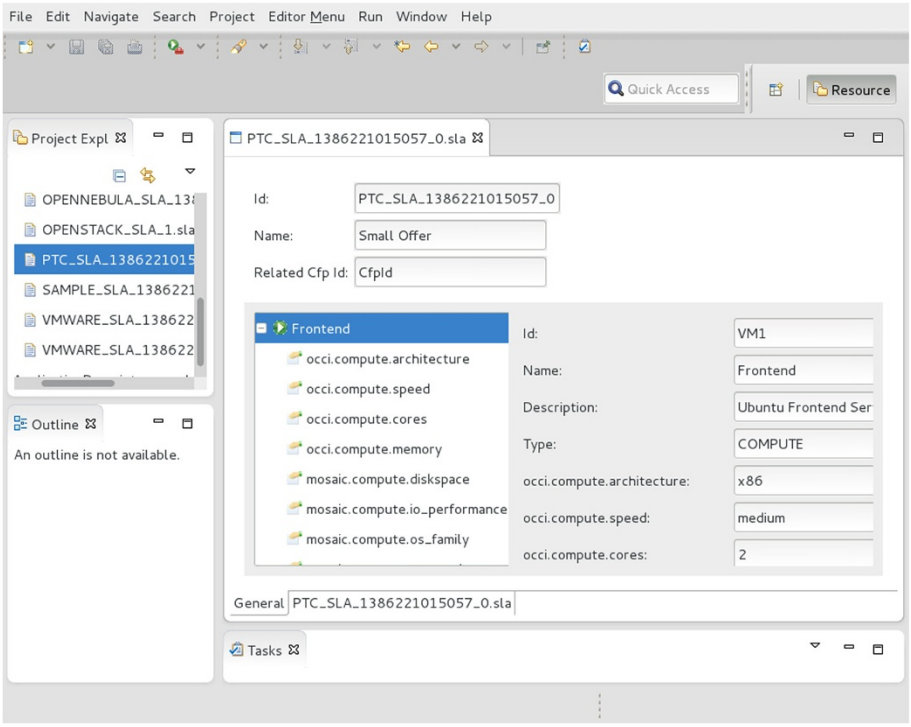

Figure 9 SLA inspection.

resources from multiple Clouds. In the second case, a special service is offering brokerage between multiple Clouds based on clients' service level agreements or provisioning rules and performs deployment, execution and monitoring.

The most known library-based approaches are jclouds, libcloud, and δ-cloud. Jclouds is an open source Java library designed to support the portability of Java applications, which allows the uniform access to the resources from various IaaS providers (jclouds.apache.org). Libcloud is a Python library that abstract the differences among the programming interfaces of Cloud services (libcloud.apache.org). δ-cloud is a REST-based API written in Ruby which allows also the connections to various Cloud resources (deltacloud.apache.org). These libraries are offering the common denominator of the underlying services, and are loosing their individuality [15]. Moreover, they are compliant with portability requirement only for off-line case (i.e. stopping the application in the current Cloud, and restarting it entirely and from the beginning in another Cloud). A more complex case is that in which the relocated application is decomposed and relocated over a new set of Clouds. Such use cases have been rarely reported until now.

In mOSAIC each of these libraries can be used as Driver interfaces with the Cloud services. The acquisition of the resources is not a subject for these libraries, nor for the Driver of mOSAIC (ensure only the second level of the interface with the Cloud service).

We classify the service-based approach for Multi-Cloud in two categories: hosted or deployable.

The most known hosted services are the commercial offers of RightScale, Kavoo and Enstratus. RightScale is offering a management platform for the control and administration of deployments in different Clouds (www.rightscale.com). Its Multi-Cloud Engine is able to broker capabilities related to virtual machine placement in Public Clouds. Kaavo allows the management of distributed applications and workloads in various Clouds (www.kaavo.com). Enstratius, allows the management, monitoring, automation and governance of resource consumption based on the services from various Cloud providers (www.enstratius.com).

Several deployable services are results of open-source projects like mOSAIC, Aoleus, Cloud4SOA or OPTIMIS. Aeolus is an open-source Cloud management software written in Ruby and provided for Linux systems by RedHat and it is based on the δ-cloud library (aeolusproject.org). Cloud4SOA is dealing with portability of applications between PaaSs by relying upon semantic technologies (www.cloud4soa.eu). OPTIMIS offers a deployable Platform (-as-a-Service) that allows Cloud service provisioning and the management of the life-cycle of the services (www.optimis-project.eu). It is more comprehensive

than mOSAIC PaaS in terms of facilities for brokerage and run-time control, while mOSAIC offers more complex tools to support the application developers.

The Cloud brokers are playing an important role in both Multi-Cloud. The most known independent Cloud brokers are: SpotCloud, Scalr and Stratos. SpotCloud provides a marketplace for infrastructure service and a matching service with the client requirements (www.spotcloud.com). Scalr provides deployment of virtual machines in various Clouds and includes automated triggers to scale up and down (www.scalr.com). Stratos offers single sign-on and monitors resource consumption and the fulfillment of service level agreements and offers autoscaling mechanisms (wso2.com/cloud/stratos). None of these brokers for Multi-Cloud are exposing their internal interfaces for the Cloud resource acquisition. Moreover, these brokers are not offering a complete solution for a Multi-Cloud, with the full stack from software development tools to run-time control; mOSAIC is trying to offer a proof-of-concept of such full stack.

A Multi-Cloud enabler is invited to follow the current Cloud standards. The current emerging standards, like OCCI [16-18], CDMI [19], CIMI [20,21], or TOSCA [22], along with others identified in [23] are still not adopted on large scale. One reason is their limited scope: at IaaS level, not yet for PaaS level [24].

In order to cope with the possible large adoption of the OCCI as standard for managing virtual machine mOSAIC is compliant with OCCI in the Call for Proposals [10]. Additional efforts were made for having a WS-Agreement [25,26] and SLA@SOI [27] compliant versions of the Call for Proposal and Service Level Agreement formats.

Conclusions

The variety of the Cloud services interfaces is a challenge to be dealt with by the Multi-Cloud resource management systems. Until standards in what concern these interfaces are are not adopted, practical approaches need to be found.

We presented one of such approach that was adopted by an open-source platform available as a deployable service and which intends to offer a proof-of-concept in what concerns the portability of Cloud-enabled applications. The approach is relying on the modularity of the platform architecture. The interfaces are different at the design, at run-time and at deployment stages.

Unlike other solutions which provide similar functionality, our focus was to provide an open source, modular solution which can be easily integrated and used due to unified interfaces without the unnecessary dependencies other solutions require.

Our approach alleviates developers of knowing the intricacies particular to cloud vendors (interfaces, APIs), thus dropping cloud application development time and allowing them to focus on more important aspects of their applications.

The focus of this paper was put on the deployment stage in which the variety of interfaces has the highest impact. We proposed to use Vendor modules to make the connection with the particular services which are following a given pattern in their description. Such modules are integrable in the broker system that was reported earlier. Moreover we proposed a pattern for describing the Cloud resource and the Cloud application that is compatible with current emerging standards, as proved by the proof-of-concept editor for application and resource description, shortly described in this paper.

Competing interests

The authors declare that they have no competing interests.

Authors' contributions

Approach for an unified interface with the Cloud services from multiple providers at deployment stage. Proof-of-concept implementation of the interface integrated in a brokerage system for Multi-Clouds. All authors read and approved the final manuscript.

Acknowledgments

The research reported in this paper was partially supported by Romanian grant PN-II-ID-PCE-2011-3-0260 (AMICAS), and refers to several parts of the platform developed in the frame of European Commission FP7-ICT-2009-5-256910 grant (mOSAIC). The tables that are included in this paper are improvements of the ones presented in the mOSAIC deliverables D2.5, D2.6 and D2.11. The team that has elaborate them is far from being reflected in the list of authors of this paper. We express our thanks in this context to Mariano Cecowksi and Miha Stopar (Xlab), Adrian Copie (Institute e-Austria Timişoara), Salvatore Venticinque (Second University of Naples), Támás Máhr (AITIA), Petr Škoda (Brno University of Technology), Jernej Južna and Vlado Stankosvki (University of Ljubljana).

Received: 6 December 2013 Accepted: 14 April 2014
Published: 12 May 2014

References

1. Grozev N, Buyya R (2012) Inter-cloud architectures and application brokering: taxonomy and survey. Software Pract Ex. doi:10.1002/spe.2168
2. Şandru C, Petcu D, Munteanu VI (2012) Building an open-source platform-as-a-service with intelligent management of multiple cloud resources. In: 2012 IEEE Fifth International Conference on Utility and Cloud Computing (UCC), pp 333–338. doi:10.1109/UCC.2012.54
3. Petcu D, Martino B, Venticinque S, Rak M, Mahr T, Lopez G, Brito F, Cossu R, Stopar M, Sperka S, Stankovski V (2013) Experiences in building a mosaic of clouds. J Cloud Comput Adv Syst Appl 2(1): 12. doi:10.1186/2192-113X-2-12
4. Petcu D, Macariu G, Panica S, Craciun C (2013) Portable cloud applications-from theory to practice. Future Generat Comput Syst 29(6): 1417–1430. doi:10.1016/j.future.2012.01.009
5. Petcu D, Panica S, Crăciun C, Neagul M, Şandru C (2013) Cloud resource orchestration within an open-source component-based platform as a service. Concurrency Comput Pract Ex. doi:10.1002/cpe.3175
6. Ardagna D, Di Nitto E, Mohagheghi P, Mosser S, Ballagny C, D'Andria F, Casale G, Matthews P, Nechifor C-S, Petcu D, Gericke A, Sheridan C (2012) Modaclouds: A model-driven approach for the design and execution of applications on multiple clouds. In: 2012 ICSE Workshop on Modeling in Software Engineering (MISE), pp 50–56. doi:10.1109/MISE.2012.6226014
7. Rak M, Luna J, Petcu D, Casola V, Suri N (2013) Security as a service using an sla-based approach via specs. In: 2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom), vol 2, pp 1–6, doi: 0.1109/CloudCom.2013.165

8. Cretella G, Di Martino B (2012) Towards a semantic engine for cloud applications development. In: 2012 Sixth International Conference on Complex, Intelligent and Software Intensive Systems (CISIS), pp 198–203. doi:10.1109/CISIS.2012.159
9. Moscato F, Aversa R, Di Martino B, Fortis T, Munteanu V (2011) An analysis of mosaic ontology for cloud resources annotation. In: 2011 Federated Conference on Computer Science and Information Systems (FedCSIS), pp 973–980
10. Venticinque S, Tasquier L, Di Martino B (2012) Agents based cloud computing interface for resource provisioning and management. In: 2012 Sixth International Conference on Complex, Intelligent and Software Intensive Systems (CISIS), pp 249–256. doi:10.1109/CISIS.2012.139
11. Venticinque S, Şandru C (2013) Agents based deployment of heterogeneous iaas clouds. Int J Comput Sci Eng 9. http://www.inderscience.com/info/ingeneral/forthcoming.php?jcode=ijcse.
12. Rak M, Aversa R, Venticinque S, Martino B (2012) User centric service level management in mosaic applications. In: Alexander M, et al. (eds) Euro-Par 2011: Parallel Processing Workshops Lecture Notes in Computer Science, vol. 7156. Springer, Berlin, pp 106–115. doi:10.1007/978-3-642-29740-3_13
13. Distributed Management Task Force (2010) Architecture for managing clouds. Distributed management task force. http://dmtf.org/sites/default/files/standards/documents/DSP-IS0102_1.0.0.pdf
14. NIST (2011) Cloud Architecture Reference Models: A Survey. http://collaborate.nist.gov/twiki-cloud-computing/pub/CloudComputing/Meeting4AReferenceArchtecture013111/NIST_CCRATWG_004v2_ExistentReferenceModels_01182011.pdf
15. Bermbach D, Kurze T, Tai S (2013) Cloud federation: effects of federated compute resources on quality of service and cost. In: 2013 IEEE International Conference on Cloud Engineering (IC2E), pp 31–37. doi:10.1109/IC2E.2013.24
16. Nyrén R, Edmonds A, Papaspyrou A, Metsch T (2011) Open Cloud Computing Interface - Core. http://www.ogf.org/documents/GFD.183.pdf
17. Metsch T, Edmonds A (2011) Open Cloud Computing Interface - Infrastructure. http://www.ogf.org/documents/GFD.184.pdf
18. Metsch T, Edmonds A (2011) Open Cloud Computing Interface - RESTful HTTP Rendering. http://www.ogf.org/documents/GFD.185.pdf
19. Storage Networking Industry Association (SNIA) (2012) Cloud Data Management Interface. http://snia.org/sites/default/files/CDMIv1.0.2.pdf
20. Distributed Management Task Force (DMTF) (2012) Cloud Infrastructure Management Interface - Common Information Model (CIMI - CIM). http://www.dmtf.org/sites/default/files/standards/documents/DSP0264_1.0.0.pdf
21. Distributed Management Task Force (DMTF) (2013) Cloud Infrastructure Management Interface (CIMI) Model and RESTful HTTP-baseds Protocol. http://www.dmtf.org/sites/default/files/standards/documents/DSP0263_1.1.0.pdf
22. Organization for the Advancement of Structured Information Standards (OASIS) (2013) Topology and Orchestration Specification for Cloud Applications. http://docs.oasis-open.org/tosca/TOSCA/v1.0/cs01/TOSCA-v1.0-cs01.pdf
23. Harsh P, Dudouet F, Cascella RG, Jégou Y, Morin C (2012) Using open standards for interoperability - issues, solutions, and challenges facing cloud computing. CoRR abs/1207.5949
24. Lewis GA (2013) Role of standards in cloud-computing interoperability In: 2013 46th Hawaii International Conference on System Sciences (HICSS), pp 1652–1661. doi:10.1109/HICSS.2013.470
25. Andrieux A, Czajkowski K, Dan A, Keahey K, Ludwig H, Nakata T, Pruyne J, Rofrano J, Tuecke S, Xu M (2011) Web Services Agreement Specification (WS-Agreement). http://www.ogf.org/documents/GFD.192.pdf
26. Venticinque S, Negru V, Munteanu VI, Sandru C, Aversa R, Rak M (2012) Negotiation policies for provisioning of cloud resources. In: Filipe J, Fred ALN (eds) 2012 4th International Conference on Agents and Artificial Intelligence (ICAART). SciTePress, pp 347–350. http://www.bibsonomy.org/bibtex/29a327c46d508a37eab3155ca147ac746/dblp.
27. Kearney KT, Torelli F, Kotsokalis C (2010) Sla*: An abstract syntax for service level agreements. In: 2010 11th IEEE/ACM International Conference on Grid Computing (GRID), pp 217–224. doi:10.1109/GRID.2010.5697973

doi:10.1186/2192-113X-3-3
Cite this article as: Munteanu *et al.*: **Multi-cloud resource management: cloud service interfacing.** *Journal of Cloud Computing: Advances, Systems and Applications* 2014 **3**:3.