Abstract—Usage scenarios involving services from multiple
Clouds, like in Federations or Markets of Clouds, are im-
posing the development of complex middleware to hide the
heterogeneity at the underlying Cloud technologies and services
in order to attract Cloud application developers. This paper
points towards the specific modules of a recent open-source
and Cloud-vendor agnostic platform-as-as-service which are
using artificial intelligence methods and automatic computing
techniques. A particular attention is payed to those modules
of the underlying multi-agent system that are ensuring the
connection to different Cloud services.

Keywords—platform-as-a-service; agents; automatic cloud

I. INTRODUCTION

Designed to offer rapid access to a large pool of available
hardware and software resources to a large variety of users,
and highlighting the benefits of elasticity in terms of costs,
the Cloud computing has been fast adopted by business
and academic communities. The preferred levels of entries
are infrastructure-as-a-service and software-as-a-service and
therefore a huge number of such services are currently
offered. This huge number as well as the lack of world-
wide adopted interoperability and portability standards or
protocols specific for this field are partially contributing to
the so-called vendor lock-in problem. We point in what
follows only two consequences, which are acting as mo-
tivating engines for using Cloud services from multiple
providers and involving automation and intelligent selection
methods. First we should note that the apparently unlimited
scaling opportunity is yet limited to the capacity of a certain
resource provider, unless the provider agrees to act in a
federation. Federations of Clouds (named also InterClouds
or Cross-Clouds), based on agreements between Cloud
providers, are still in infancy and an automated management
of the resources from multiple providers’ Clouds are not yet
reported. On another hand the selection of the best fitted
place to deploy a Cloud application is a complex issue in
a Market of Clouds (named also Sky Computing), based
on vendor-independent brokers, and application developers
should be assisted in his or her decision by semi-automated
tools including knowledge-based selection methods.

Conceptually designed as middleware for developers, as-
sisting them in the connection to and consuming of Cloud
resources, a Platform-as-a-service (PaaS) should hide the
complexity of intelligent selection procedures and the au-
tomated management of the resources allocated to a certain
application. However, already in the design process, most of
the current available PaaSs were already vendor dependent,
as relying upon their own resources or proprietary tools.

In this context and following the belief that an open-
source and vendor-agnostic middleware for developing and
deploying Cloud enabled applications should have at least
for scientific communities and small enterprises the impact
that Globus had for Grid computing, or PVM for parallel
computing, the mOSAIC project consortium (www.mosaic-
cloud.eu) has recently proposed and developed an open-
source PaaS focusing on ensuring the portability of applica-
tions consuming Cloud resources from Private or Public
Clouds (the acronym stands for Open-source API and Plat-
form for multiple Clouds). In order to achieve its goal to
serve application developers, the mOSAIC PaaS relies upon
artificial intelligence methods in the different procedures,
like in the selection of the Cloud resources to be consumed
(based on a multi-agent system). Programmable control
of the allocated resources, even from multiple providers,
is subject for the recently proposed vendor-agnostic API.
Taking into account its deployable and open-source nature,
features that are nowadays rarely encountered in PaaS mar-
ket, the mOSAIC PaaS is further develop as solution for
Cloud resource providers in the frame of AMICAS project
(http://amicas.hpc.uvt.ro). Practical approaches in what con-
cern automatic computing procedures like scheduling and
elasticity in heterogeneous environments were recently pro-
vided: the scheduling of application components requiring
high availability is subject of heuristics, while agents are
used in a vendor-agnostic auto-scaling procedure.

In this paper we present a short overview of the mOSAIC
PaaS from the point of view of the intelligent and automated
management procedures of the Cloud resources. Previous
papers about mOSAIC, from the last two years, have re-
ported the design and functionality of different architectural
modules. In this paper we put a special focus on the latest
developed modules which are allowing the connection to
different Cloud resources to be consumed.

II. SHORT OVERVIEW OF mOSAIC’S PAA S

Triggered by the need to offer a solution for the porta-
bility problem of Cloud enabled applications, mOSAIC’s
PaaS was designed to be an open-source and deployable

Building an Open-source Platform-as-a-Service
with Intelligent Management of Multiple Cloud Resources

Călin Şandru, Dana Petcu, Victor Ion Munteanu
Institute e-Austria Timişoara and West University of Timişoara, Romania
The developer of an application can currently describe the application in Java, Python, Erlang or Node.js following the mOSAIC’s API recommendations (general ones, not restricted to the three proof-of-the-concept languages) 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 (special feature in mOSAIC, instead at a larger granularity level, at virtual machine level, as usual), if the component is able to scale (independent from other components, using message passing instead synchronous communications etc). An event-driven programming style has been adopted to reduce the network traffics. The connection with the Cloud resources is done to the level of the API through the so-called Cloudlets and Connectors, first ones expressing the reaction of the application to the events related to Cloud resource consumption, second ones being 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). Details about the basic concepts and performance results can be found in [8], while simple examples of API usage in Java are provided in the documentation found at http://developers.mosaic-cloud.eu.

The Application Tools are assisting the developers in the deployment process by offering tools for the application descriptor (stating e.g. the basic requirements in terms of the relationships between the application components), control interfaces for the deployed applications (e.g. web interface allowing to manually stop, start, replace components without stopping the entire application), as well as an innovative component, the Portable Testbed Cluster, allowing the developed and debugging of application on a desktop or on local premises (using VirtualBox). Video demonstrations of the functionalities of these modules are available on YouTube (key-phrase ‘mOSAIC cloud computing’).

The Software Platform includes the core modules of the PaaS, that are ensuring the packaging of the application and deployment in virtual machines acquired in Private or Public Clouds, the control of the deployed components, the registration and discovery of new components, or the credential management. The interoperability service is acting 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. for message passing protocol AMQP, for Riak key value store, for Hadoop distributed file system, or for Amazon S3). mOS is the key module that is ensuring the platform functionality on virtual machines from different Cloud providers (with Xen, KVM or LXC support, tested until now on CentOS, Ubuntu, Suse, Redhat). Deployable open-source technologies (like RabbitMQ as message queuing system, Riak as key-value store, Jetty as web server) are used also as Cloud resources available on the provisioned virtual machines running the mOSAIC platform. The open-sources codes of the latest stable version are provided at https://bitbucket.org/mosaic.

In what concerns the particular components of the Software Platform that are making a step forward in the direction of automated management of multiple Cloud resources (subject of AMICAS extension of mOSAIC’s PaaS as support for Cloud providers), we point here towards the particular scheduler and scaler. The scheduler concepts are exposed in [6]: it is designed to support highly-available and long-running
component-based applications on top of multiple Clouds. The virtual machines or components failures are covered by replication and the optimal number of components deployed on several VMs takes into account the running costs; genetic algorithms are used to solve the complex problem. The generic Cloud-agnostic auto-scaler is exposed in [2]: it is designed to be fault tolerant and itself scalable. It is based on a P2P overlay network of automatic services (one for each VM) that are able to make scaling decisions in a decentralized manner.

The application developer is assisted in the process of finding the proper resources by a Cloud Agency (CA), a multi-agent system designed to support brokering, provisioning, monitoring and reconfiguration of Cloud resources (details, for example, in [12]). The brokering process is based on service level agreements (details in [9]) and the result is an application deployment descriptor; the definition of cloud resource attributes (parameters) is based on the OCCI notation (occi-wg.org). In order to tackle with the variety of terms and relationships between of them, a Cloud Ontology was build early in the project (details in [7]). The Semantic Engine and Service Discoverer are further supporting the application developer to find the proper functionality for her or his application or the proper type of Cloud resource (details in [3]).

Simple examples of Web applications are available in the open-source repository. Complex proof-of-the-concept applications emerging from the needs of both scientific and commercial communities are under development. A first report about these applications can be found in [5].

mOSAIC scientific publication list from the project web site includes further innovation reports at module levels. Each individual module has been evaluated versus the most closest technical solution. In what follows we are briefly pointing the main conclusions.

III. RELATED WORK

In what concerns the mOSAIC’s open-source PaaS, several deep comparisons were done in previous papers (e.g. in [8]). We point here the fact that the most similar approach is VMWare’s Cloud Foundry (www.cloudfoundry.com), developed in the same time with mOSAIC, which is an open-source PaaS targeting also application portability from Private to Public Clouds. VMWare has support for Java, Ruby, Node.js and Groovy, while mOSAIC supports Java, Python and Erlang. The data management of Cloud Foundry is related to MongoDB, SQLFire, PostgreSQL and Redis, while in mOSAIC, Riak, CouchDB, Redis, MySQL and HDFS are targeted. RabbitMQ is used by both for messages passing system. Beyond a command line interface, as provided by Cloud Foundry, mOSAIC offers a complex Web control interface and for the Cloud Agency, REST interfaces (compliant with the emerging standard for Cloud resource provisioning, OCCI). On another hand Cloud Foundry has a strong components for performance analytics. This are small differences, the main one being at the conceptual level: mOSAIC was designed to target the independence of the Cloud enabled applications from the underlying Cloud infrastructure services, and therefore it provides (a) generic mechanism to ensure the functionality using different operating systems or virtualization mechanism and is currently working with several Public Cloud services (b) an API for Cloud enabled application. Cloud Foundry, designed mainly for Web applications, is based on VMWare virtualization mechanisms and its API is resumed to several deployment templates. In what concerns the usage of the COTS and application descriptors, mOSAIC is close to dotCloud (www.dotcloud.com), a recent commercial hosting service for Web applications that allows to combine Cloud services based on open-source deployable software.

In what concerns the usage of multi-agent technologies when using Cloud computing, mOSAIC is not bringing breakthrough new concepts, but is surely a successful implementation. The concepts which inspired the development of multi-agent system from mOSAIC were exposed in early papers of other groups like [1]. A recent comprehensive overview of the capabilities of agent technologies for Cloud intelligent services is presented in [11]. A parallel developed and even more complex solution at this moment in terms of services is for example Cloudle [10], based also on a Cloud ontology; however the Cloudle codes are not available for comparisons. Moreover, experiments with agents for multiple Cloud providers have not been found to be reported yet in literature.

In what concerns the automatic computing, mOSAIC is trying to follow the principles exposed in [4] related to the reactive techniques such as the policy-based, goal-based, or utility-based approaches, enabling the system to respond to problems only when they occur, like in the case of the scheduler and scaler.

IV. VENDOR AGENTS IN mOSAIC’S PaaS

In this section we are focusing on the latest developed modules, the Vendor Agents, as being relevant for the intelligent management 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.

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 (Figure 2.a and b). The CA support at least three kinds of applications: (a) ones that run on top of the mOSAIC Platform having CA as a resource provider
for the mOSAIC platform itself; (b) ones that run on the Cloud without mOSAIC Platform and CA provides provisioning and resource management as well as scaling up and down of the resources; (c) ones that include both mOSAIC components intended to run on top of the mOSAIC Platform and non mOSAIC components to be serviced by the CA. Consequently, 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 should also be 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 the execution 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 below:

```xml
<deployment_descriptor>
  <application>mOSAIC Application</application>
  <description>Requiring platform, CA, compute, storage</description>
  <resource_classes>
    <resource_class resource_type="compute" name="CA_VM">
      <provider>AMAZON</provider>
      <description>CA Virtual Machine</description>
      <url>deployed_image://CA.ami</url>
      <files></files>
      <run><program>startCA.sh</program></run>
  </resource_class>
</deployment_descriptor>
```
The deployment descriptor above mainly includes a set of descriptions for all the resource classes involved in the application to deploy. The example presents an application which needs: (a) the CA support, thus a CA Virtual Machine has to be available; (b) the mOSAIC Platform support, thus the Platform Control VM and Platform Execution VMs should be available; (c) a storage for a mOSAIC Platform Driver implementing a key-value storage; (d) the Web Server functionality presented as a distinct tier with all the resources at this tier being load balanced. A resource class is uniquely identified by a name and its type. After resource provisioning it is tight to a specific provider and a specific SLA. Not that 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. The first distinction between Cloud resources is made by the traditional classification, e.g. compute – a VM, storage – a bucket, load_balancer – a balancer associated with VMs, map_reduce – a resource implementing the map-reduce protocol. 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. A resource identifier includes information about the resource class.

Different vendor agents are expected to have some common behaviors which integrate 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 a uniform approach in the CA. The concept of Vendor Module was introduced therefore to encapsulate the specifics of Cloud providers (Vendor backends in Figure 1). Such a module, pluggable into the vendor agent, is based on an Abstract Vendor Module entity (Figure 2.c) intended to address all the common functionality of the vendor agents and their integration in the CA.

The operations outlined in the Abstract Vendor Module description falls into three categories: public, protected and abstract operations marked with different signs in Figure 2.d, respectively with +, # and italics. The last ones are implemented only by Vendor Modules, the others can be also in Abstract Vendor Modules.

The resource provisioning (Figure 2.a) 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 by that proposal are prepared in order for resource services to be created/activated. The creation/activation when an SLA is negotiated is not yet possible, as resources are intended to be created when needed 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).

The resource management refers here to all the operations on resources once they are provisioned (Figure 2.b). 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.

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.

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

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 to be transmitted or received from the CA API.

Further concerns of the Vendor Agents, like their migration from a local environment to a Cloud environment, as well as vendor specifics for the selected providers of technologies depicted in Figure 1, or more details of above the above described procedures can be found in the mOSAIC’s project deliverable on vendor agents and user guides. Note that currently in the open-source repository of mOSAIC, the codes of the Vendor Agents and Modules are available at https://bitbucket.org/mosaic/mosaic-agency-vendors/.

The proof-of-the-concept modules currently exposed and tested are designed for Amazon, CloudSigma, Flexiscale, NIIFI (Public Clouds), Eucalyptus and OpenNebula (Private Clouds), while agents for the other vendors mentioned in Figure 1 will be available until next Spring.

V. Conclusions and Future Work

As component of a Cloud-vendor agnostic and open-source platform-as-a-service, the resource management procedures have a high degree of complexity that are requiring artificial intelligence methods and a high degree of automatization. mOSAIC is one of the first middlewares that is successfully applies multi-agents and automatic computing technologies for an easy development of Cloud applications. One of its strengths that was revealed in this paper is related to the agent-based procedures for connecting to the platform new Public and Private Cloud resources.

The developments of mOSAIC will further continue in several directions. Validation of its practical approach through the intensive testing of the proof-of-the-concept applications (and implicit of the vendor agents usefulness and functionality) is the next immediate step. The automated resource management procedures will be further developed in the next two years in the frame of AMICAS project.

Acknowledgment

This research is partially supported by the FP7-ICT-2009-5-256910 grant (mOSAIC) and has mention some of the modules developed in the frame of the Romanian grant PN-II-ID-PCE-2011-3-0260 (AMICAS). The development of mOSAIC is being done by a large multinational team that is far from being reflected in the list of authors of this paper. We express our gratitude to the entire team and especially to the developers of the vendor agents for different Public and Private Clouds.

References