DOI 10.12694/scpe.v15i3.1019
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 15, Number 3, pp. 251–270. http://www.scpe.org

PORTABILITY IN CLOUDS: APPROACHES AND RESEARCH OPPORTUNITIES

DANA PETCU* AND ATHANASIOS V. VASILAKOS†

Abstract. The migration towards Cloud environments is still hindered by several barriers. One of them is the low portability of the applications that are consuming Cloud services. This paper intends to provide an image of the state of the art in this particular topic and to identify the potential paths to follow in order to solve the problem. The main concerns are the portability reasons, scenarios, taxonomies, measurements, requirements, and current technical solutions (through open standards, open application programming interfaces, semantics and model-driven engineering). A research agenda is following the current analysis of the state of the art.

Key words: Cloud computing, Portability, Multi-Clouds

AMS subject classifications. 68M14, 68W15, 68U01

1. Introduction. Portability is know as being the ability to use software components on multiple hardware or software environments. In particular, the portability in Cloud or portability between Clouds, or shortly Cloud portability in this paper, is expected to ensure that an application, service or data works in the same manner regardless of the consumed Cloud services and has a common method of interaction with these services.

The Cloud portability problem has been recognized already from the beginning of Cloud computing (e.g. in [1]). More recently, the IDC report from 2012 [2] points that portability is one of the most important obstacle hindering increasing Cloud adoption. However, the work on Cloud portability is gaining now momentum, as demonstrated by recent standardization initiatives for Cloud portability (e.g. TOSCA [3]). Cloud providers are nowadays also concerned about portability (e.g. IBM and HP are supporting to OpenStack initiative, while Amazon and Microsoft are investing in virtual machine portability [4]).

Several research papers, proof-of-concept implementations, or commercial products, have dealt until recently with the portability problem. However, Gonidis et al [5] observed that there is no extensive survey to describe in detail the problem space of Cloud application portability and how current solutions are mapped to that space. They underlined that such a study is essential in order to understand the root causes of the problem and the desirable characteristics of solutions to this problem. A first step has been done by them in [5], and we intend to continue in this paper their survey, as well as our earlier attempts in the same direction. More precisely, Petcu et al [6] proposed also a classification of the portability levels, while earlier Petcu [7] has underline the difference between interoperability and portability.

The novelty of this paper is two-fold. First, it gathers in one place the latest solutions. Second, it identifies the missing pieces needed to solve the problem. Therefore, the paper results can be useful to researchers in Cloud computing who need an identification of relevant studies, as well as to practitioners interested in understanding the available methods, techniques and tools and their maturity level in supporting portability in Cloud environments.

The structure of the paper is as follows. Section 2 is discussing the need for portability. Section 3 is defining the problem and the requirements. Section 4 investigates the current solutions. Section 5 is identifying the current barriers, challenges, open issues and the future solution path finding. Section 6 draw the conclusions.

2. The need for Cloud portability. Cloud computing is currently recognized as a computing model based on ubiquitous network access to a shared and virtualized pool of computing capabilities (like processing, network, storage, message queuing) that can be rapidly provisioned [8]. However this conceptual model is implemented by a large number of service providers in very different ways. The current software stacks of Cloud services are heterogeneous and the provided features that are often incompatible between different service providers. This diversity is an obstacle with respect to demands such as promoting portability and preventing vendor lock-in.

* West University of Timişoara, and Institute e-Austria Timişoara, Romania, (petcu@info.uvt.ro).
† University of Western Macedonia, Greece, (vasilako@ath.forthnet.gr).

The portability is requested by reasons varying from optimal selection regarding utilization, costs or profits, to technology changes, as well as legal issues. We discuss them shortly in what follows.

2.1. Economical reasons. Cloud portability is needed from an economical point of view for at least two reasons:

1. Customer: protection of the end user investments in developments;
2. Market: development of a Cloud eco-system and market.

Optimizing the operational cost is often the reason that is behind the interest of the Cloud service customer in Cloud portability. Rewriting the services and applications is usually required to comply with the change of the Cloud service. The porting process of customer application and data is also often triggered by changes of the offers of the providers.

The Cloud service providers can be interested to enhance their own Cloud resource and service offers to be more attractive in a highly dynamic market mainly by improving the quality of their resources and services. However agreements with other providers are possible (or even overtakes) in the case they provide services with particularities not provided elsewhere or complementary resources. The porting process of services is usually triggered by the operational changes.

If the Cloud portability is ensured, third parties are able to be developed, acting as intermediaries between multiple customer and multiple Cloud providers, enabling deployments according the customer requirements to the appropriate Cloud and adaptation to the current status and consumption conditions of the multiple Cloud services. The porting process of data, applications or services is usually triggered at customer request or automated by the third party.

2.2. Technical reasons. Cloud portability is needed from a technical point of view for at least two reasons:

1. Concept: to exploit the advantage of elasticity and pay-as-you concept;
2. Continuity: to ensure continuity in application and service functionality.

The portability between Private and Public Clouds is essential in realizing the vision of the Hybrid Cloud that handles the peaks in service and resource requests addressed to a Private Cloud using external resources of a Public Cloud. The porting process is usually triggered on demand basis.

The Cloud service providers are interested to use other provider services to ensure backup-ups to deal with disasters or scheduled inactivity. The porting process is usually triggered according to the providers agreements.

The replication of data and applications consuming services from different Clouds can be used to ensure their high availability. The porting process is usually triggered at the design phase.

2.3. Legal reasons. Cloud portability is needed from a legal point of view for at least two reasons:

1. Constraints: data, application and service protections according to the locations or national laws;
2. Free will: avoid the dependence on only one external provider.

Changes in a country legislation or changes in the customer location can trigger the need to port the software assets from a Cloud environment to another.

The Cloud non-portability of application, data, applications or services relaying upon a certain Cloud provider faulty service can produce a cascade effect on the activities of its service dependent customers. In this context the portability is necessary to reach a good recovery time objective.

Moreover, public institutions using auctions for acquiring services are reluctant to be engaged in contracts which are not offering proper services or guarantees in case of incidents.

3. Problem definition. As stated earlier, portability is the ability to move software assets among different runtime platforms without having to rewrite them partly or fully (enhancement of the definition given by Lenhard and Wirtz [9]). If the Cloud portability is achieved, data, application or service components can be moved and reused regardless of the operating system, storage or application programming interface (API).

In what follows we discuss some scenarios and attempts for classifications of various cases as well as the basic terminology.

3.1. Portability scenarios. The real world scenarios are highlighting two main categories of portability scenarios encountered in current Cloud service market (mentioned first by Ranabahu and Sheth [10]):

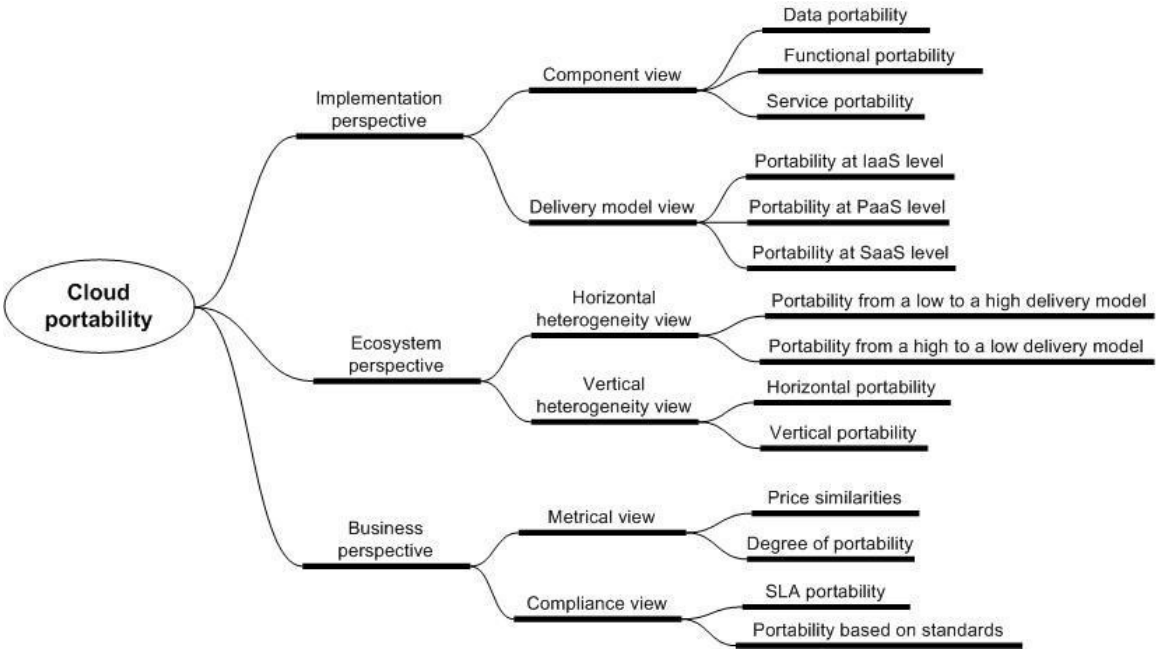

Fig. 3.1: Cloud Portability Taxonomy

1. Porting legacy applications, data or services: the legacy software assets are currently requiring a significant effort to be invested in porting them into in Cloud environment due to the dependence on particular technologies and data organization, on both sides, the initial environment as well as the Cloud environment.
2. Porting Cloud native applications, data or services: even when applications, data or services are written from scratch in a Cloud environment, they are usually locked and targeted for a specific Cloud, and the effort for porting in a different Cloud is usually a one time exercise.

Porting between Clouds of all or a part of existing services, applications or data is usually done one time. However a third party running services on multiple Clouds and offering unique entry points to various service customers is interested to ensure that the porting process is reversible, fast and semi-automated.

The most challenging scenario for portability is that in which the Cloud applications are distributed across several administrative domains of different providers simultaneously, and, moreover, at least data (if not even application and service components) are ported from one Cloud environment to another.

A less discussed scenario is the reverse portability from the Cloud environment towards the own premises resources (even partially, lets say only the data part). This is critical when the service provider changes the API or imposes API restrictions or depreciate a service, so that all depenedent services fall (domino effect).

3.2. Portability taxonomy. Figure 3.1 introduces a taxonomy of Cloud portability based on the studies that are detailed bellow.

Kolb and Wirtz [11] categorized Cloud portability from three different perspectives: business, ecosystem and implementation (while the study was done for platform-as-as-service level, it is generally valid also for the other service delivery models):

1. Implementation perspective: is related to portability constraints that are implementation-specific requirements or restrictions, e.g. deployment descriptors, restricted usage of runtime APIs or specific management API calls.
2. Ecosystem perspective: describes concrete portability constraints including application specific dependencies like runtimes and specific services.

3. Business perspective: is the most abstract perspective and includes business relevant, non-functional and abstract constraints on portability, like pricing, compliance or SLAs.

From the implementation perspective, depending on the software assets that are ported, the Cloud portability can be classified in three categories (first mentioned by Oberle and Fisher [12]):

1. *Data portability:* enables the re-use of data components between Clouds. It is based on import and export functionality of Cloud data services for data structures.
2. *Functional portability* or *application portability* refers to the Cloud service agnostic definition of application functionality.
3. *Service portability* or *platform portability* is the ability to add, reconfigure and remove Cloud resources on the fly, independent on the Cloud provider.

Data portability is achieved when application data can be retrieved from one provider and imported into an application hosted by another provider. To reach this type of portability, platform-independent data representation is needed to be done in a short-term, as well as specific target representations and code for the application's data access layer. On long term, data portability depends on standardization of import and export functionality of data and its adoption by the providers that are the market key-players.

Functional or application portability requires the definition of the application's functionality details in a vendor-agnostic manner. A particular issue is the application portability between development and operational environments. Cloud computing is bringing development and operations closer together, the two being integrated as DevOps. This requires that the same environment is used for development and operation, and the application is portable between development and operation environments. If different environments are used the application portability can become an important issue.

Service or platform portability is based on Cloud-agnostic control APIs allowing Cloud resources to be added, configured, or removed in real time by humans or programmatic, based on factors like current outages. In the case of service or platform portability there are two scenarios:

1. Service component portability: service or platform components are ported between Clouds, without involving the applications and data on top of them;
2. Machine image portability: bundles containing applications and data and their services or platforms are ported between Clouds.

The portability expectations are different for each service delivery model:

IaaS: The main scenario is related to the migration of applications, data and services to a new IaaS environment.

PaaS: The aim is to minimize the amount of application rewriting while preserving or enhancing controls, and a successful data migration.

SaaS: The main scenario supposes that the Cloud customer is substituting software applications with new ones. The aim is to preserve or enhance the application functionality.

For example, in the case of PaaS, specific points of conflict emerging when attempting to deploy an application to multiple platforms are the followings (according Gonidis et al [5]): programming languages and frameworks, platform-specific services, data stores and platform specific configuration files.

The Cloud service eco-system and market has currently a high degree of heterogeneity. This heterogeneity is classified by Ranabahu and Sheth [10] and as being vertical or horizontal. In the first case the heterogeneity appears within the same delivery model. For example, in IaaS case, raw infrastructure resources are exposed as services and various resources are exposed differently between IaaS providers. In the horizontal case, the heterogeneity appear within different service delivery models. For example, a typical PaaS requires the use of specialized libraries and run-times with limited functionalities. A IaaS native application assuming the control over operating system requires extensive changes when moving into a PaaS environment. Note that the split according vertical and horizontal views is not yet commonly accepted, as the same terms are reused in other contexts in solving portability issues. In particular, Jonnalagedda et al [13] distinguish two major cases:

1. *Vertical portability* as the capability of an application intended for a Private Cloud to be portable to a Public Cloud. The restriction is that the application runs on the same technology stack on both platforms.
2. *Horizontal portability* as the ability to port an application from one technology stack to another, staying

at the same level of abstraction but changing the technology provider.

3.3. Measuring the portability. The porting process is build from various tasks:

1. rewrite and recompile codes in case of the applications and services;
2. re-structure data and applications;
3. re-configure services;
4. transfer data, applications, services or machine images between Clouds.

If these tasks are straightforward or automated, the degree of portability is high. If these tasks are requiring timely human effort, the degree of portability is low.

In particular, according to the service delivery model the portability is evaluated based on:

IaaS portability of the virtual machines (VMs) and of the configurations across different run-time environments.

PaaS openness degree of the source programming languages for application development, openness degree of the data formats, coupling degree of the coupled services, platform-dependence degree of the abstraction layers for messaging and queuing services.

SaaS openness of the source code, openness of standard data formats, integration technologies and application server or operating system.

Lenhard and Wirtz [9] have define software quality metrics in order to quantify the degree of portability of an executable service-oriented process. They have map existing metrics for program portability to service and process-oriented programs and they have combined the metrics with empirical data on language support in various runtimes. Only program elements that are supported by a majority, or all, runtimes are considered by Lenhard and Wirtz to be portable. Note that their measure of the portability of executable process definitions take only the runtimes for the analyzed process into account. If all runtimes available support all of the language elements available in the same manner with respect to semantics, then any compilable application of service is supposed be portable to any runtime. In such context, Lenhard and Wirtz stated that the first step towards measuring the portability of a process is to calculate the degree of portability for each language element and its configuration: this degree can be identified by the number of runtimes that support a language element. This measurement procedure can be currently applied only partially to portability in PaaS and IaaS environments.

3.4. Cloud portability requirements. Figure 3.2 enumerates several requirements in order to ensure the portability between Clouds. The technical requirements are split into three categories according the stage of the application (functionality), data or service in their life-cycle.

3.5. Cloud portability relationship with Cloud interoperability. The interoperability and portability are often used in conjunction when Cloud adoption barriers are mentioned. They are often seen as one and the same concept due to the fact that many of their technical solutions are the same. However the two concepts are different.

Cloud interoperability is the ability of diverse Cloud services, data and applications to work together. It is mainly a concern for Cloud service providers to ensure the proper functionality of their services with the external world (even with services of other Cloud providers). Usually it assumes a previous knowledge or agreement of the systems with which the current one will inter-operate. Therefore the issue appears mainly in the case of Federations of Clouds, that are based on agreements between Cloud providers. The customer of the Cloud service is usually not aware of the Cloud interoperability facilities of the Cloud provider which whom is in agreement. Moreover, the interoperability degree between Clouds is usually measured at runtime. Data synchronization is for example a concern that is encountered in Cloud interoperability and is not a concern for Cloud portability. A recent comprehensive study on Cloud interoperability was provided by Toosi et al [14].

On opposite site, Cloud portability is a concern that appears mainly at design and deployment stages of an application, data or service. The aim is to minimize the human efforts in re-design and re-deployment of application, data and services when moving from one Cloud to another.

4. Current solutions for Cloud portability. Cloud portability is currently achieved through open standards, protocols, widely used APIs or through abstraction layers which decouple application development from specific target Clouds.

Fig. 3.2: Portability Requirements

4.1. Types of solutions. Gonidis et al [5] and Silva et al [15] suggested that there are three types of solutions for the Cloud portability problem:

1. adoption of existing or emerging standards and protocols (like TOSCA, CDMI, OCCI, OVF);
2. usage of intermediary layers (like jClouds or mOSAIC);
3. adoption of semantics and model-based solutions.

These solutions are complementary rather than contradictory. Moreover, regardless of the solution type, a common approach is creating an abstraction layer that seeks to hide the differences between various Cloud services. The abstraction layer is agnostic to the Cloud services by hiding or wrapping the proprietary technologies of particular Cloud providers. Moreover the abstraction layer prevents developers from being bound to specific programming languages or data stores. Adapters are used to support the interaction between the abstraction layer and the target service.

4.2. A time-line perspective. The technical solutions for the portability have evolve in time, and three main stages are visible:

Stage 1: The virtual machine portability: the export and import of virtual machines, with the introduction of Open Virtualization Format (OVF) standard.

Stage 2: The networked virtual machine portability: virtual machines together with their network settings are moved between Clouds.

Stage 3: The data, application, and services portability: the portability issues are focusing on the storage and as well as APIs and grouping on-demand.

4.3. Open standards. The classical way to bust the portability is the adoption of standards and open source. Surely they are important mostly for the Cloud providers and service developers, and not in a similar degree, for the consumers.

Tables 4.1 and 4.2 are pointing towards the most important standards for Cloud portability, and, respectively, Cloud standard initiatives. Loutas et al [16] have done a complex survey of the efforts by the industry, standardization groups, and research community, at the level of 2011. More recently Lewis [17] provided a non-exhaustive list of standards-related efforts for Cloud computing, discussing also the role of standards in Cloud computing. He is also arguing that the standards do not make sense beyond IaaS, taking into account the fact the service model often involves value-added features (taking the form of application software, in the case of SaaS, or the form of platforms or libraries, in the case of PaaS).

The most prominent standard in Table 4.1, initially intended for IaaS layer, is Open Cloud Computing Interface (OCCI). OCCI is a specification for remote management of Cloud service infrastructure. It allows the development of tools for common tasks including deployment, monitoring, and autonomic scaling. Its API supports three concepts: compute, network and storage. The current OCCI standard claims to be applicable for all layers, but it is still missing a concrete proposal besides the IaaS management API (according Kolb and Wirtz [11]).

Other open standards are OVF and CDMI. OVF describes virtual appliances to be deployed across various virtualization platforms. CDMI specifies the functional manner on how applications operate with data residing on Cloud environments.

The Topology and Orchestration Specification for Cloud Applications (TOSCA), the result of standardization effort from OASIS, is the most recent entry in Table 4.1. Binz et al [3] presented how TOSCA enables the portable and standardized management of Cloud services. The composite applications and their management in a modular way is done based on service templates that contain service's topology and its operational aspects. TOSCA specifies the meta-model for types of services as well as the templates (more specific, the language for defining them). TOSCA enables application developers to model re-occurring tasks explicitly into plans using a workflow language supported by a management environment. The operations are described in WSDL, REST, or scripts. The last ones are implementing particular management operations on a specific node. These operations can be external services or their implementation can be included in the service template (as an implementation artifact). The portability is dependent on the workflow language and engines that are used.

An on-going initiative is the Guide for Cloud Portability and Interoperability Profiles (CPIP): the IEEE project P2301 (from Table 4.2), is developing a portability standard. It proposes to group different interfaces, operation conventions or tile formats of the Cloud elements into logical profiles.

4.4. Open-source libraries and deployable services. The use of abstraction layers and adapters has been widely adopted by libraries, tools and platforms aiming to enable Cloud portability. An abstraction layer hides the differences between providers and exposes a uniform semantics and syntax.

According Hill and Humphrey [18], the APIs that are allowing Cloud portability are classified in three categories:

1. multiple independent implementations, like Eucalyptus versus EC2, AppEngine versus AppScale;
2. runnable on multiple Clouds, but not through independent implementations, like the several implementations of MapReduce;
3. separators of the application into application-logic layer and Cloud layer (Table 4.3).

Representative for the first category, AppScale is an open source software system implementing a PaaS. It is API-compatible with Google App Engine (GAE): it allows the execution of GAE applications on-premise or over public Cloud infrastructures, without any code modification [19].

The latest category (and most general) requires a large time investment by a developer to create the layers, and, later on, to maintain them as the APIs change. Several open APIs offer already a thin abstraction level

Table 4.1: Standards, reference architectures, open group proposals

Type	Acronym	Link	Usefulness for portability
Standard	OCCI	occi-wg.org	It is designed to be a management API for IaaSs and includes tools for deployment, monitoring and autonomic scaling.
	OVF	www.dmtf.org/standards/ovf	Describes virtual appliances for the deployment across heterogeneous platforms, like the ones using different hypervisors, allowing its users to deploy their virtual appliances in various Clouds
	CDMI	cdmi.sniacloud.com/	Used for data management, it specifies how applications create, retrieve, update, delete data in the Cloud
	CIMI	dmtf.org/sites/default/files/standards/documents/DSP0263_1.0.1.pdf	It addresses the management of the lifecycle of infrastructure services. Basic resources of IaaS are modeled in order to provide to the consumer a management access to an implementation of IaaS. It focus on the portability between Cloud implementations that support the specification. A REST-style protocol is used.
	TOSCA	http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html	Specification language to describe service components and their relationships. Its service template uses a combination of topology and orchestration and describes what is needed to be preserved across deployments in various environments in order to enable inter-operable deployment and management of services when the applications are ported over alternative Cloud environments.
Architecture	IETF	tools.ietf.org/html/draft-khasnabish-cloud-reference-framework-01	Proposed a draft of Intra-Cloud and Inter-Cloud reference framework, documenting basic functions or layers to support the general requirements of Cloud services. The framework can be used to standardize the interfaces between the functions/layers.
	DMTF	dmtf.org/sites/default/files/standards/documents/DSP-IS0102_1.0.0.pdf	A reference architecture for Cloud services that describes key components, such as actors, interfaces, data artifacts, profiles and the interrelationships among these components.
Open groups	The Open group	www.opengroup.org/getinvolved/workgroups/cloudcomputing	The Cloud Work Group within Open group collaborate on standard frameworks and models for eliminating vendor lock-in.
	CSCC	www.cloud-council.org	Has released a guide for SLAs that highlights the critical elements of a SLA for Cloud and provides guidance on what to expect and what to be aware of when negotiating an SLA.
	OCC	opencloudconsortium.org	Has develop benchmarks for Cloud computing. MalStone Benchmark is targeting large data Clouds.
	GICTF	www.gictf.jp	Promotes interfaces through which Cloud services are interworking with each other. It has published: an Intercloud interface specification – the Cloud resource data model and Intercloud protocol – as well as technical requirements for the Intercloud networking, with specific use cases.

Table 4.2: Standard initiatives

Organization	Link	Usefulness for portability
ETSI	www.etsi.org/ technologies- clusters/ technologies/grid- and-cloud-computing	The technical committee for Cloud is looking to interoperable solutions involving IT and Telecom industries, with emphasis on IaaS. It focuses on applications and services which are inter-operable due to the standards as well as on the validation tools that are support such standards.
IEEE	standards.ieee.org/ develop/project/2301. html	The IEEE group is working on standardizing Cloud management (including portability), by using specific interfaces and file formats.
ITU-T	www.itu.int/en/ITU-T/ focusgroups/cloud/ Pages/	A study group which works on standards for renewed networks in conjunction with Clouds.
CSA	cloudsecurityalliance. org/wp-content/ uploads/2011/09/ Domain-6.docx	The working group 'Portability and Interoperability' has elaborated a study on the risks and benefits involved with Cloud portability
NIST	www.nist.gov/itl/ cloud/	Program to develop a set of Cloud computing standards. NIST' special publications are referring to Cloud architectures, Cloud security, Cloud deployment in the context of various strategies of USA federal government
OASIS	www.oasis-open.org/,	Three important technical committees are activating: (1) SAF, referring to capacity, quality of service, and cost; (2) CAMP, Cloud application management for platforms, defining interfaces for self-service provisioning, monitoring, and control of Cloud platforms; (3) TOSCA
OW2	www.ow2.org/view/ Cloud/	Open Source Cloudware initiative (OSCi) intended to develop an OSCi component set with state-of-the-art Cloud stacks or component set; currently it supports CompatibleOne
SNIA	www.snia.org/tech_ activities/work/twgs	Cloud Storage Technical Work Group developing an architecture related to system implementations of Cloud storage technology.
TM Forum	www.tmforum.org/ Digital Services/13907/ home.html	Cloud services initiative stimulating an open marketplace for Cloud services by promoting to large eco-system of enterprise customers, Cloud and technology providers the adoption of standards

for the unique representation of the resources (namely, they are wrappers solutions); for example, jclouds when using Java, libcloud when using Python, SimpleCloud when using PHP, and δ-Cloud when using HTTP.

The jclouds introduces the following abstractions: description of node with metadata such as CPU, RAM, security policy; an abstract representation of a node with parameters such as minCPU, OS type; group of nodes to be managed together; set of command to be executed on nodes; information about the provider.

Most of the above enumerated libraries are language-dependent in their attempt to provide a common access to multiple Clouds. Typically they provide provide a code-based model of the infrastructure and do not offer any mechanism for automatic provisioning and deployment of applications on the Clouds.

Cloud resource management tools and services that are exposed in Table 4.4 are also using adaptors, but they include deployment mechanisms.

Cloudify model for deploying applications includes: a recipe for information like required infrastructure and how it should be used, the cluster of service instances that make up an application tier, the configuration (including provisioning and scaling rules) of an application and the services it is made of, the set of services

Table 4.3: Open-source libraries that support a certain degree of portability

Acronym	Link	Short description
δ-Cloud	deltacloud.apache.org	It is REST-based API written in Ruby necessary to connect to various Cloud providers (Amazon EC2, Eucalytus, SmartCloud, GoGrid, OpenNebula, RackSpace, OpenStack and others)
Dasein Cloud API	dasein-cloud.sourceforge.net/	Java-based library for the access to compute services such as a virtual machine support, volume support, and other features associated with cloud compute as a service
fog	fog.io	Ruby-based library providing a high level interface to various Clouds 'collections' (images, servers). 'Requests' allow to dive in service particular features. 'Mocking' allows simulations by an in-memory resource representation.
jclouds	www.jclouds.org	It is an open source Java library that introduces abstractions aiming the portability of applications. It support more than thirty Cloud providers and software stacks including AWS, GoGrid, vCloud, OpenStack, Azure.
libcloud	libcloud.apache.org	Apache Libcloud is a standard Python library abstracting the differences between multiple Cloud provider APIs
Simple Cloud	www.simplecloud.org	It is a PHP library providing common interfaces for file and document storage services, queue services and infrastructure services

working together, probes used to monitor the status of the system.

OpenNebula provides interfaces to interact with the physical and virtual resources. The interfaces are allowing the use of OCCI, EC2 EBS and Query, a Sunstone GUI, and a particular portal. Low-level APIs are available for Ruby, XML-RPC and Java. A catalog of virtual appliances ready to run in OpenNebula-based environments is also available.

OpenStack offers services for compute, storage and networking. These services include identity, image management, as well as a web interface, and integrate the OpenStack components with each other and with external systems. Moreover, its DevStack enable the run of a OpenStack instance on a laptop or inside one VM.

The mOSAIC open-source PaaS is offering two level of abstractions: from the Cloud provider technologies as the above mentioned technologies, but as well as from the classical synchronous programming style. The applications can follow an event-driven architecture [20]. Programming libraries are available for Java, Python and Erlang. Opposite to OpenNebula web interface and OpenStack dashboard, its web interface focus on the life-cycle of the application components, not on the management of Cloud resources. Adaptors are build for the major Public and Private Cloud technologies. The Personal Testbed Cluster (PTC) allows the development of the applications on a desktop and test on few virtual machines (as much as the desktop allows) and later on the seamless transfer of the applications into Private or Public Clouds.

CompatibleOne is based on a platform and a model for the description, aggregation and integration of distributed resources provisioned by heterogeneous Cloud service providers. Its platform uses the OCCI. CompatibleOne is taking part to OW2 Open Source Cloudware initiative (OSCi).

Commercial solutions like RightScale or Kaavo are able to deploy applications in various Clouds, but not yet migrate the running ones.

In what concerns storage and data portability, three different approaches were proposed: common namespace, service registry and uniform APIs. The most significant initiatives are enumerated bellow in chronological order.

CSAL, introduced by Hill and Humphrey [18], is a storage abstraction layer enabling applications to use scalable storage services provided by Public Clouds and also to be portable across platforms. It provides

Table 4.4: Open-source tool or service that support a certain degree of portability

Acronym	Link	Short description
Aoleus	www.aeolusproject.org	Cloud management software sponsored written in Ruby, open source, and provided by Red Hat. It allows users to select private, public or hybrid Cloud services, using δ-Cloud library
CompatibleOne	compatibleone.org	Its is an open Cloud broker offering an interface allowing for the description in terms of resources of user needs, as well as the provisioning on the most appropriate Cloud environment. It provides also a language for the description and management of Cloud services.
Cloud Foundry	cloudfoundry.org	Open source PaaS on top of VMware-based IaaS.
Cloudify	getcloudify.org	It orchestrates the creation of the infrastructure services requested for an application, from compute resources to networks and block storage devices. It works with OpenStack, monitor the application progress and scale the resources when needed.
ConPaaS	contrail-project.eu/conpaas	Provides a federation layer support for bringing Cloud providers together. It allows Cloud bursting. The Contrail Virtual Execution Platform interfaces with the IaaS layer of Cloud providers and upgrades the supported Cloud providers by adding SLAs features
Docker	github.com/docker/docker	Is an deployment engine relying on the Linux containerization sandboxing method
mOSAIC platform	bitbucket.org/mosaic	An API and a deployable PaaS that allows deployment, configuration management, and control of the life-cycle of applications or services consuming IaaS. Supports more than ten providers
Nimbus	www.nimbusproject.org	Introduces a virtual site layer for dynamically provisioning of distributed resources lying on multiple data centers. It is based on a closed federation model in which resources are shared based on cooperation agreements
OpenNebula	github.com/OpenNebula/one	Open-source tool for data center virtualization. It offers a solution to build and manage virtualized data centers.
OpenStack	www.openstack.org	Is a open-source system to control compute, storage, and networking resources of a data-center through a dashboard.
OPTIMIS	www.optimis-project.eu/Toolkit_v2	Is a platform for Cloud service provisioning. It deals with the lifecycle of the service. Moreover, it addresses the risk and trust management

abstractions for blobs, tables, and queues. A key differentiator for CSAL is that it provides a single namespace for each storage type (by this is able to decouple the application code from explicitly making calls to specific storage services).

Thabet and Boufaida [21] proposed an agent-based architecture to publish and describe the different data services offered by providers. A service registry contains a description of services. An intermediate layer serves to enhance and help manage service invocations. The agents allow searching, collecting and filtering data in order to transfer appropriate and requested data. The adapter agents for converting exchanged data formats from one environment to another.

Rafique et al [22] proposed an abstraction layer that provides a uniform API for three particular PaaS services, including scalable data storage, blob storage, and asynchronous task execution.

Portability at PaaS layer has attract a lot of attention in the last time period. Some significant proposals in chronological order are the followings.

Cuhna et al [23] proposed a definition of a architecture allowing developers to create and expose services through a particular service delivery platform. It enables the portability of service between PaaS through a common API.

Zeginis et al [24] implemented a tool in the frame of Cloud4SOA project that allows the portability of Java applications between multiple PaaS.

Kolb and Wirtz [11] defined a model of current PaaS offerings and identify different portability perspectives. Starting from the model, they derive a standardized profile with a common set of capabilities that can be found among PaaS providers and matched with one another to check application portability based on ecosystem capabilities (project homepage: https://github.com/stefan-kolb/paas-profiles).

4.5. Semantics. The semantics technologies are applied to the interface, data or component levels using often a unified Cloud resource model. Moreover, the semantics are used to annotate applications and services (more precisely, their properties), in order to deploy and execute them efficiently on the selected infrastructure.

Several theoretical studies have been issued expressing various aspects of using semantics for Cloud portability. We mention the most significant ones, in chronological order.

The first proposal for a Cloud ontology was given by Youseff et al [29]. It has prove the benefit of using an ontology as an general method for the conceptual description of information implemented by Cloud resources.

Meximilien et al [30] envisaged a Cloud-agnostic broker for any Cloud client, to not being tied to any Cloud provider. For this purpose a meta-model was created and claimed to be applicable to any Cloud platform. The meta-model serves is an abstract representation of the APIs and Cloud functionalities.

Bernstein and Vij [31] proposed a mediation mechanism capturing the features of a Cloud provider infrastructure, and logically groups and exposes them as standardized units. The mediation mechanism uses a resource catalog approach, based on an ontology of Cloud computing resources and defined using RDF. A 'root' provider act as broker and host this resource catalog.

Ranahabu and Sheth [10] presented a semantics-centric programming paradigm for the development of portable Cloud applications (named MobiCloud). The four types of semantics for a Cloud application are inspired by the four types of semantics for a general service: logic/processes, data, system and non-functional semantics. Annotations are linking different descriptors and models addressing various concerns. To port data, a transformation is activated using the generic data format (this is a example of using the lifting lowering mechanism: the data is first transformed to a common format, i.e. lifted, and then transformed back to the target format, i.e. lowered).

Loutas et al [32] introduced a PaaS semantic interoperability framework that resolve semantic conflicts raised during the migration or deployment of an application. It defines the following dimensions: types of semantics, fundamental PaaS entities, and levels of semantic conflicts.

Cretella and Di Martino [33] proposed an approach to perform automatically the enrichment of API with semantic content and the alignment of software. The approach aimed at analyzing the functionalities and features offered by Cloud service provider, through an automatic analysis of their APIs, and building a semantic representation that is linked to the application and functional domain to which the API refers.

Dukaric and Juric [34] proposed a unified taxonomy of Cloud frameworks. The taxonomy of IaaS consists of several layers: resource abstraction, core service, support (communication layer between core service and resource abstraction), security, management, control, and value-added services. Ten IaaS were mapped to the proposed taxonomy and then compared using the taxonomy.

Di Martino [35] is underlining that metadata added to specific service APIs through annotations pointing to generic operational models could enable the portability and interoperability among the heterogeneous Cloud environments.

Another portability solution that is discussed in [35] is that of the Cloud patterns usage. The patterns are describing common behaviour in Cloud environments and they support the idea of an application redesign, reengineering or redeployment. Multiple pattern catalogs are currently available, like the ones from `cloudpatterns.org`.

Table 4.5 enumerates several concept implementations.

Table 4.5: Implementations of semantic concept for ensuring portability

Acronym	Reference	Short description
Cloud4SOA	[25]	Multi-cloud PaaS management that uses semantic-based adapters to bridge multiple PaaS. It is based on an approach to semantically interconnect heterogeneous PaaS offerings of different Cloud providers sharing the same technology. Cloud4SOA solution comprises of a set of interlinked collaborating software components and models to provide developers and platform providers with a number of core capabilities: matchmaking, management, monitoring and migration of applications.
CPIM	[26]	A Java library exposing a vendor independent API that provides an abstract intermediation layer for the most important middleware services typically offered by PaaS (e.g. NoSQL services, message queues and memcache). The library supports the portability of applications across Java platforms for AppEngine and Azure, reconciles the difference in the semantics of the queue services offered the two PaaS, and implements two different services called task queue, adopting the AppEngine semantics and message queue adopting the Azure semantics.
mOSAIC ontology& semantic engine	[27]; [28]	A Cloud ontology was developed in OWL. Its main part contains the definition of general concepts as well as the relationships with the Cloud services (like resource, VM). The APIs and Services sub-ontologies describe the model, the profile and the grounding for the APIs and services. The Requirements sub-ontology is modelling service level agreements. The Semantic Engine is used for Cloud applications development: it offers a semantic discovery of API components and functionalities, at the design time, as well as of resources, at deployment time. It supports the user in selecting the list of needed resources to be acquired from the Cloud providers.
UCI	`code.google.com/p/unifiedcloud/`	Unified Cloud Interface is a programmatic point of contact referring to the infrastructure stack through a unified interface. It uses the resource description framework (RDF) to describe a Cloud data model.

4.6. Model-driven engineering. Model driven engineering (MDE) is based on two core ideas [5]: abstraction and automation. Abstraction enables decoupling application development from targeting specific platforms. Automation refers, among others, to the ability to change the level of abstraction automatically, using model transformations (e.g. domain specific languages, shortly DSLs). The DSLs allows to the domain experts to create programs having little knowledge in programming.

The model-driven approach, often summarized as model once, generate anywhere, is particularly relevant when it comes to provisioning and deployment of applications and services across multiple Clouds, as well as migrating the source code and data from one service provider to another. Through high level models and automatic transformations, the application developers can focus to a conceptual and platform-independent level.

Unfortunately, there is no consensus about the set of languages, models, transformations or software processes to be used for a successful development of Cloud applications using MDE (according Da Silva and Lucredio [36]). Current efforts are focused mostly on the identification of abstractions allowing the specification and creation of systems independently of the underlying platform (as reflected also in the previous section), and the automatic code generation for specific Cloud services.

One main concern for Cloud portability is the variety of data representations in Clouds. MDE is trying to respond to this challenges using abstract data models. Logical data models are presented for example by Samba [37], while Shirazi et al [38] provided a solution through design patterns to enable the portability between

graph databases and column family databases.

Di Martino et al [39] investigated a methodology to automatize the recognition of similar elements in Cloud related design patterns, leading to an automatic mapping among representations of application that follow these patterns.

Ranabahu et al [40] presented an abstraction-driven approach to address the application portability issues and focus on the application development process. The approach is using a specification in form DSL script, to generate platform specific but functionally equivalent executable applications. The approach is based on a formal definition of a Cloud application, based on the assumptions that each semantic aspect can be expressed using a DSL, and the abstract syntax model of each of these DSLs can be represented using a graph. The transformation from the domain model to a Cloud-supported implementation model depends on the details of the domain meta-model. Proof-of-concept tests were done with the MobiCloud platform proposed by the same authors.

A proof-of-concept platform, named CodeCloud, introduced by Caballer et al [41], supports the execution of scientific applications covering different programming models on Cloud infrastructures. The platform automatically deploy the applications on required virtual infrastructure and perform the execution of the jobs according to the specified programming model.

Model-driven development combined with model-driven risk analysis for Cloud computing application cycle is currently studied in MODAClouds project (overview papers: [42], [43] and [44]). Its aim is to enable application developers to specify the models of Cloud services in which they are interested and to enrich these models with quality parameters. Moreover, to close the current gap between the design and run-time stages of an application, performing quality predictions, as well as run-time monitoring and optimizations, provide valuable information to the design time environment.

Three levels of abstraction are considered by MODAClouds: CCIM, the Cloud enabled Computation Independent Model to describe an application and its data; CPIM, the Cloud-Provider Independent Model to describe Cloud concerns related to the application in a Cloud agnostic way; and CSPM, the Provider Specific Model to describe the Cloud concerns needed to deploy and provision the application on a specific Cloud. CCIM describes Cloud applications at the service level. Hence, for a given application it contains the description of the services that compose it, the public interfaces of each service, the business processes that describe their orchestration and the domain model of the data exchanged by them through their public interfaces. Modeling concepts and technologies for the provisioning, deployment and adaptation of applications in various Clouds (CPIM/CPSM) can be done using the uniform interfaces (mentioned in the previous section) provided by various libraries for application development, deployment and control at run-time.

To illustrate the differences between the three abstraction layers, the data models are considered here. The CCIM Data Model describes the main data structures associated to the software to be. It is expressed in terms of a typical Entity Relationship diagram and enriched by a meta-model that specify functional and non-functional data properties. The CPIM Data Model describes the data model in terms of models being typically offered by Cloud providers, without referring to a particular one. Since the majority of available data storage software provides services that include distributed file system, NoSQL solutions, and blobs, these three models are considered at the CPIM level that are suitable to represent these different cases. The models are conceived for graph data, hierarchical data and flat data. The CSPM Data Model describes data structures implemented in the Cloud providers. The following families are considered: relational data, hierarchical data, flat data, key-value data, column-based data.

The models@runtime paradigm proposes to leverage models during the execution of adaptive software systems to monitor and control the way they adapt [45] . This approach enables the continuous evolution of the system with no strict boundaries between design-time and runtime activities (supporting therefore the DevOps concept). The approach is adopted for Clouds by CloudML, promoted by MODAClouds, ARTIST [46] and PaaSage [47], a follow up of PIM4Cloud from REMICS [48]. CloudML is domain-specific modeling language enhanced with a run-time environment. It facilitates the specification at design-time of provisioning, deployment, and adaptation concerns, as well as their enactment at run-time [49].

5. Barriers, Challenges and Open Issues. Reflecting on the above described solutions, we can discover several advantages on using one or another solution, as well as their limitations that are generating a need for

Table 5.1: Advantages and limitations of various approaches for Cloud portability

Approach	Advantages	Limitations
Standard	– Result of a collective agreement – Extract the key actions and characteristics – Should be implementable	– Not widely adopted – From the provider point of view, hinders diversity – The number of emerging and overlapping standard makes the problem to grow
Open library	– Offers an abstraction layer that is simple to use – Available for major languages – Similarity with major Cloud provider APIs – Decouple the application code from the underlying Cloud service – Adaptors available for major Cloud services	– Usually refers to the common denominator of the Cloud services – Language dependent – Adaptors needs to be build for emerging new services – The connected service programming style usually maintained – Require Cloud computing knowledge as deployment is usually not supported – Introduces an overhead compared with the direct connection to the Cloud service
Open service	– Offer unique entry point for application deployment and Cloud resource management – Application deployment can be done by non-Cloud specialists – Part of them are offering also APIs for programming applications – Usually it offers support for multiple programming languages – Monitoring tools are generate alarms needed to trigger a re-deployment	– The diversity of deployment services raise also another dimension for the portability – Manual intervention at deployment phase is usually still needed – Life migration is still not possible – Re-deployments are not automated – Rely upon adaptors that need to be build for new services or updated when a service version appears
Semantics	– Offers an abstraction layer that can support various customers – Offers viable mechanisms for common understanding of service terminology and actions – Allow the annotation of services with quality marks by externals from the provider team	– Not widely adopted – The variety of taxonomies and ontologies makes the portability problem to grow – The overhead of semantic processing is not negligible
Model-driven	– Enhance the abstraction layers with an automation process – Allow a feedback from operational modules to the design modules	– Available tools are not yet generating code for various Clouds – The models that are used potentially omit special features of the services

improvements in the near future.

Table 5.1 is pointing therefore to some advantages and limitations of the main existing solutions for Cloud portability that were presented in the previous section.

In what follows we discuss in more details what is missing and what is possible to be done in a near future to achieve Cloud portability.

5.1. Standards. Standardization is not an appealing solution for some Cloud service providers. The providers are interested to differentiate themselves by unique offers.

OVF, OCCI and CDMI have partially failed to be adopted on large scale by the providers as are currently implemented only in few Cloud management tools. Unless there is a well-accepted and widely used standard, it is a questionable solution.

As the emerging standards are not widely adopted, a common understanding on a certain action or feature of a Cloud service has not yet been reached. Consequently, the third party supporting for example an uniform interface needs to understand each particular service interface in order to plug it in their service or software. This fact is not compliant with the requirement of a seamless joining procedure for a new service. If the conditions that the interface requires to comply with is constantly increasing market, the compliance with them is considered a moving target, that is difficult to reach.

New standards are emerging nowadays, like TOSCA or CAMP, and several collaborative groups are working to elaborate other proposals. However there are several gaps in the collections of available standards, like proposals for Cloud metrics and real-time monitoring, interfaces for security(-as-a-)services, accountability associated with transparency and responsibility.

A unified policy of the contractual terms was not yet established at national or international levels, while several proposals are on the way. One of the most disputed topic is the privacy and data protection compliance, for which no general accepted proposal is currently available.

There is a clear need to define standardized, machine-readable and self-descriptive representations of the basic characteristics of the resources (like API, size, or platform), the advanced characteristics of the resources (like quality attribute values or pricing), the negotiation processes and protocols, as well as billing processes and protocols [17].

System developers need to leverage standards to support the architecture of a system. However, such standards should not drive such an architecture [17]. The software architecture for Cloud-based systems in which standards-reliant components should be implemented as components that are separated from the rest of the system (in order to minimize the impact on the evolution of the standards).

Moreover, the standardization should focus now on the basic use cases of user authentication and data migration. Workload migration and workload management can be the subject of dynamic use cases in which location, negotiation, and provisioning of Cloud resources occur at run-time.

Cloud service portability has not been examined until now with regard to management and operational tasks. These tasks are an increasing and significant cost factor. Moreover, standard for defining composite applications and their management are now emerging.

The review made by Jamshidi et al [50] identified the need for a standardized migration framework. The lack of attention to crosscutting concerns and migration execution is also observed. The study also showed a lack of tool support to automate and facilitate Cloud migration tasks.

5.2. Design. The diversity of the APIs is natural, as each providers intends to offer something new or unique compared with other offers, in order to attract customers. The portability issue is therefore an issue for the management and governance levels where automation should be achieved as much as possible.

Limitations of abstraction layers include maintenance in response to changes made by a Cloud provider, and the limited coverage of provider functionalities.

The mix-in of services from different providers can be a strong argument in using an unique entry level instead a direct connection to only one provider, despite the overhead that is associated with the new layer. Therefore the management and service automation levels are the hot-spots of the development activities in the last three years.

An experimented developer can see a lack of flexibility in using a uniform interface or a model driven approach. The primary concern is the limitation of the abstractions to the set of smallest common subset of features. Following such approaches, a fully portable application can only be made at the level of the least capable service. This fact becomes a serious limitation when the applications need to be exploit the capabilities of the targeted Cloud.

While several libraries offering are offering diverse uniform interfaces, there is no wide acceptance of one or another proposal.

All tools that are offering unique entry points do not come with a structured approach, and the provided methods are at a technical level, thus, the application developer will typically be left hacking at code level rather than engineering the application following a structured tool supported methodology.

Once the application is deployed and adapted to a certain Cloud, in order to move it in another Cloud, an inspection of the source code is needed to identify the specific API calls or to build a model or representation of the code. Tools that can do that are only in early stage of prototyping and not yet available for large scale usage.

One strategy to avoid the limitation of a uniform interface with the smallest set it feature is to use a DSL to generate the code, and then to customize the generated code [10]. The customization should be loosely attached to the code to avoid overriding by subsequent updates.

Silva et al [4] made a comprehensive survey of Cloud lock-in solutions. They claim that there is no solution dealing with the impact of changes in the business process due to the use of one specific product, or working on issues related to the cross-organizational business in a portability scenario. Furthermore they claim that, from a research point of view, a challenge is to conduct more rigorous studies (e.g. only part of the studies of studies included a form of evaluation). Another challenge is increasing the use of empirical evidence to support the studies.

A roadmap for the Cloud software engineering was proposed by Da Silva and Lucredio [36]. The key-points of their roadmap are: solutions for avoidance of data lock-in; decision making about the migration towards the Cloud; legacy software migration; a re-engineering process for the Cloud migration; mechanisms to facilitate the hybrid Cloud; implementation of Modelling as a Service; Cloud service composition; case studies; open source platforms.

5.3. Run-time. At the infrastructure level, several partial solutions exists to migrate virtual machines, virtual storage or services. Despite the presence of standard OVF format, the virtual machines and their associated data are not yet ready for real-time migration. For example, Amazon is one of the few Cloud providers who are allowing to export virtual machines; however, their related resources (e.g. network, storage) cannot be exported too. In general, virtual machine images cannot be transferred from one hypervisor to another. The virtual machine images can be converted today with tools like qemu-image, but this requires to stop the virtual machine and to apply the adaptation off-line.

Most IaaS Clouds expose limited capabilities to control how a service behaves at run-time, beyond basic low-level scalability rules for VMs (once it has been deployed). Different Cloud services rely on specific rule engines to help enforcing the rules governing the service during its whole life-cycle. This fact reduces the portability chances of any given set of rules. The usage of a standard procedure for specifying the governance rules is needed to ensure the rule portability. According Moran et al [51], the Rule Interchange Format (RIF) is likely a candidate as specification language.

Portability degree needs to be detected by experiments. There are only few reports that are exposing such experiments. For example, Gonidis et al [5]: reported the results of an experiment carried out on cross-platform application development and deployment with four PaaS (OpenShift, App Engine, Heroku and Elastic Beanstalk).

5.4. Research agenda for Cloud portability. Following the comments that were exposed in the previous section, Table 5.2 proposes a research agenda for Cloud portability, with topics split on short and long term (next two years, respective a half decade).

6. Conclusions. In this paper, we conducted a survey on portability of the applications that are consuming Cloud services. We provided a taxonomy of Cloud portability, the latest solutions, reviewed existing works and identified the missing pieces needed to solve the problem. In the end, we also pointed out potentially future research directions and useful design guidelines for Cloud portability.

The focus of the paper was on the methodological and technical tools supporting the portability. However the porting a particular application requires also a decision methodology and an approach for the analysis of the destination Cloud environment, that were not discussed in this paper.

We hope this work provides a well-established framework that helps researchers to find existing proposals easily and to develop well founded future works.

Table 5.2: Research agenda for achieving Cloud portability

Approach	Short term	Long term
Standards	– Enhance the number of standard implementations	– Unified policy of the service level agreements
	– Establish standards for metrics, monitoring, accounting, security	– Establish standards for workload and data migration
	– Establish standards for machine-readable representations of services, quality, negotiations, processes	– Reference architecture for basic components of software consuming Cloud services
Design	– Support for decision making for Cloud migration	– Define re-engineering process for Cloud
	– Introduce Modelling-as-a-Service	– Mechanisms for code inspections and rewriting
	– Mechanism for service compositions	– Follow a structural approach in the design of the supporting tools
	– Build use cases and benchmarks for Cloud portability	– Ensure the portability of elasticity rule engines
	– Define the portability degree	– Combine automation with customization
Run-time	– Adopt open-source platforms	– Mechanisms for real-time migration
	– Increase the use of empirical evidence of portability	– Tools for the full service cycle, including Cloud governance
	– Automate re-deployments	– Open-source platforms ensuring automated portability or encompassing various approaches

Acknowledgment. The work of the first author is supported partially by AMICAS –Romanian grant PN-II-ID-PCE-2011-3-0260). The model-driven engineering part of the paper is a result of MODAClouds – European Commission grant FP7-ICT-2011-8-318484.

REFERENCES

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599 – 616, 2009. [Online]. Available: http://dx.doi.org/10.1016/j.future.2008.12.001

[2] D. Bradshaw, G. Folco, G. Cattaneo, and M. Kolding, "Quantitative estimates of the demand for cloud computing in Europe and the likely barriers to up-take," EDC, Tech. Rep. SMART 2011/0045, February 2012. [Online]. Available: http://cordis.europa.eu/fp7/ict/ssai/docs/study45-d2-interim-report.pdf

[3] T. Binz, G. Breiter, F. Leyman, and T. Spatzier, "Portable cloud services using TOSCA," *Internet Computing, IEEE*, vol. 16, no. 3, pp. 80–85, May 2012. [Online]. Available: http://dx.doi.org/10.1109/MIC.2012.43

[4] G. Silva, L. Rose, and R. Calinescu, "A systematic review of cloud lock-in solutions," in *2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)*, vol. 2, Dec 2013, pp. 363–368. [Online]. Available: http://dx.doi.org/10.1109/CloudCom.2013.130

[5] F. Gonidis, A. J. H. Simons, I. Paraskakis, and D. Kourtesis, "Cloud application portability: An initial view," in *Proceedings of the 6th Balkan Conference in Informatics*, ser. BCI '13. New York, NY, USA: ACM, 2013, pp. 275–282. [Online]. Available: http://doi.acm.org/10.1145/2490257.2490290

[6] D. Petcu, G. Macariu, S. Panica, and C. Crăciun, "Portable cloud applications-from theory to practice," *Future Gener. Comput. Syst.*, vol. 29, no. 6, pp. 1417–1430, Aug. 2013. [Online]. Available: http://dx.doi.org/10.1016/j.future.2012.01.009

[7] D. Petcu, "Portability and interoperability between clouds: Challenges and case study," in *Towards a Service-Based Internet*, ser. Lecture Notes in Computer Science, W. Abramowicz, I. Llorente, M. Surridge, A. Zisman, and J. Vayssire, Eds. Springer Berlin Heidelberg, 2011, vol. 6994, pp. 62–74. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-24755-2_6

[8] P. Mell and T. Grance, "The NIST definition of cloud computing," NIST Special Publication 800-145, 2011. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[9] J. Lenhard and G. Wirtz, "Measuring the portability of executable service-oriented processes," in *Enterprise Distributed Object Computing Conference (EDOC), 2013 17th IEEE International*, Sept 2013, pp. 117–126. [Online]. Available: http://dx.doi.org/10.1109/EDOC.2013.21

[10] A. Ranabahu and A. Sheth, "Semantics centric solutions for application and data portability in cloud computing," in *2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, Nov 2010, pp. 234–241. [Online]. Available: http://dx.doi.org/10.1109/CloudCom.2010.48

[11] S. Kolb and G. Wirtz, "Towards application portability in platform as a service," in *Proceedings of the 8th IEEE International Symposium on Service-Oriented System Engineering (SOSE), Oxford, United Kingdom*. IEEE Computer Society, 2014, pp. 218–229.

[12] K. Oberle and M. Fisher, "ETSI cloud – initial standardization requirements for cloud services," in *Economics of Grids, Clouds, Systems, and Services*, ser. Lecture Notes in Computer Science, J. Altmann and O. Rana, Eds. Springer Berlin Heidelberg, 2010, vol. 6296, pp. 105–115. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-15681-6_8

[13] M. Jonnalagedda, M. C. Jaeger, U. Hohenstein, and G. Kaefer, "Application portability for public and private clouds," in *Proceedings of the 1st International Conference on Cloud Computing and Services Science*. SciTePress, 2011, pp. 484–493. [Online]. Available: http://dx.doi.org/10.5220/0003394104840493

[14] A. N. Toosi, R. N. Calheiros, and R. Buyya, "Interconnected cloud computing environments: Challenges, taxonomy, and survey," *ACM Comput. Surv.*, vol. 47, no. 1, pp. 7:1–7:47, May 2014. [Online]. Available: http://doi.acm.org/10.1145/2593512

[15] G. Silva, L. Rose, and R. Calinescu, "Towards a model-driven solution to the vendor lock-in problem in cloud computing," in *2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)*, vol. 1, Dec 2013, pp. 711–716. [Online]. Available: http://dx.doi.org/10.1109/CloudCom.2013.131

[16] N. Loutas, E. Kamateri, F. Bosi, and K. Tarabanis, "Cloud computing interoperability: The state of play," in *2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)*, Nov 2011, pp. 752–757. [Online]. Available: http://dx.doi.org/10.1109/CloudCom.2011.116

[17] G. Lewis, "Role of standards in cloud-computing interoperability," in *2013 46th Hawaii International Conference on System Sciences (HICSS)*, Jan 2013, pp. 1652–1661. [Online]. Available: http://dx.doi.org/10.1109/HICSS.2013.470

[18] Z. Hill and M. Humphrey, "CSAL: A cloud storage abstraction layer to enable portable cloud applications," in *2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, Nov 2010, pp. 504–511. [Online]. Available: http://dx.doi.org/10.1109/CloudCom.2010.88

[19] C. Krintz, "The AppScale cloud platform: Enabling portable, scalable web application deployment," *IEEE Internet Computing*, vol. 17, no. 2, pp. 72–75, March 2013. [Online]. Available: http://dx.doi.org/10.1109/MIC.2013.38

[20] D. Petcu, B. Martino, S. Venticinque, M. Rak, T. Mahr, G. Lopez4, F. Brito, R. Cossu, M. Stopar, S. Sperka, and V. Stankovski, "Experiences in building a mOSAIC of clouds," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 2, no. 1, p. 12, 2013. [Online]. Available: http://www.journalofcloudcomputing.com/content/2/1/12

[21] M. Thabet and M. Boufaida, "An agent-based architecture and a two-phase protocol for the data portability in clouds," in *2013 27th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, March 2013, pp. 785–790. [Online]. Available: http://dx.doi.org/10.1109/WAINA.2013.129

[22] A. Rafique, S. Walraven, B. Lagaisse, T. Desair, and W. Joosen, "Towards portability and interoperability support in middleware for hybrid clouds," in *Proceedings of the 2014 IEEE INFOCOM Workshop on Cross-Cloud Systems*. IEEE Computer Society, 2014, pp. 7–12.

[23] D. Cunha, P. Neves, and P. Sousa, "Interoperability and portability of cloud service enablers in a PaaS environment," in *Proceedings of the 2nd International Conference on Cloud Computing and Services Science*. SciTePress, 2012, pp. 432–437. [Online]. Available: http://dx.doi.org/10.5220/0003959204320437

[24] D. Zeginis, F. D'Andria, S. Bocconi, J. G. Cruz, O. C. Martin, P. Gouvas, G. Ledakis, and K. A. Tarabanis, "A user-centric multi-PaaS application management solution for hybrid multi-cloud scenarios," *Scalable Computing: Practice and Experience*, vol. 14, no. 1, 2013. [Online]. Available: http://www.scpe.org/index.php/scpe/article/view/824

[25] E. Kamateri, N. Loutas, D. Zeginis, J. Ahtes, F. DAndria, S. Bocconi, P. Gouvas, G. Ledakis, F. Ravagli, O. Lobunets, and K. Tarabanis, "Cloud4soa: A semantic-interoperability paas solution for multi-cloud platform management and portability," in *Service-Oriented and Cloud Computing*, ser. Lecture Notes in Computer Science, K.-K. Lau, W. Lamersdorf, and E. Pimentel, Eds. Springer Berlin Heidelberg, 2013, vol. 8135, pp. 64–78. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-40651-5_6

[26] F. Giove, D. Longoni, M. S. Yancheshmeh, D. Ardagna, and E. Di Nitto, "An approach for the development of portable applications on PaaS clouds," in *Proceedings of the 3rd International Conference on Cloud Computing and Services Science*. SciTePress, 2013, pp. 591–601. [Online]. Available: http://dx.doi.org/10.5220/0004511605910601

[27] F. Moscato, R. Aversa, B. Di Martino, T. Fortis, and V. Munteanu, "An analysis of mOSAIC ontology for cloud resources annotation," in *2011 Federated Conference on Computer Science and Information Systems (FedCSIS)*, Sept 2011, pp. 973–980.

[28] G. Cretella and B. Di Martino, "Towards a semantic engine for cloud applications development," in *2012 Sixth International Conference on Complex, Intelligent and Software Intensive Systems (CISIS)*, July 2012, pp. 198–203. [Online]. Available: http://dx.doi.org/10.1109/CISIS.2012.159

[29] L. Youseff, M. Butrico, and D. Da Silva, "Toward a unified ontology of cloud computing," in *Grid Computing Environments Workshop, 2008. GCE '08*, Nov 2008, pp. 1–10. [Online]. Available: http://dx.doi.org/10.1109/GCE.2008.4738443

[30] E. M. Maximilien, A. Ranabahu, R. Engehausen, and L. C. Anderson, "Toward cloud-agnostic middlewares," in

Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications, ser. OOPSLA '09. New York, NY, USA: ACM, 2009, pp. 619–626. [Online]. Available: http://dx.doi.org/10.1145/1639950.1639957

[31] D. Bernstein and D. Vij, "Using semantic web ontology for Intercloud directories and exchanges," in *2010 International Conference on Internet Computing*, H. R. Arabnia, V. A. Clincy, J. Lu, A. Marsh, and A. M. G. Solo, Eds. CSREA Press, July 2010, pp. 18–24.

[32] N. Loutas, E. Kamateri, and K. Tarabanis, "A semantic interoperability framework for cloud platform as a service," in *2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)*, Nov 2011, pp. 280–287. [Online]. Available: http://dx.doi.org/10.1109/CloudCom.2011.45

[33] G. Cretella and B. Di Martino, "Towards automatic analysis of cloud vendors APIs for supporting cloud application portability," in *2012 Sixth International Conference on Complex, Intelligent and Software Intensive Systems (CISIS)*, July 2012, pp. 61–67. [Online]. Available: http://dx.doi.org/10.1109/CISIS.2012.162

[34] R. Dukaric and M. B. Juric, "Towards a unified taxonomy and architecture of cloud frameworks," *Future Generation Computer Systems*, vol. 29, no. 5, pp. 1196 – 1210, 2013, special section: Hybrid Cloud Computing. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X12001793

[35] B. Di Martino, "Applications portability and services interoperability among multiple clouds," *IEEE Cloud Computing*, vol. 1, no. 1, pp. 74 – 77, 2014. [Online]. Available: http://dx.doi.org/10.1109/MCC.2014.1

[36] E. Da Silva and D. Lucredio, "Software engineering for the cloud: A research roadmap," in *2012 26th Brazilian Symposium on Software Engineering (SBES)*, Sept 2012, pp. 71–80. [Online]. Available: http://dx.doi.org/10.1109/SBES.2012.12

[37] A. Samba, "Logical data models for cloud computing architectures," *IT Professional*, vol. 14, no. 1, pp. 19–26, Jan 2012. [Online]. Available: http://dx.doi.org/10.1109/MITP.2011.113

[38] M. Shirazi, H. C. Kuan, and H. Dolatabadi, "Design patterns to enable data portability between clouds' databases," in *2012 12th International Conference on Computational Science and Its Applications (ICCSA)*, June 2012, pp. 117–120. [Online]. Available: http://dx.doi.org/10.1109/ICCSA.2012.29

[39] B. Di Martino, G. Cretella, and A. Esposito, "Semantic and agnostic representation of cloud patterns for cloud interoperability and portability," in *2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)*, vol. 2, Dec 2013, pp. 182–187. [Online]. Available: http://dx.doi.org/10.1109/CloudCom.2013.123

[40] A. Ranabahu, E. Maximilien, A. Sheth, and K. Thirunarayan, "Application portability in cloud computing: An abstraction driven perspective," *IEEE Transactions on Services Computing*, vol. PP, no. 99, pp. 1–1, 2013. [Online]. Available: http://dx.doi.org/10.1109/TSC.2013.25

[41] M. Caballer, C. de Alfonso, G. Molta, E. Romero, I. Blanquer, and A. Garcia, "CodeCloud: A platform to enable execution of programming models on the clouds," *Journal of Systems and Software*, no. 0, pp. –, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2014.02.005

[42] D. Ardagna, E. Di Nitto, G. Casale, D. Petcu, P. Mohagheghi, S. Mosser, P. Matthews, A. Gericke, C. Ballagny, F. D'Andria, C. Nechifor, and C. Sheridan, "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)*, June 2012, pp. 50–56. [Online]. Available: http://dx.doi.org/10.1109/MISE.2012.6226014

[43] E. Di Nitto, M. A. Almeida da Silva, D. Ardagna, G. Casale, C. D. Craciun, N. Ferry, V. Muntes, and A. Solberg, "Supporting the development and operation of multi-cloud applications: The ModaClouds approach," in *2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, Sept 2013, pp. 417–423. [Online]. Available: http://dx.doi.org/10.1109/SYNASC.2013.61

[44] D. Petcu, E. Di Nitto, D. Ardagna, A. Solberg, and G. Casale, "Towards multi-clouds engineering," in *2014 IEEE INFOCOM, Workshop on Cross-Cloud Systems (CrossCloud)*, April 2014, p. in print.

[45] G. Blair, N. Bencomo, and R. France, "Models@ run.time," *Computer*, vol. 42, no. 10, pp. 22–27, Oct 2009. [Online]. Available: http://dx.doi.org/10.1109/MC.2009.326

[46] A. Menychtas, C. Santzaridou, G. Kousiouris, T. Varvarigou, L. Orue-Echevarria, J. Alonso, J. Gorronogoitia, H. Bruneliere, O. Strauss, T. Senkova, B. Pellens, and P. Stuer, "ARTIST Methodology and Framework: A Novel Approach for the Migration of Legacy Software on the Cloud," in *2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, Sept 2013, pp. 424–431. [Online]. Available: http://dx.doi.org/10.1109/SYNASC.2013.62

[47] N. Ferry, F. Chauvel, A. Rossini, B. Morin, and A. Solberg, "Managing multi-cloud systems with CloudMF," in *Proceedings of the Second Nordic Symposium on Cloud Computing & Internet Technologies (NordiCloud)*, 2013, pp. 38-45. [Online]. Available: http://dx.doi.org/10.1145/2513534.2513542

[48] A. Sadovykh, C. Hein, B. Morin, P. Mohagheghi, and A.J. Berre, "REMICS – REuse and Migration of Legacy Applications to Interoperable Cloud Services," in *Towards a Service-Based Internet*, ser. Lecture Notes in Computer Science, W. Abramowicz, I. Llorente, M. Surridge, A. Zisman, and J. Vayssire, Eds. Springer Berlin Heidelberg, 2011, vol. 6994, pp. 315–316. [Online]. Available: Available: http://dx.doi.org/10.1007/978-3-642-24755-2_32

[49] N. Ferry, A. Rossini, F. Chauvel, B. Morin, and A. Solberg, "Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems," in *2013 IEEE 6th International Conference on Cloud Computing (CLOUD)*, L. O'Conner, Ed. IEEE Computer Society, 2013, pp. 887–894. [Online]. Available: http://dx.doi.org/10.1109/CLOUD.2013.133

[50] P. Jamshidi, A. Ahmad, and C. Pahl, "Cloud migration research: A systematic review," *IEEE Transactions on Cloud Computing*, vol. 1, no. 2, pp. 142–157, July 2013. [Online]. Available: http://dx.doi.org/10.1109/TCC.2013.10

[51] D. Moran, L. Vaquero, and F. Galan, "Elastically ruling the cloud: Specifying application's behavior in federated clouds," in *2011 IEEE International Conference on Cloud Computing (CLOUD)*, July 2011, pp. 89–96. [Online]. Available:

http://dx.doi.org/10.1109/CLOUD.2011.53

Edited by: Marcin Paprzycki
Received: Aug 21, 2014
Accepted: Sept 21, 2014