Chapter 1:
A framework for cloud interoperability based on compliance and conformance

José Carlos Martins Delgado

Department of Computer Science and Engineering

Instituto Superior Técnico, Universidade de Lisboa

Av. Prof. Cavaco Silva, Taguspark, 2744-016 Porto Salvo, Portugal

jose.delgado@ist.utl.pt

Abstract: The current cloud computing panorama includes many cloud providers, each with its own model, set of services and API (Application Programming Interface), leaving users with an interoperability problem when trying to avoid a potential dependency on a specific provider. Current approaches tend to tackle this problem (user to cloud or cloud to cloud) by abstracting it, either by providing a common API, which has to map onto each cloud's API, or by introducing brokers that adapt the views of the user and of the cloud. This chapter proposes another approach that tries to solve the problem at its source, by defining a common service and resource model, a small set of common services (core API), an interoperability mechanism based on compliance and conformance and an extensibility mechanism that allows providers to build different clouds, based on this core and with support for interoperability. The chapter also presents an interoperability framework with three dimensions: lifecycle of services, levels of abstraction in interoperability and concerns, entailing aspects such as security, quality of service (SLA, Service Level Agreement) and financial aspects. The main goal is not to provide an interoperability solution to existing systems but rather to establish a foundation layer for cloud computing that shows how clouds should be organized to cater for provider differentiation while supporting interoperability from scratch.

Keywords: Cloud computing, Interoperability, Framework, Compliance, Conformance, Service, Resource

1.1 Introduction

In practical terms, *cloud computing* can simply be defined as the remote creation and use of computer-based resources and services, in a setting characterized by elastic,

dynamic and automated resource provisioning, paid as used and managed in a self-service way [1].

Resource virtualization is the key enabling factor. From a pool of physical resources (servers, storage, networks, and so on), virtual resources can be dynamically allocated and provisioned, or decommissioned and released, to form an apparently elastic fabric of resources that are used on demand and paid as used. However, users want the services that resources support, not the resources themselves. Resources are becoming a commodity [2], allowing some ICT (Information and Communications Technology) enterprises, the providers, to specialize in providing resource infrastructures cheaper, more reliably, better managed, faster provisioned and in a more scalable way than any of the organizations that just require these resources for their lines of business, the users.

This growing dichotomy between users and providers, known as *utility computing* [3], is a marriage of convenience, as any outsourcing agreement. Providers alleviate users from many issues, including expertise, risks, costs and management, in what concerns resources and generic services. The user still has a share of the overall solution in application-specific services, but at least this means that the user is leaning more towards the core business than the ICT technologies that support it.

Moving on from an initial period of slow growth, in which concerns about security, privacy, performance and availability were inhibiting factors, cloud computing finally took the world by storm. Essentially, it becomes so easy, cheap and fast to get computing resources that conventional ICT simply cannot compete. The advantages have now more weight than the risks and disadvantages. It seems that there is not a single large ICT provider that is not investing heavily in cloud computing.

This is clearly a market driven by providers, with users still cautious with the transition, but the scenario is evolving at a fast pace, including not only individuals but also enterprises as customers. Two of the most recent examples of this, at the time of writing (July 2013), are:

- Amazon is driving a price war with other resource providers and has sliced prices of dedicated instances (virtual servers running on dedicated physical servers) by up to 80% [4];
- Two giants, Salesforce and Oracle, signed a partnership to increase their combined weight over competition in the cloud computing service market [5].

Gartner [6] predicts that the worldwide market of public cloud services will grow 18.5% in 2013, to a total of US$131 billion worldwide. The growth in 2012 has been 16.8% and this rate should be sustainable for the next 5 years.

This is comparable only to the Web revolution, 20 years ago. It seems that both users and providers were just waiting for a technology that would interconnect them in a flexible and effective manner, so that the goals of everybody can be met. Two main factors also gave their precious contribution to the bootstrap of cloud computing:

- From the point of view of consumers, social networking and multiplatform mobility (laptops, tablets and smart phones) raised the pressing need of storing information in a server somewhere, always available to be accessed from anywhere and synchronized across platforms;
- From the point of view of enterprises and service providers, the market pressure from increasing global competition and ever-shortening turnaround times, combined with a sluggish global economy, emphasized the basic principle of concentrating on core business and (dynamically) outsourcing the rest.

However, we are still facing the same problem that drove the appearance of the Web: *interoperability*. The goal is to endow distributed systems with the ability of meaningfully exchanging information in interaction patterns known as *choreographies*. Today, unfortunately, the problem is even worse than 20 years ago:

- The Web allowed uniform e global access to media information and appeared before the market, creating it instead of reacting to it. This gave time to standards (HTTP, HTML and, later, XML) to be established before diversity could set in. This is why today we can use any browser to access any Web site. Even in the service realm, either with SOA or REST, the scenario is basically standardized (although standards are not enough to ensure interoperability [7]);
- The cloud enables global access to all kinds of computer-based resources, in a very dynamic environment, but these are more complex than mere hypermedia documents and the market exploded before standardization was achieved. This means that today there are many cloud providers with incompatible interfaces [8] and we cannot use the various clouds seamlessly.

Clouds are becoming the new datacentres, now virtualized and much more dynamic. In principle, this means that it should be much easier to change the provider of resources and to discover and use the most convenient services. Unfortunately, this is usually not the case. Complexity, diversity and the lack of standardization in interoperability lead users to *lock-in*, since the costs of changing provider are typically higher than the benefits of the optimizations stemming from the free choice of a new provider. Market share and reputation become the main drivers for the initial choice of provider, instead of a continuous analysis of service quality.

Therefore, we have the problem of standardizing a market that is blossoming at a fast pace, with a war for market share between providers, battling with innovation, prices and alliances. Will current standardization efforts stand a change in such an environment, or will the stronger providers just impose *de facto* specifications?

The main goals of this chapter are:

- To discuss the interoperability problem and to get a better grasp of its dimensions;

- To assess the limitations and problems of current efforts towards solving this problem;
- To present a new approach and to discuss its relevance in dealing with concerns such as adaptability, changeability and reliability;
- To contribute to the systematization of interoperability in the realm of cloud computing, by establishing a foundation layer that shows how clouds should be organized to cater for provider differentiation, while helping to deal with interoperability issues in a coherent way, right from scratch.

The chapter is organized pursuant to these goals. Section 1.2 provides some background information. Section 1.3 discusses the problem to understand it in a more profound way. Section 1.4 discusses the limitations of current approaches. Section 1.5 presents the new approach. Section 1.6 compares it to current approaches and discusses the benefits, the risks and limitations. Section 1.7 provides some hints into future directions of research. Section 1.8 draws conclusions on this matter.

1.2 Background

This section provides a brief review of the vast existing work on cloud computing, in particular in the interoperability slant, and by no means is intended as an exhaustive study. This is complemented by further information provided in sections 1.3 and 1.4.

1.2.1 Cloud characterization

A cloud is a platform onto which computer based resources and services can be deployed for later use, in a way that can be described as:

- Virtual (abstracting many deployment details);
- Elastic (deploy/decommission as much as needed);
- Dynamic (changes can be frequent and are quick to implement);
- Utility-like (payment proportional to use, which is measured);
- Self-service (automated, on-demand access);
- Omnipresent (accessed through a network, from anywhere).

One of the most cited definitions of cloud computing, encompassing these characteristics, is given by the US National Institute of Standards and Technology (NIST) [9]. Although many variants of the service models available are possible, the NIST considers the following three:

- Infrastructure as a Service (IaaS). The user gets essentially raw resources and must deploy services to them;
- Platform as a Service (PaaS). Resources already have basic and common services, but the user must still deploy the application-specific services;
- Software as a service (SaaS). Resources are provided ready to use, with services deployed. The user may have to configure them before use.

The NIST also considers cloud deployment models, namely private, community, public and hybrid clouds. The first three express the relationship between the owners and the users of the cloud, whereas the latter refers to composition of several clouds, in which the interoperability issue is of particular relevance.

Several survey papers [1, 10, 11] discuss the most relevant issues involving cloud computing.

1.2.2 Cloud APIs and standards

There are many cloud providers at the IaaS level, the most developed one. Besides large providers with proprietary APIs (Application Programming Interfaces), such as Amazon, Microsoft and Google, there is also a strong open-source movement in the cloud market [12], with cloud management platforms such as OpenStack [13], CloudStack [14] and Eucalyptus [15]. Although not open source, VMWare's vCloud [16] is used as the underlying platform by several cloud providers. OpenStack and vCloud seem to be the most popular platforms among the non-proprietary IaaS providers.

Nevertheless, popular cloud management systems are not a solution to prevent lock-in. Each cloud ends up having its own features and characteristics, since cloud providers need differentiation to attract customers, and there are several proprietary cloud providers, usually the largest ones.

Several standards have been proposed to help solving the cloud interoperability problem. If clouds can interoperate, a user (individual or enterprise) can use several clouds and minimize lock-in.

At the IaaS level, CIMI (Cloud Infrastructure Management Interface) [17], a DTMF (Distributed Management Task Force, Inc.) standard since 2012, provides an API to provision and manage resources typically found in clouds (such as virtual machines, storage volumes and networks).

OCCI (Open Cloud Computing Interface) [18] is another standard, produced by the OGF (Open Grid Forum) in 2011, which entails a more general and higher-level resource model. It is described by three documents, which specify a generic core (applicable to IaaS, PaaS, SaaS and even non-cloud environments), IaaS resources and a RESTful HTTP rendering of the API.

CMDI (Cloud Data Management Interface) [19] is a SNIA (Storage Networking Industry Association) standard, adopted by ISO/IEC, which provides a RESTful API to deal with storage resources in a cloud.

OVF (Open Virtual Format) [20], a DTMF standard adopted by ISO/IEC, allows applications to be packaged and deployed to virtualized systems. Given its low level and usefulness, it is the most used standard in cloud computing and constitutes a means to promote portability of applications between clouds.

TOSCA (Topology and Orchestration Specification for Cloud Applications) [21] is a specification being developed by OASIS (Advancing Open Standards for Information Systems). At the time of writing (July 2013), a first version has been produced [22]. Like OVF, TOSCA is intended for application packaging and distribution, but at higher level, emphasizing the services in their entire lifecycle (design, deployment, monitoring and maintenance) rather than the infrastructure components that support those services.

DTMF is also working towards a standard for cloud audit and governance, Cloud Auditing Data Federation (CADF), to allow examining data and events in applications, even if they extend across several clouds. A working draft is available from the DTMF site.

Existing security standards and open specifications for distributed environments (such as OAuth, OpenID and SAML – Security Assertion Markup Language) are used in the cloud computing context, since there are no specific cloud security standards. Nevertheless, there are guidelines in the area of GRC (Governance, Risk and Compliance), from the CSA (Cloud Security Alliance), and in the area of security and privacy, by the NIST (which has also produced a document on a Security Reference Architecture).

The ITU (International Telecommunication Union) has a cloud computing Focus Group, which has produced a set of documents (available from the ITU site) regarding the cloud computing domain, with emphasis on the telecommunication services in a multiple cloud environment, in which interoperability is a major concern.

The IEEE Cloud Computing Initiative (CCI) represents another standardization effort, with particular emphasis on cloud interoperability and two foreseen standards: IEEE P2301 (Guide for Cloud Portability and Interoperability Profiles) and IEEE P2302 (Standard for Intercloud Interoperability & Federation). The latter has produced a working draft, outlining the architecture of the intercloud, a network of interconnected clouds that establishes, at a higher level, a parallel with the Internet (seen as a network of interconnected servers).

In [23], a good summary of existing standardization efforts is provided, with particular emphasis on cloud interoperability and on the role that standards can play in the field of cloud computing.

1.2.3 Other approaches to interoperability

Without immediate standardization concerns, there is a vast literature on the cloud interoperability problem [24, 25, 26], encompassing many aspects and proposals.

Cloud interoperability is a complex issue, but we can distinguish two main situations in a service's lifecycle that require it: at deployment time (*portability*) and at operation time (*integration*).

Portability [27] entails the ability to deploy the application that implements a service to several clouds, or to dynamically migrate it from one cloud to another, with minimal effort. Specifications for service packaging and distribution, such as OVF [20] and TOSCA [21], are fundamental to achieve this goal, by providing a common means to deploy and to migrate services.

Nevertheless, portability is not enough. A user may need to use or to manage services in several clouds, or a service may be deployed across several clouds and its components need to interoperate and to be managed in an integrated fashion. This requires integration, for which several approaches exist.

One of the approaches is to recognize that working with a cloud implies invoking the features of its API and therefore interoperability can be achieved by using a common API, most likely organized into layers. This provides an abstraction that needs to be mapped onto the specific APIs of cloud providers. Examples of this approach are a cross-platform API [28], an object-oriented abstraction of cloud resources [29] and an abstraction layer for cloud storage resources [30].

However, syntactic integration is limited because the semantics of services cannot be expressed and have to be dealt with tacitly, by users and developers. Besides basic semantic information (e.g., semantic web services), we need semantic registries [31], semantic frameworks [32, 33] and cloud ontologies [34, 35, 36].

The interoperability in the context of multiple clouds has been tackled by active research [8, 37, 38, 39]. The organization of multiple clouds as an intercloud, considered by the IEEE Cloud Computing Initiative, has been described in the literature [40, 41, 42].

We should also be aware that interoperability is not an exclusive problem of cloud computing or any virtualized ICT system. Distributed applications, whether supported by a cloud or by a conventional data centre, have the same basic interoperability problems. There is an ongoing effort to systematize an interoperability body of knowledge, as a foundation for an interoperability science [43], and several interoperability frameworks have been proposed [44, 45].

1.3 Understanding the interoperability problem

Deriving a good solution implies that the corresponding problem must first be well understood. The Cloud Computing Use Case Discussion Group defines cloud interoperability in terms of writing code that is able to work with more than one cloud provider simultaneously [46], whereas in [25] it is defined as the ability for multiple cloud providers to work together. A broader set of situations that require cloud interoperability is described in [47].

The goal of this section is to describe scenarios, actors, concerns and use cases that require interoperability, as a first step to identify foundational interoperability concepts and to derive a model that can express them in an orthogonal way.

1.3.1 Basic scenarios

The current global computing scenario is not limited to clouds and neither is interoperability. The classical ICT setting is rather static, with typical N-tier enterprise information systems deployed to a single data centre, eventually including a cluster of servers. Users and developers are typical actors, both accessing applications in that data centre and in other web servers. Figure 1 illustrates this scenario.

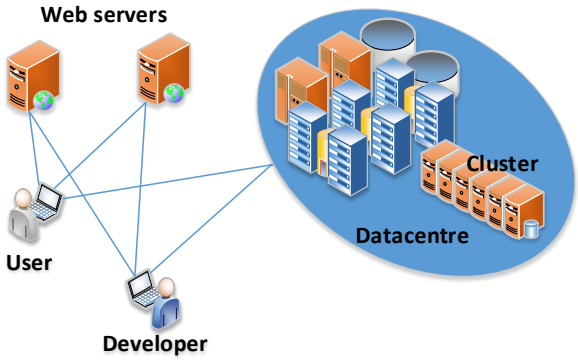

Figure 1: Classical ICT scenario, with enterprise applications in data centres and global access to web servers.

Today, the scenario can be drastically different, with a plethora and mix of disparate computing systems that need to interoperate, as illustrated by Figure 2.

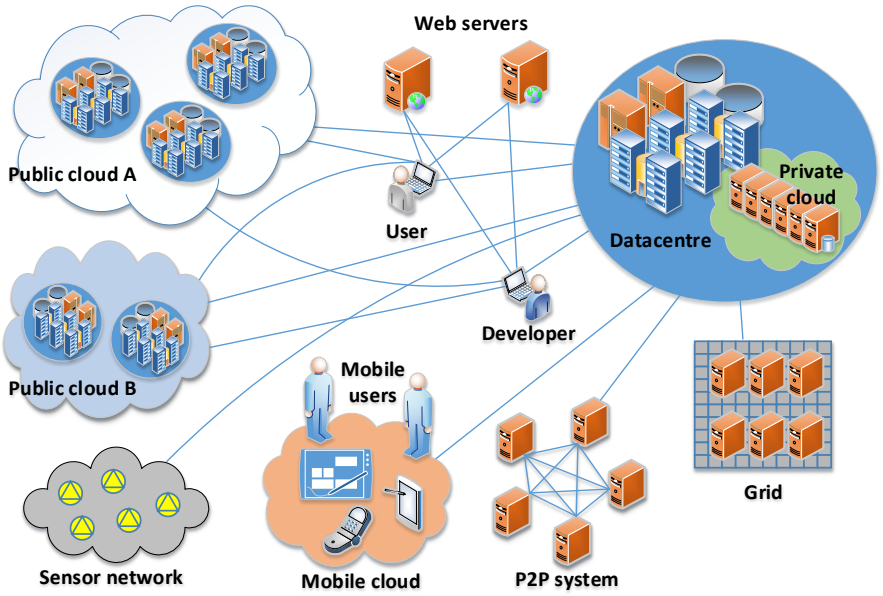

Figure 2: Modern ICT scenario, with enterprise applications deployed to hybrid clouds and integrated with non-cloud systems.

The following situations are now likely to occur:

- The datacentre still exists, to hold critical or sensitive data, but it will probably include a private cloud instead of a mere cluster;
- The enterprise applications will be deployed to a hybrid cloud setting, integrating the enterprise's owned infrastructure with one or more public clouds;
- Mobile cloud computing [48] is another possibility, given the ever increasing pervasiveness of smartphones and tablets that created a surge in the BYOD (Bring Your Own Device) tendency [49];
- The explosive development of the IoT (Internet of Things) [50] and of RFID tags [51] in supply chains raises the need to integrate the enterprise applications with the physical world, namely through sensor networks [52];
- Grids [53] and P2P systems [54] can be used for batch processing and specific applications.

For simplicity, not all possible connections are depicted in Figure 2, but the inherent complexity of integrating all these systems is easy to grasp. Such heterogeneous scenarios have been compared to a jungle of computer-based systems, by using the designation *jungle computing* [55].

Tables 1 and 2 provide a characterization of the main types of these systems. Table 2 distinguishes resources (provider view) from the services (user view) that

they implement. This distinction is important in systematizing the problem because it provides a mapping between an idealized vision (the world as a set of interconnected services) from a more practical one (a heterogeneous set of computing infrastructures that provide resources to implement services).

Table 1: General characterization of the main types of computing infrastructures.

Infrastructure model	Simple description	Main distinguishing features	Network	Dispersion
Server	Physical or virtual	Always on computing resource	None	None
Cluster	Set of servers	Load balancing, reliability	LAN	None
Datacentre	Set of servers/clusters	Centralized management and physical infrastructures	LAN	None
Web	Global set of HTTP servers	Global access to information and services	Internet	Global
Peer to peer	Set of peers	Decentralized cooperation	Any	Large
Grid	Set of workers	Opportunistic performance	Internet	Large
Cloud-IaaS	Set of raw resources	Utility-style provisioning of raw resources	Internet	Small
Cloud-PaaS	Set of resources with basic services	Utility-style provisioning of preconfigured resources	Internet	Small
Cloud-SaaS	Remote resource with an application	Utility-style use of an existing application	Internet	Small
Multi-cloud	Set of clouds	Brokered or seamless usage of more than one cloud	Internet	Large
Intercloud	Global set of clouds	Brokered or seamless usage of all clouds	Internet	Global
Jungle	Global set of networked resources	Brokered or seamless usage of all networked resources	Any	Global

Table 2: Detailed characterization of the main types of computing infrastructures, providing a distinction between resources and services.

	Resources						Services		
Infrastructure model	Scale	Size (servers)	Heterogeneity	Owners	Node manager	Elasticity	Nature	SLA	Cost to users
Server	Small	1	None	One	Owner	Virtualization	Generic	Ownership	Full
Cluster	Medium	10-100s	None	One	Owner	On-supply	Generic	Ownership	Full
Datacentre	Large	100-1000s	Medium	One	Owner/users	On-supply	Generic	Ownership	Full
Web	Global	Millions	High	Many	Owners	On-supply	Generic	As allowed	Free/contract
Peer to peer	Large	100-1000s	High	Many	Owners	On-supply	Specific	As allowed	Free/contract
Grid	Huge	1000s	High	Many	Owners	On-supply	Specific	As allowed	By use
Cloud-IaaS	Large	100-1000s	Low	One	Owner/users	On-demand	Generic	As needed	By use
Cloud-PaaS	Large	100-1000s	Medium	One	Owner/users	On-demand	Generic	As needed	By use
Cloud-SaaS	Small	1	Low	One	Owner	On-demand	Specific	As needed	By use
Multi-cloud	Huge	1000s	High	Several	Owners/users	On-demand	Generic	As needed	By use
Intercloud	Global	Millions	High	Many	Owners/users	On-demand	Generic	As needed	By use
Jungle	Global	Billions	Huge	Many	Mixed	Mixed	Mixed	Mixed	Mixed

1.3.2 Actors, roles, goals and expectations

In complex systems, there are usually many *actors* involved, each with a different set of motivations, assumptions, expectations, goals and contributions. This richness is fundamental to understand the full scope of the interoperability problem.

We must distinguish *role* from *actor*. Each individual or organizational actor can play, at a time or simultaneously, more than one role in the system. For example, a programmer is an individual actor that performs both the roles of *developer* (creating services) and *user* (invoking services during development or any other activity). We use the role, rather than the actor, to establish the motivations for interoperability, because it is the best way to organize needs and expectations in an orthogonal way.

Table 3 contemplates some of the most relevant roles and, in a simplified way, describes the main motivations and expectations of each with respect to interoperability. The cloud is used as the archetype of a computer-based infrastructure, but the issues are basically the same in all the infrastructure types described in Tables 1 and 2.

Table 3: Motivations and expectations regarding interoperability of the main roles performed by actors involved in computer-based systems.

Role	Simple description	Wish list
User	Configures, uses and pays services (no distinction made between individual and enterprise users)	• Compatible cloud APIs (to use, configure and manage applications) • Compatible security, SLA and business models • Transparent choice of provider, for best SLA/cost ratio • Single point of contact (provider or broker)
Developer	Creates, deploys and manages services	• Compatible platforms and service libraries • Choice of a single development environment • Develop once, run anywhere • Transparent support for dynamic migration and cloud bursting
Provider	Owns resources and rents them by dynamic provisioning	• Compatible cloud APIs (to get more customers and third-party services, to use other clouds for resource bursting, to act as a broker with innovative services) • Mechanism for cloud API extension, to enable differentiation from competition • Standards with several layers of conformance

		• Sectorial standards (covering specific topics), instead of large standards covering everything
Broker	Adapts and aggregates services	• User + Developer + Provider, since a broker is a mix of all three roles • Many different providers and third-party services, to increase the broker's value
Auditor	Audits and certifies services and infrastructures	• Abundant standards, covering almost all aspects • Conformance to standards by almost all providers • Very limited extension mechanisms, to avoid variability that is difficult to audit
Consultant	Provides knowledge and solutions	• Many standards • Many different providers and third-party services • Many extension mechanisms
SDO (Standards Developing Organization)		• Universal adoption of its standards • Breadth of applicability • Completeness of specifications

In summary:

- Users, developers and auditors are not fond of heterogeneity or variability, but users and developers want to be able to choose among competing providers or development environments;
- Providers acknowledge that being compatible with others may bring more customers due to a lower entry barrier, but the same argument is also valid to lose them to competitors. Therefore, extensibility, innovation and differentiation are of prime importance;
- Brokers and consultants get their business from variability, by reducing its effects to other roles. Therefore, they require interoperability (so that solutions exist) but not normalization (so that variability is possible);
- SDOs try to counter market variability by producing widely adopted specifications. However, SDOs are not globally coordinated and standards partially overlap and/or compete. Also, the fast pace of technology evolution hinders most current standardization efforts. The realm of cloud computing is still ruled by the strongest providers.

These goals are somewhat conflicting and actors performing more than one role may adopt different postures, according to the project in which they happen to be involved.

1.3.3 Use cases and concerns

To understand interoperability, we need to identify which situations require it. A good description of use cases is made in [23], including seven use cases (on the topic of deployment) from the Cloud Computing Use Cases White Paper [46], 21 from NIST (on management, use and security) and 14 from DTMF (on management).

A list of requirements for multi-cloud interaction is presented in [8], regarding development, deployment and execution of applications, spanning across the lifecycle of applications.

This plethora of use cases, scenarios and requirements is not detailed here, for simplicity. If we compare the APIs of large clouds, such as Amazon AWS, Microsoft Azure and Rackspace, we can easily understand why standardization is so difficult in cloud computing. Too many providers, specifications and users have appeared into play before the technology had the change to follow an organized and controlled route.

Many of these use cases are high level and depend on the architecture of the specific cloud, but others are recurrent and are present in all clouds (such as virtual machine provisioning and storage access), although differing in the details. The former refer to specific services, representing the differentiation of each provider, and are not particularly interesting to include in an interoperability effort. The latter constitute the basis for an interoperability systematization and an eventual standardization proposal.

Any interaction with a cloud (from another cloud, user, developer, etc.) requires interoperability, even if it pertains to some feature exclusive of that cloud. Here, we are interested in the most common features, those that a typical cloud implements. A cloud API can be structured in categories of features, according to each type of resources or services with most relevance, as illustrated by Table 4.

Table 4: Typical categories of features of a cloud API.

Service model	Category	Main aspects/features
IaaS	Workload (virtual machine images)	Hypervisors, image format, deployment, migration, load balancing, management
IaaS	Persistent data	Data models, storage and migration
IaaS	Network	Virtual networks, name resolution, policies, management
IaaS	Identity Access Management	Authentication, authorization, account management
PaaS	Queue and notification	Asynchronous messaging and eventing
PaaS	Database	Queries, management and administration
PaaS	Workflow management	Task coordination, scheduling

PaaS	Content delivery	Web caching, request routing, server load balancing
PaaS	Middleware and libraries	Support for application development
SaaS	User application	Web Services, RESTful interface

These categories of features exist, in one form or another, in most clouds currently offered on the market (if they support the indicated service models). However, the underlying resource and service architectures, as well as the APIs, can be quite different. The higher-level the service model is, the more difficult interoperability becomes.

There are also transversal concerns, present in most, if not all, interactions with a cloud, such as:

- Security;
- Service level agreements;
- Reliability and disaster recovery;
- Metering and monitoring;
- Cost and charging models;
- Regulatory compliance and legislation;
- Auditing and risk management;
- Strategy and governance;
- Policies and rules;
- Management and control frameworks and standards.

Some of these concerns need to be addressed explicitly and programmatically (by using the API), whereas others can only be dealt with in a tacit manner or at the documentation level.

1.4 Analysis of current solutions to the interoperability problem

Standards are the canonical solution to interoperability problems but, depending on the perspective (Table 3), the obligation to adopt some specification can be seen as both a bonus and a curse.

On one hand, it puts everyone on the same ground, provides a concrete target for development and testing and enables interoperability. Only lack of conformance to the standard can still originate problems. However, on the other hand, a standard can behave as a straightjacket and hamper innovation, differentiation from competition and customization.

In the general case, standards are more effective (with a broader acceptance) when:

- They stem from real cooperation between competing specifications. This was the case of the genesis of UML, in which three leading gurus of the modelling domain, James Rumbaugh, Grady Booch and Ivar Jacobson truly cooperated to fuse their three approaches, respectively Object-Modelling Technique (OMT), Object-Oriented Design (OOD) and Object-Oriented Software Engineering (OOSE);
- They appear as the result of a perceived need and before the market expands (e.g., HTTP, HTML, XML, Web Services);
- What a standard specifies is not the core business in itself (although supporting it) but is seen as useful. This is typically the case of standards pertaining to low-level aspects, such as OVF (Open Virtual Format) [20].

Unfortunately, in the case of cloud computing, the market expanded before standards, driven by innovation and perceived customer needs. When this happens, either we have a set of incompatible specifications or one dominates and takes the form of a *de facto* standard. Amazon and Salesforce have been early adopters and became leaders in their cloud market segments. Others, such as Microsoft, appeared later in the scenario with enough quality to establish their presence, but until now none had enough strength to impose *de facto* cloud standards.

Deficient standardization leads to incompatibilities and lock-in, which is an undesirable situation to all involved actors (with the probable exception of eventual dominant providers).

As described in section 1.2.2, the number of standardization efforts in cloud computing interoperability is significant. However, the main cloud providers seem to be able to add features and capabilities at a much faster pace than standards can settle and mature. In such a dynamic, market-oriented field, the standards that survive tend to be more *de facto*, spurred by real usefulness and market acceptance, than *de jure*, designed by working groups in standardized bodies.

Since an all-encompassing standard is not likely to appear in the near future, the current main alternative solutions are:

- Set of standards, each covering a range of aspects of cloud computing interoperability. Some overlap and even compete and not all aspects are covered. The main problem for a standard is gaining sufficient market traction, in particular taking into account that cloud providers need to maintain their differentiation and added value regarding competition. Also, in some cases standardization seems to be more SDO-driven than required by cloud providers;
- Abstraction layer over several cloud APIs [28, 30]. This corresponds to mapping a given specification (a portable API) onto the APIs of the various clouds, but it entails losing performance and dealing with compromises resulting from conflicting requirements;
- Brokerage [56, 57, 58], in which a set of services provide an API implemented by invoking the APIs from several cloud providers. This can involve adapting services, choosing the best service from a set of alternatives or adding new

services, probably by aggregating services from one or more providers. This is a flexible solution, capable of custom adaptations, but again it can involve loss of performance and compromises. The aggregated services can add value, but the fact is that brokerage leads to a new API that may simply be one more contender, increasing the problem instead of solving it.

Analysing this stack of solutions, we can identify several relevant issues:

- Standards tend to be an all or nothing solution, although in some cases the standard includes limited extension mechanisms, such as options and levels of conformance in OVF [20], the modular design of OCCI [18] and the provision for external extensions in TOSCA [21];
- Standards tend to be specification silos. Each tries to solve all the aspects of the area it tackles. Different standards usually deal with similar aspects with different underlying models, ontologies and APIs. For example, the concept of container is rather universal, but each standard deals with it differently;
- In the same track, the current approaches to cloud interoperability and its standardization start with the current layered conception of a cloud (IaaS, PaaS and SaaS) and impose rules and constraints to what we can do and cannot do, in each layer and between layers. This brings much complexity because, instead of considering a small set of foundational concepts and then deriving interoperability rules from them, the starting point is a working cloud, already a very complex system;
- Standards tend to be over-encompassing and over-specifying, instead of being modular (each dealing with a separate issue) and foundational (sticking to core specifications, not to every detail);
- There should be a backbone standard, specifying which are the areas and topics that make sense standardizing and how these can be related. Each standard, tackling one area or topic, would then fit a slot in that backbone. Unfortunately, there is no such standard, as it would require global coordination of SDOs;
- The approaches using an abstraction layer and brokerage lead to APIs that suffer from the same problems than standardized APIs, with the added problem that they have neither SDOs nor large cloud providers to back them;
- Open source software, such as OpenStack [13] is sometimes heralded as a solution to interoperability, avoiding provider lock-in. This works up to some level, since a common specification is a good starting point. However, clouds based on open specifications quickly gain extensions and new services that hinder transparent interoperability. The main advantage of open source software is lowering the entry barrier for cloud providers, which avoid building the cloud management platform from scratch, not interoperability.

We contend that a different approach is needed, so that a better response to the cloud interoperability problem can be given. This is the purpose of the next section.

1.5 A new approach to the cloud interoperability problem

1.5.1 The strategy

We need to make clear that the cloud interoperability problem is unsolvable in its entirety, given its nature, mainly due to the following reasons:

- Not all aspects are equally amenable to interoperability, because not all roles in Table 3 have the same goals;
- The market innovates faster than it organizes. This means that the pressure to provide more and better services than competitors is far greater than the need to interoperate with them. Laggards desire interoperability, but innovators and early adopters cannot (and usually do not want to) wait for specifications that support it;
- When the interoperability problem is solved at a given abstraction level (which happens when that level becomes a commodity and everybody just wants a standard), the need for differentiation by providers or the evolution in applications makes the interoperability problem move to the next upper abstraction level. An example of this is the OSI model [59], which structured interoperability at the communication level but then higher levels of interoperability started to be tackled explicitly and today, 20 years later, we are still struggling with (certainly not less) interoperability issues.

Therefore, it is not strange that the most successful specifications are low-level standards, such as OVF and CMDI (because they are near the commodity level), and open source platforms, such as OpenStack and CloudStack (because the open source cloud market appeared after, and as a result of, the development of these platforms).

The obvious starting point to cloud interoperability is the IaaS service model, and it has been questioned [23] whether it makes sense to standardize other service models, such as PaaS and SaaS, which exhibit more variability and complexity. The fact is that complex systems can be (recursively) described by the composition of simpler systems, which means that even in complex systems we can identify repeatable patterns which are amenable to standardization. We just need to discover the basic concepts that constitute the fabric of clouds.

Chemistry and Physics have taught us that much that we observe and deal with daily are just macroscopic and complex manifestations of very small and very simple elementary components while interacting with each other. Chemistry devised the Periodic table of elements, the basic properties of atoms and of other elementary particles. Physics has gone even further, by studying ever-smaller particles and other artefacts, in search of a Theory of Everything that is able to unify all the fundamental

forces of nature (such as gravity and electromagnetism) and explain all observable phenomena.

Clouds are very complex systems at a much higher level than elementary particles, but the same composition principle should apply. There must be a set of concepts and of their interactions that, by multiple and arbitrarily complex composition, should be able to fully describe the behaviour and characteristics of clouds.

Instead of considering what to do with complex cloud subsystems and how to make them interoperable, we propose to tackle cloud interoperability by adopting an approach based on the composition principle, along the following guidelines:

- To try to discover which are the fundamental and primitive artefacts and aspects that underlie the entire cloud computing domain and, if possible, the entire spectrum of computing infrastructures described in Tables 1 and 2;
- To devise an interoperability framework that shows how to make them interoperable and under which conditions;
- To model system interoperability as determined by the composition of the primitive artefacts.

Taking into account these guidelines and the problems of current approaches described in the previous section, our strategy to tackle the cloud interoperability problem can be outlined in the following way:

- To devise a generic framework, valid for any computing infrastructure in Tables 1 and 2. This can be the basis for a standard that can act as a backbone of other more specialized standards and includes:
  - A generic model of foundational artefacts, including a composition mechanism;
  - An extensible mechanism for self-description of artefacts;
  - A foundational interoperability mechanism (the basis of all interactions);
  - A framework to express the relevant aspects to interoperability between these artefacts;
- To define a core specification based on this framework, built with a set of primitive resources and their compositions;
- To define and to build clouds, and other infrastructures, by composition and extension of existing resources.

The following sections detail the interoperability framework. The core specification outgrows the context of this chapter, but a simple example of the approach is given in section 1.5.5.

1.5.2 A generic model of foundational artefacts

Our foundational artefact model, depicted in Figure 3, is very simple and includes only two main kinds of artefacts, resources and services, a composition mechanism for resources and an interaction mechanism for services. In spite of its simplicity, it can be used to build any arbitrarily complex computing infrastructure, including heterogeneous scenarios such as the one depicted in Figure 2.

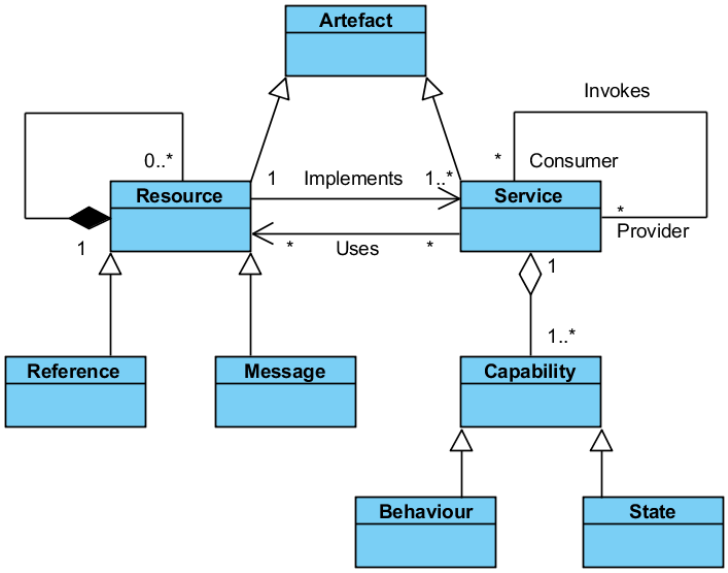

Figure 3: A foundational artefact model.

This model and can be described in the following way:

- A *service* is the set of *capabilities* (involving behaviour, state or both) that as a whole model some abstraction. Services can invoke one another, in the roles of consumer and provider. All interactions relevant to the interoperability context occur between services;
- A *resource* is the artefact providing the implementation of services and is either primitive or recursively composed of other resources. A resource implements at least one service (its management service), which exposes its inherent capabilities, but it can implement any number of other services;
- Services refer to each other by *references*, which are resources. Services interact by sending each other *messages*, which are also resources;
- One of the inherent capabilities of a resource is to incorporate and to expose some of the capabilities of a message sent to its management service. This can be

accomplished by exposing a new service or changing that resource's own capabilities (behaviour or state);

- All services expose a capability that, upon a request message from a consumer, responds with a message describing all its capabilities, thus supporting self-description.

Exposed capabilities correspond essentially to exposed operations in an API. Behaviour capabilities are operations that execute specific actions, whereas state capabilities correspond to data getter and setter operations.

A server and a storage container are examples of resources (most likely virtual, but not necessarily). The server will certainly have a management service that exposes a deployment operation, which receives a resource with the code that describes the service to be deployed and creates a new service with that code. The storage container will have a management service that includes getter and setter operations.

Figure 4 illustrates the relationship between resources and services in two different clouds. There are two resources, *A* and *B*, each with its own management service. Developers invoke operations in these services to deploy two new services, *X* and *Y* (by sending them resources with the service descriptions), which interact and need runtime interoperability. At this generic level, management services provide essentially the same functionality, which can translate to deployment-time interoperability by using compatible resource formats such as OVF.

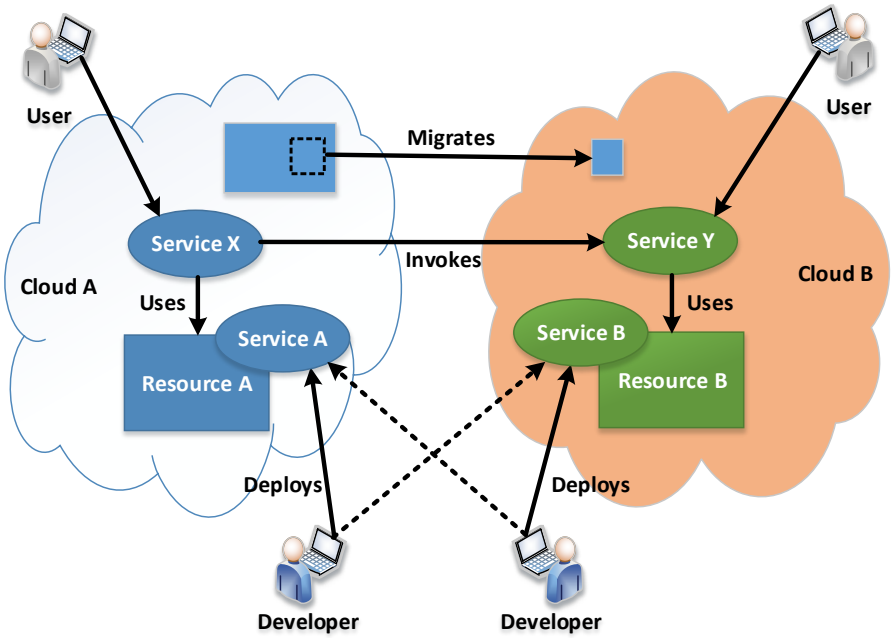

Figure 4: Relationship between services and resources.

Resource composition corresponds to clustering a set of resources and then using them as a new, composed resource, in a higher-level system. For example, a full cloud in Figure 2 is a composed resource that can be used to build a multi-cloud by composition. The concept of system composition is recursive and universal.

Virtualization now makes possible to change dynamically the structure of composed resources. Migration of a service can be accomplished by moving the resource that implements it from one place to another in the global resource tree. This is illustrated in Figure 4, in which each cloud is the top container resource for all the resources it contains and a resource has migrated from one cloud to the other.

The migration of a service running on virtual machine corresponds to stopping the service, generating a resource with its code and state, and sending that resource as a message to the management service of another server, creating a new instance of the service there and resuming execution.

In a grid, the worker resources will have operations to receive jobs, execute them and sending back the results. Migration involves data, not code, but the resource model applies exactly in the same way.

References will probably implement URIs, but not necessarily. Non-Internet networks, such as sensor networks, may use other addressing schemes and formats.

The concept of resource in this model is similar to the resource of the REST architectural style [60], but lower-level and more general. There is also a clear distinction between resources and services, with the operations offered by services not limited to a fixed set.

This model combines the structural nature of REST with the service flexibility of SOA [61] It has been designated *Structural Services* and it is described in more detail in [62].

In any case, the most important aspect of this model is that it treats all kinds of resources and of services in the same way, under a common abstraction. Specific features will be introduced by specializing capabilities in resources and services, but at least now we have a common ground on which to base interoperability.

1.5.3 A foundational interoperability mechanism

This section considers the basic interaction between two services and how interoperability can be established. This is valid not only for runtime interoperability, such as between services *X* and *Y* in Figure 4, but also at deploy-time, because deploying a service to a resource corresponds to sending a message with the description of the service to that resource's management service. Therefore, deploy-time for a service corresponds to runtime for the management service of the target resource.

Figure 5 considers the two services *X* and *Y* of Figure 4, in which *X* plays the role of consumer (sending a request) and *Y* plays the role of provider (honouring that request and sending a response).

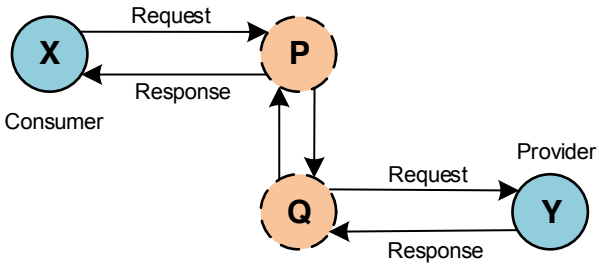

Figure 5: Compliance and conformance. *P* is how *X* views provider *Y*. *Q* is how *Y* views consumer *X*. *P* and I are specifications, not actual services.

If *X* and *Y* were designed to work together, in these roles, there would be no interoperability problem (assuming that the design was correct). Unfortunately, this is not always the case. Suppose that *X* was designed to interact with a provider *P* and *Y* was designed to expect a consumer *Q*. Having *X* interoperable with *Y* involves two conditions [63]:

- *Compliance* [64]. *P* must *comply* with *Q* in terms of requests, which means that *P* must satisfy all the requirements of *Y* to honour requests. Therefore, *X* can use *Y* as if it were *P*;
- *Conformance* [65]. *Q* must *conform* to *P* in terms of effects (including eventual responses), which means that *Q* must fulfil all the expectations of *P* regarding the effects of a request. Therefore, *Y* can replace (take the form of) *P* without *X* noticing it.

Note that, given these definitions, interoperability is inherently asymmetric in nature. Apparently symmetric interactions are the result of role reversal, in which interacting services alternate between consumer and provider roles.

Partial interoperability has been achieved by *subsumption*, with the set of capabilities that *X* uses as a subset of the set of capabilities offered by *Y* and as long as *Y* (or another service that replaces it) supports the specification *Q*.

In many cases, services are conceived and implemented to work together, i.e., made interoperable by design. When systems are complex and evolve in an independent way, ensuring interoperability is not an easy task. A typical and pragmatic solution is to resort to Web Services and XML data, sharing schemas and namespaces, or to RESTful APIs, which are simpler to use and require that schemas (media types) be standardized or pre-agreed. In these technologies, both customer and provider are forced to implement full interoperability (for example, sharing a XML schema), even if only a fraction of the possible interactions is used. This leads to a greater coupling than needed.

Other solutions involve discovering Web Services similar to what is sought, by performing schema matching with similarity algorithms [66], and ontology matching and mapping [67]. However, manual adaptations are usually unavoidable.

The notion of partial interoperability, illustrated by Figure 5, introduces a different perspective, stronger than similarity but weaker than commonality (resulting from using the same schemas and ontologies). The trick is to consider only the intersection between what the consumer needs and what the provider can offer. If the latter subsumes the former, the degree of interoperability that the consumer requires can be granted, regardless of whether the provider supports additional features or not.

1.5.4 A multidimensional interoperability framework

Compliance and conformance describe the basic interoperability mechanism in abstract terms but do not give the full picture. This section outlines a framework that provides further insight into this mechanism.

We consider three main dimensions, corresponding to fundamental perspectives of interoperability:

- *Lifecycle* of the services. Interoperability is not merely about the operation stage but starts much earlier in the system's lifecycle, in the conception stage. After all, what happens during operation is a consequence of what has been conceived and designed. In current agile and virtualized environments, development, deployment and migration are frequent operations. In the same line of thought, OCCI [18], TOSCA [21] and [8] consider explicitly several stages in the lifecycle;
- *Abstraction*. Interoperability (compliance and conformance) can be considered at several levels of abstraction, such as semantics and syntax. Considering these levels separately leads to a better structuring of the interoperability aspects;
- *Concerns*. Interoperability is not limited to functional objectives (the intended effect by invoking another service). There is a range of other concerns that need to be considered, mostly non-functional, such as configuration, SLA, reliability, security, policy management, legislation and regulatory compliance, financial aspects, and so on. If, for example, the provider cannot meet (cannot conform to) the service level requirements of the consumer, interoperability will not be properly achieved.

For simplicity, we will not consider here a fourth dimension, *evolution*, which corresponds to successive versions of the interacting services obtained by successive iterations of their lifecycles.

Figure 6 illustrates a typical lifecycle. Different software engineering methods may use different lifecycles, but the general idea is that it flows along phases such as conception, implementation, deployment, production (also known as operation, or execution) and finally transition. During conception, a strategy assessment determines if the lifecycle proceeds to subsequent phases or the service is not

worthwhile and is terminated. During operation, monitoring may determine reconfiguration or migration (for load balancing, for instance) and evaluation of management metrics (KPIs, Key Performance Indicators) may determine changes to the service, with deployment of a new version of the service (and the current one is finalized).

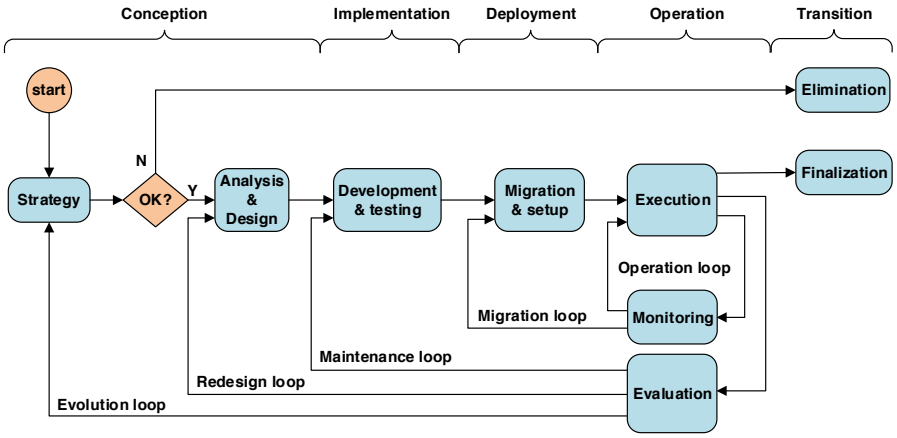

Figure 6: A typical service lifecycle, including redeployments as migrations.

The lifecycle is very important for interoperability, because not only interaction with other services must be considered by design (not as an afterthought) but also it constitutes a way of managing what happens to the service, including migration, in a controlled way. This is even more important in dynamic provisioning environments, such as clouds.

When a service sends a message to another, there is a full spectrum of aspects that both sender and receiver must understand, from low-level protocols up to the intentions that motivated the interaction. These aspects can be organized into levels of interoperability abstraction. We use a scale of five levels, which can be further refined:

- *Symbiotic*, reflecting the *intent* (motivation and goals) of a service when engaging in an interaction with another. This can be tackled at levels of governance, alignment, collaboration or mere outsourcing;
- *Pragmatic*, dealing with the *effect* of the interaction on the other service. This can be specified at the choreography, process and service levels;
- *Semantic*, which expresses the *meaning* of the messages exchanged and of the resulting behaviour at the levels of rules, knowledge and ontology;
- *Syntactic*, which deals with the *format* of the messages, in terms of schema, primitive resources and their serialization;
- *Connective*, establishing the *protocol* at the message, routing, communication and physical levels.

Interoperability is possible only if all these levels contribute to it. For example, if the intended effect is expressed correctly at the upper levels but one service sends a message in XML and the other is expecting JSON, they will not be able to interact. The same happens if services use exactly the same technologies and tools, but one expects an effect (such as receiving a payment) that the other does not provide.

In most practical cases, only a few levels are dealt with explicitly. The most common are syntactic and pragmatic, with semantic gaining ground. The others can be tackled:

- *Tacitly*, by assuming that what is missing has somehow been previously agreed, is obvious or described in the documentation that the developer will read. Inferring intent and semantics from documentation or undocumented behaviour constitute examples of this;
- *Empirically*, by resorting to some tool or technology that deals with what is necessary to make interoperability work. An example is using Web Services without caring about the details of how they work.

The correlation between the lifecycle and interoperability dimensions of the framework is illustrated by Figure 7 (in which the lifecycle has been simplified) and can be described in the following way:

- All the levels of abstraction of interoperability must be considered in every stage of the lifecycle. Intentions behind interactions are still present during execution, although in a tacit way, and connectivity is already needed in the conception stage, although in an empiric way. However, it is only natural that the method used to evolve the service along its lifecycle considers interoperability at an abstract level in the initial stages and at full detail in the operation stage. The progression illustrated by Figure 7 is just an example, since it depends on the method used;
- There must be compliance (in the consumer to provider direction) and conformance (in the provider to consumer direction) in each corresponding cell (in a given stage and abstraction level) in both interacting services. Figure 7 illustrates this in detail in the operation stage only. However, the same should happen in other stages (the dashed arrows at the top of the figure indicate this). The rationale for this is twofold:
  - Services must be interoperable at each abstraction level of interoperability, be it at the intention level (one must understand and accept the intentions of the other), at the meaning level (interoperable ontologies and semantic relationships) or at the basic protocol level (there must be message connectivity);

  - Each stage in the lifecycle is a consequence of the previous stage, in a model-driven fashion. For systems to be interoperable at operation time, their conception and design must also be made interoperable;

- Evaluation and monitoring need not be made interoperable. Each service can evaluate its design or operation independently, although the specifications for interoperability must not be overlooked and need to be considered in the context of the partners' relationship. This is the case of SLA, for example, but these types of aspects are dealt with by the third dimension of the framework (concerns).

Figure 8 depicts the framework in its three dimensions. The plane (two dimensions) of Figure 7 is repeated for each of the concerns relevant to interoperability. Besides the functional interoperability (resulting from the specification of the services), we need also to consider non-functional aspects such as security, SLA and financial aspects.

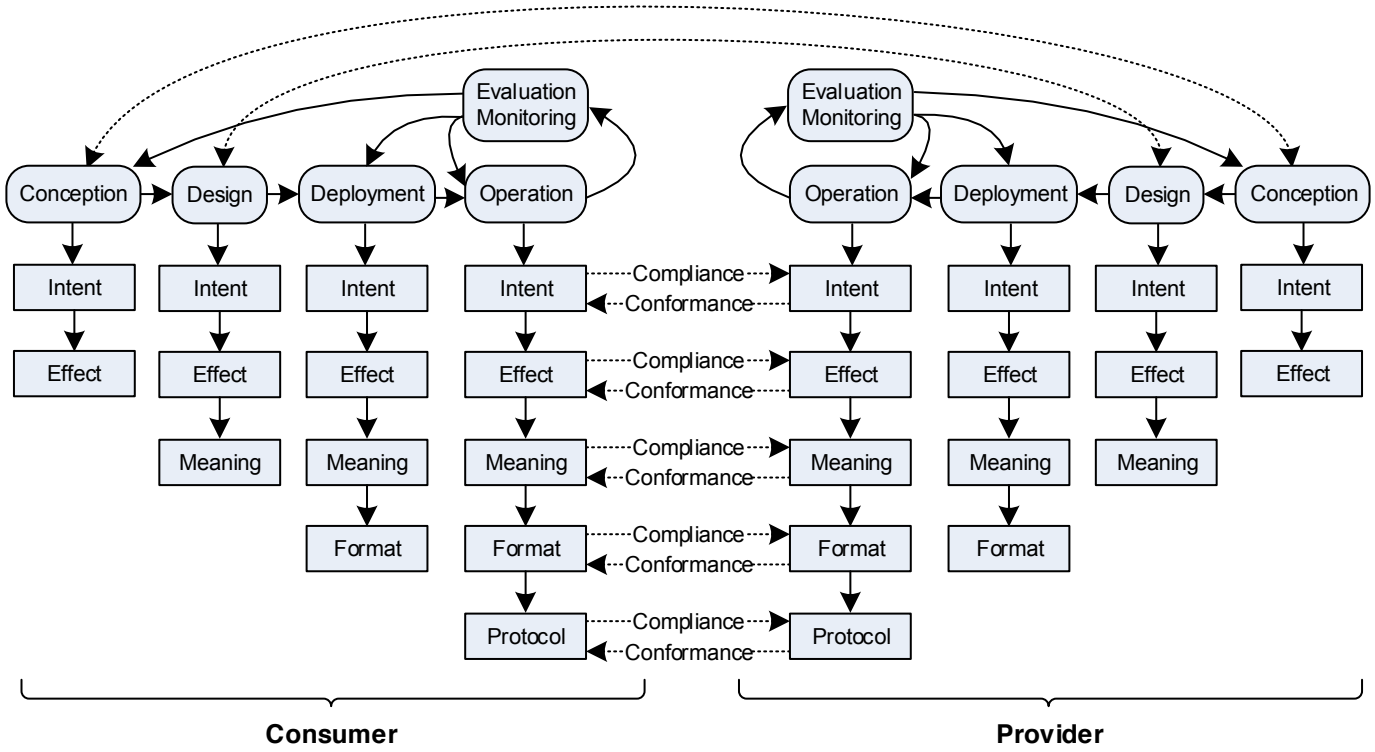

Figure 7: Service lifecycle combined with interoperability abstraction levels.

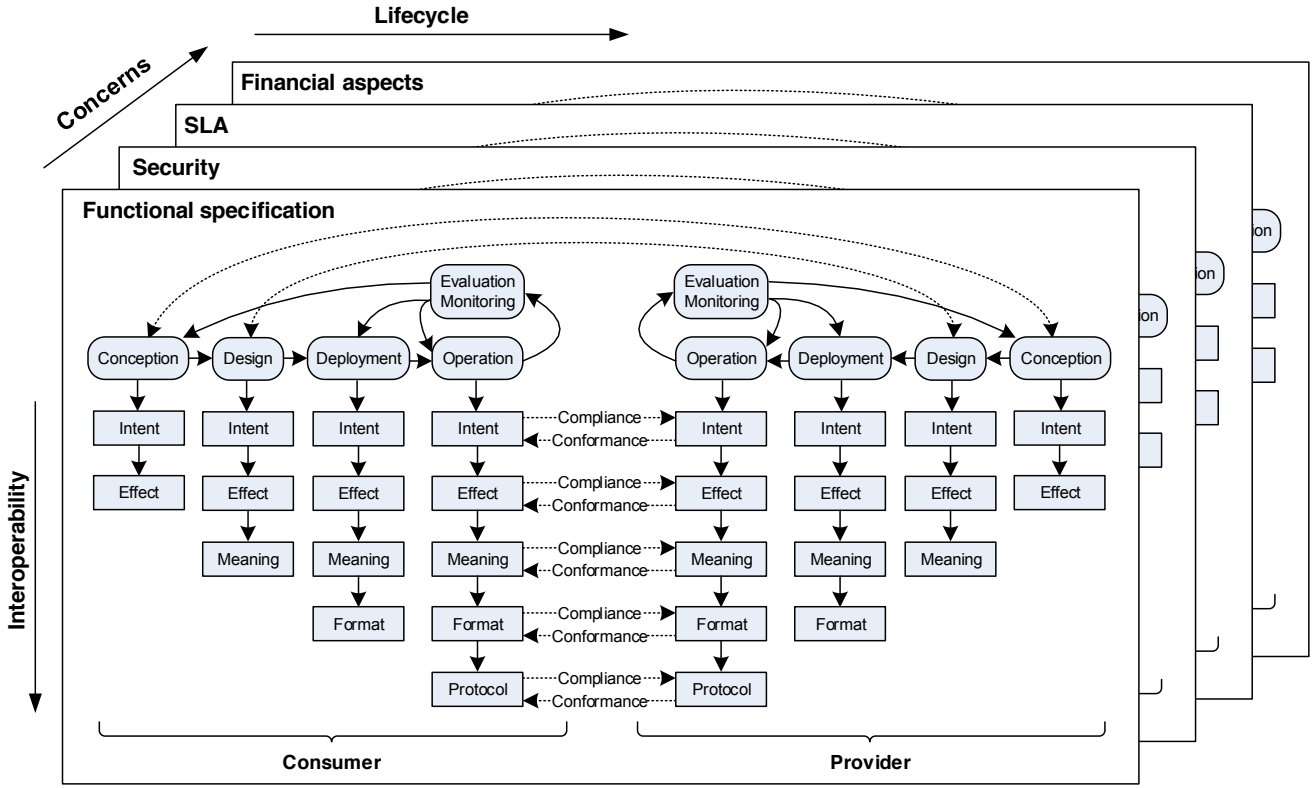

Figure 8: The three dimensions of the interoperability framework.

Compliance and conformance still apply in the non-functional realm, e.g.:

- The consumer must comply with the provider's security policies when sending a request. The provider must conform to the consumer's security policies when sending a response;
- The consumer must comply with the SLA metrics and policies that the provider is able to provide and accept. In turn, the provider must conform to the requirements (such as performance and reliability) sought by the consumer. If this is not accomplished, interoperability can be achieved in functional terms, but not entirely because non-functional requirements have not been met;
- The same can be said about financial aspects, in which cost and payment terms need to be agreed under basically the same principles as an SLA.

1.5.5 A simple example

This section provides a very simple example of how partial compliance and conformance can be used in a structural way and how it can promote API interoperability. We will illustrate data compliance and conformance only, for simplicity, but it should give an idea of what is involved. We use JSON to illustrate the details because it is easier to read than XML.

Suppose that we want to use a cloud to create a virtual server. First, we need to know which types of servers the cloud is able to create. To support self-description, the cloud's API should have an operation to return that information. In the light of the model described in section 1.5.2, that operation could return a JSON structure such as the one in Listing 1.

```
{
   "cpu" : {
      "architecture" : ["x86", "x64", "powerPC"],
      "cores" : [1, 2, 4, 8],
      "performance" : ["extra", "high", "medium"],
      "ram" : [4, 8, 16, 32, 64],
      "reliability" : ["extra", "high"]
   },
   storage : {
      "size" : { "min" : 0.1, "max" : 1000 },
      "performance" : ["high", "medium"],
   },
   "zone" : ["Europe", "US", "Asia"]
}
```

Listing 1: JSON example describing the characteristics of a server.

The JSON array elements express alternatives that the server can support. Object members correspond to characteristics of the resource (cpu or storage). Note the structural organization of the server's characteristics. They are not just a linear list.

Listing 1 acts as a schema for the servers in this cloud, including all the characteristics and possible values and alternatives. Servers are composed of a cpu, a storage and are located in one of three alternative zones. The cpu and storage also have their characteristics (self-explanatory, at the level of this example). Ram and storage sizes are expressed in gigabytes.

The cloud will have several server types available, each with a subset of the possible combinations of characteristics. The cloud should also have an operation to retrieve a list of available server types. For example, that list could include the two following server types described in Listing 2.

```
{
   server_type_1 : {
      "cpu" : {
         "architecture" : ["x86", "x64", "powerPC"],
         "cores" : [2, 4, 8],
         "performance" : ["extra", "high", "medium"],
         "ram" : [8, 16, 32],
         "reliability" : ["extra"]
      },
      storage : {
         "size" : { "min" : 1, "max" : 50 },
         "performance" : ["medium"],
      },
      "zone" : ["US", "Asia"]
   },
   server_type_2 : {
      "cpu" : {
         "architecture" : ["x86", "x64"],
         "cores" : [2, 4],
         "performance" : ["high"],
         "ram" : [4, 8, 16],
         "reliability" : ["high"]
      },
      storage : {
         "size" : { "min" : 0.1, "max" : 20 },
         "performance" : ["high", "medium"],
      },
      "zone" : ["Europe"]
   }
}
```

Listing 2: JSON example describing the characteristics of types of servers offered by a cloud.

Now, imagine that we wanted to create a server with the following characteristics (the array elements express the alternatives we are willing to accept):

```
{
   "cpu" : {
      "architecture" : ["x86", "x64"],
      "cores" : [4, 8],
      "performance" : ["medium"],
      "ram" : [16, 32],
   },
   storage : {
      "size" : { "min" : 1, "max" : 2 },
      "performance" : ["high"],
   },
   "zone" : ["Europe", "US"]
}
```

Listing 3: JSON example describing the server characteristics required by a customer.

Although we haven't discussed the algorithms to test compliance and conformance, an informal comparison between these JSON listings illustrates the following comments:

- `server_type_1` cannot be used because it does not conform to the server required by the customer, in Listing 3. Checking the various characteristics, there is a matching solution for all except storage performance. The customer requires "high" but the server only offers "medium". If it were the other way around, conformance would be verified;
- `server_type_2` conforms to Listing 3. For all characteristics, there is a matching solution. Note that the customer requires a "medium" performance cpu but the provider only offers "high". From the point of view of performance this is valid, but it may hinder conformance when cost is introduced (it may be higher than acceptable, since the cpu is better than required). All characteristics need to be taken into account when checking interoperability;
- The customer specified nothing regarding "reliability", in Listing 3. This means that anything is accepted in this characteristic and that it is not taken into account when checking conformance;
- Compliance is also needed when the customer requests the creation of a server. The operation to inquiry the cloud about the server types it supports exists to help in the design, but there must be another operation to make the actual request for the creation of the server instance. The type of the actual argument to pass to that operation is Listing 3 (what the customer requires) and the type of the formal argument is Listing 1 (what the cloud can provide). When operations are invoked, the actual arguments must comply with the formal arguments. In this case, Listing

3 must comply with Listing 1 (which we can verify by checking that there is a matching solution for all characteristics in Listing 3). After checking argument compliance, the server creation operation uses Listing 2 to search for a suitable server type, which it then creates;

- The names and structure of the elements in Listing 1 must be in accordance with the ontology used by the cloud. The customer needs not use exactly the same ontology in Listing 3. Another one that complies with it is enough. Compliance and conformance, as asserted in section 1.5.4, must hold at all interoperability levels, namely semantics, ontology [68] included.

1.6 Discussion and rationale

1.6.1 Usefulness of the approach

How can this framework be useful in the current interoperability panorama? The main purpose of a framework is to aid in systematizing human thought, so that the problem can be better structured and a solution is easier to find. The structure of this chapter follows this approach, with a first structuring of the problem, analysis of current solutions, a framework and the outline of a solution.

Even at the standardization level, we believe that the interoperability problem (in any computing infrastructure and in clouds in particular), should use the same approach, which can be outlined in the following way:

- To define an interoperability framework, based on a generic artefact model and on orthogonal dimensions that cater for the lifecycle of artefacts, their interoperability at various abstraction levels and non-functional concerns (security, SLA, etc.);
- To define a set of primitive types of artefacts for each type of computing infrastructure, with a self-describing interface. This set may be organized into categories of artefacts;
- To define standard interfaces for the most used artefacts, in such a way that a basic computing infrastructure (such as a cloud or a grid) could be built in a standardized fashion, by composing the adequate types of artefacts;
- To define a backbone standard, corresponding to a minimalist computing infrastructure (with basic resources and services), which further standards could be laid upon;
- To define provider-specific artefacts (non-standardized, thus providing explicitly a mechanism for provider differentiation), either anew or by extension (based on compliance and conformance) of primitive or standardized artefacts;

- To build provider-specific computing infrastructures (namely, clouds), by using both standardized and provider-specific artefacts;
- To build an API centred on resources. This means that, instead of having a global API, each resource should have its own API. The advantage of this is that there are many operations common to different types of resources. Thanks to compliance and conformance, which behave as a distributed inheritance mechanism, common operations can be dealt with in the same way, as if we had distributed object-oriented polymorphism.

This is an incremental approach, which tries to standardize the minimum possible but with a structure that is designed to grow, instead of solving a reasonably sized chunk of the interoperability problem, which is the approach followed by most of the current standardization efforts.

The partial interoperability granted by compliance and conformance provides a better decoupling between services than a full standard specification that all services must use or implement. Better decoupling increases adaptability (fewer dependencies), changeability (it is easier to change a service if others depend on less of its features) and reliability (it becomes easier to find a replacement for a service that failed) [63].

1.6.2 Comparison with other approaches

Approaches such as OCCI [18] and TOSCA [21] are more modular than others, but still try to solve most of the problems in the areas they tackle.

In [29], the abstraction of cloud resources is presented in an object-oriented way, but without a framework that guides the interoperability aspects.

In [8], guidelines and requirements for the functionality to include in a multi-cloud API are presented, along the service lifecycle, but the ideas to implement them start at existing technologies, platforms and tools. Although an evolutionary route, easier to implement, it does not solve the problem of the standardization of the underlying model.

The intercloud [40, 41, 42] is an attempt to integrate multiple clouds, in a parallel with the integration between servers provided by the Internet and the Web in particular. The problem is that in cloud computing the market expanded rapidly before the standards and the situation is quite different. The intercloud will only be as good as the interoperability between multiple clouds. Again, this is a top-level approach to interoperability, trying to integrate existing clouds without solving the lower level interoperability problems first.

The approach that consists in defining a common API, abstracting the actual APIs of cloud providers [28, 29, 30] is not without problems. It needs to abstract the different artefacts used by cloud providers (which means compromises that enable

support only for the most used API patterns) and to deal with constant evolutions in the APIs, since the field is not mature yet.

One recent trend, seen in OCCI [18], TOSCA [21] and in the Meta Cloud [69] is the use of structural resource templates to describe the structure of a service, or the resources it requires, so that provisioning is simpler and more automated. This is in line with our artefact model (Figure 3), which adds the benefit of compliance and conformance, as exemplified in section 1.5.5.

It seems that everybody agrees that a single standard, encompassing all the aspects of cloud computing, namely interoperability, is not a viable solution. The main difference between the existing approaches is the subset of features to include, more vertically (a small number of aspects, but crossing the whole stack of cloud service models) or more horizontally (encompassing only one cloud service model) orientated.

Up to now, the most successful approach has been horizontal and low level, with the OVF and CMDI standards. The other efforts tend to go to upper levels and to build on what exists. Our proposal is different, starting by agreeing on a common artefact model, simple enough to be general but with extension mechanisms that enable each provider to build its own cloud model and platform. This approach starts at the simplest and lowest layer, providing a cleaner foundation for the computing infrastructure domain, not just for clouds.

1.6.3 Risks and limitations

Freed from the compatibility burden, our approach lays down a clean model, based on active and interacting resources, with their services and solutions conceived specifically for them, instead of using established technologies designed for hypermedia document exchange and that simulate services on top of a document-based abstraction.

New approaches, such as ours, have the advantage of exploring new routes, which is an indispensable step for innovation, but are usually not exempt from their own limitations and incur the risk of never generating enough traction to be adopted by the market, thereby never progressing beyond the stage of an interesting solution.

In our case, the following main risks and limitations are easily perceivable:

- Essentially, this is an untried approach. There is a prototype implementation, but it does not implement all the features and certainly has not been quantitatively assessed at a reasonably large scale, with a real comparison with current approaches, namely SOA and REST. Therefore, the claims made still need practical and meaningful validation;
- The approach is deeper than the mere domain of cloud computing. It goes back to the Web itself, since it is based on the service paradigm and not on the classical client-server hypermedia document exchange model. Although a potential better

match to modern application requirements, this is a huge paradigm shift, with many possible problems in the way, yet to uncover;
- Any approach needs tools and middleware to support it, which requires market interest and takes time to develop, leading to a bootstrap problem. This has yet to be tackled, with a migration path that allows coexistence with current approaches;
- Compliance and conformance, in particular, constitute a rather different solution from the schema sharing approach currently used. It raises several problems, from formal treatment to performance, which have not been properly addressed yet in realistic scenarios;
- Although the polymorphism granted by compliance and conformance has the potential to support a common core of cloud service specifications, it has not been demonstrated yet how this can actually be done in a practical way, to really support useful standardization while allowing freedom for provider innovation.

1.7 Future research directions

From the point of view of the users, the ideal would be to treat providers in a cloud fashion, i.e., using services or installing their applications without really caring about which cloud provider they are using. This means virtualizing the cloud providers themselves, not just the infrastructure or the platform.

We believe that the proposal presented in this chapter is a step in that direction, but a lot needs to be done before this vision can turn into reality, namely:

- There is currently no language that implements structural compliance and conformance as described in section 1.5.3. We have presented a simple example in section 1.5.5 using JSON as notation, but this is much more limited than what we can do with a language that supports better data structures and operations, not just data, and combines the best of SOA (flexible services) and REST (flexible resources). We are developing a compiler and platform for such a language (SIL – Service Implementation Language) [70];
- The semantic level, namely compliance and conformance at the ontology and knowledge level, needs further study. At the moment, most of the research made on interoperability at the semantic level concerns similarity and matching [66, 67], not compliance and performance;
- The interoperability framework that we have presented has not been tested yet. We are still building the prototype of the SIL environment, but the interoperability problem in the domain of cloud computing is a very good match to the capabilities of the language. We will use it as one of the first practical examples. The goal is to build a very basic cloud platform, with a small number of universally used resource types and operations, and then derive two example clouds with different APIs, with additional resources and functionality. SIL itself

serves to specify the structure of the resources (as shown in section 1.5.5) and services of a compose application, allowing to test service and resource templates;

- One of the basic features of SIL is that it is compiled and serialized in binary. It has two representations, one with a look and feel similar to JSON for programmers and a binary format for computer processing. The compiler maintains the synchronism between the two. Native support for binary means that service images to deploy to resources no longer need to have a separate manifest or sent in a zip file. The same binary format used to serialize the resources (in a TLV approach – Tag followed by length and value) can be used to include these images. This allows us to study an alternative to OVF with the goal of providing a native implementation of the model depicted in Figure 3.

1.8 Conclusions

The seamless interoperability that we enjoy today both at the Internet and Web levels cannot be achieved in the near future in the field of cloud computing. In the former cases, technology and standards had time to mature before the market expanded. In the latter, it happened the other way around. Under user pressure and by initiative of some early-adopter providers (in cloud storage), the market expanded and is evolving very rapidly, before standards had time to impose some order.

Today, large providers are not particularly interested in standardization. It seems that the value of compliance with standards (perspective of the users) is greater than the value of conformance to those standards (perspective of the providers). In other words, the absence of widely accepted standards endows large providers with the power of the lock-in law.

Only the lowest-level standards, such as OVF and CMDI, have enjoyed significant success. The reasons for this are simple. These are the oldest standards and, given their low level, do not contribute significantly to the differentiation that providers need to attract a larger customer base.

Nevertheless, users continue with an interoperability problem, trying to defend themselves against provider lock-in. Current attempts to solve or reduce this problem use essentially high-level solutions, building common APIs over the existing providers and cloud computing concepts, or exposing brokers that hide the differences between providers.

In this chapter, we have presented a different approach, starting from a foundation artefact model, based on resources and services, adding a basic interoperability mechanism, based on structural compliance and conformance, and foreseeing extension mechanisms upon which providers can build their differentiating services. We proposed the idea of a core standard with the foundation concepts, which can serve as a backbone for higher-level standards.

In a sense, this approach tackles the interoperability problem in the incremental and generic way in which it should have evolved. However, it does not constitute an evolutionary route, and is therefore most likely longer and harder to implement than

other approaches. Nevertheless, it has the merit of providing potential rewards in terms of accomplishing a cleaner structure and organization than what exists today in the cloud computing landscape.

1.9 References

1. Armbrust M *et al*., (2010), A View of Cloud Computing, *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, April 2010
2. Carr N, (2004), Does IT matter?: information technology and the corrosion of competitive advantage, *Harvard Business Press*, 2004
3. Brynjolfsson E, Hofmann P and Jordan J, (2010), Cloud computing and electricity: beyond the utility model, *Communications of the ACM*, vol. 53, no. 5, pp. 32-34, May 2010.
4. Morgan T, (2013), *Amazon slices prices on dedicated EC2 private cloud puffs*, The Register, 10 July 2013, http://www.theregister.co.uk/2013/07/10/amazon_slashes_prices_on_dedicated_ec2_server_slices/, accessed 16 July 2013
5. Jack Clark, (2013), *Salesforce and Oracle forge partnership to smash rivals*, The Register, 25 June 2013, http://www.theregister.co.uk/2013/06/25/salesforce_oracle_partner_analysis/, accessed 16 July 2013
6. Anderson E *et al*., (2013), *Forecast Overview: Public Cloud Services, Worldwide, 1Q13 Update*, Gartner report, http://www.gartner.com/resId=2473016, accessed 16 July 2013.
7. Lewis G, Morris E, Simanta S and Wrage L, (2008), Why standards are not enough to guarantee end-to-end interoperability, *Proc. Seventh International Conference on Composition-Based Software Systems*, pp. 164-173, February 2008
8. Petcu D, (2013), Multi-Cloud: expectations and current approaches, *Proc.of the International Workshop on Multi-cloud Applications and Federated Clouds*, pp. 1-6, New York, NY, April 2013
9. Mell P and Grance T, (2011), *The NIST definition of cloud computing*, Special publication 800-145, National Institute of Standards and Technology, Sept. 2011, http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, accessed 16 July 2013
10. Rimal B, Choi E and Lumb I, (2009), A taxonomy and survey of cloud computing systems, *Proc. Fifth International Joint Conference on INC, IMS and IDC*, pp. 44-51, August 2009
11. Zhang Q, Cheng L and Boutaba R, (2010), Cloud computing: state-of-the-art and research challenges, *Journal of Internet Services and Applications*, vol. 1, no. 1, 7-18, 2010
12. Bist M, Wariya M and Agarwal A, (2013), Comparing delta, open stack and Xen Cloud Platforms: A survey on open source IaaS, *Proc. IEEE 3rd International Advance Computing Conference*, pp. 96-100, 2013
13. Jackson K, (2012), *OpenStack Cloud Computing Cookbook*, Packt Publishing, 2012
14. Sabharwal N and Shankar R, (2013), *Apache Cloudstack Cloud Computing*, Packt Publishing Ltd, 2013
15. Nurmi D *et al*., (2009), The eucalyptus open-source cloud-computing system, *Proc. 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 124-131, May 2009
16. Gallagher S and Dalgleish A, (2013), *VMware Private Cloud Computing with vCloud Director*, John Wiley & Sons, 2013
17. DTMF, (2012), *Cloud Infrastructure Management Interface (CIMI) Model and REST Interface over HTTP Specification*, Document Number: DSP0263, version 1.0.1, 12 Sept. 2012, http://www.dmtf.org/sites/default/files/standards/documents/DSP0243_1.0.1.pdf, accessed 16 July 2013

18. Edmonds A, Metsch T and Papaspyrou A, (2011), Open Cloud Computing Interface in Data Management-Related Setups. In *Grid and Cloud Database Management,* pp. 23-48, Springer Berlin Heidelberg, 2011
19. SNIA, (2012), *Cloud Data Management Interface (CDMI) v1.0.2*, 18 March 2013, http://docs.oasis-open.org/tosca/TOSCA/v1.0/cs01/TOSCA-v1.0-cs01.pdf, accessed 16 July 2013
20. DTMF, (2012), *Open Virtualization Format Specification*, Document Number: DSP0263, version 2.0.0, 13 Dec. 2012, http://www.dmtf.org/sites/default/files/standards/documents/DSP0243_2.0.0.pdf, accessed 16 July 2013
21. Binz T, Breiter G, Leyman F and Spatzier T, (2012), Portable cloud services using Tosca, *IEEE Internet Computing*, vol. 16, no. 3, pp. 80-85, May-June 2012
22. OASIS, (2013), *Topology and Orchestration Specification for Cloud Applications - Version 1.0*, 4 June 2012, available at http://www.snia.org/cdmi, accessed 16 July 2013
23. Lewis G, (2012), *The Role of Standards in Cloud-Computing Interoperability*, Software Engineering Institute, Paper 682, Oct. 2012, http://repository.cmu.edu/sei/682, accessed 16 July 2013
24. Loutas N, Kamateri E, Bosi F and Tarabanis K, (2011), Cloud computing interoperability: the state of play, *Proc. IEEE Third International Conference on Cloud Computing Technology and Science*, pp. 752-757, Nov. 2011
25. Zhang Z, Wu C and Cheung D, (2013) A survey on cloud interoperability: taxonomies, standards, and practice, *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, no. 4, pp. 13-22, April 2013
26. Kostoska M, Gusev M, Ristov S and Kiroski K, (2012), Cloud Computing Interoperability Approaches-Possibilities and Challenges, *Proc. Fifth Balkan Conference in Informatics*, pp. 30-34, Serbia, Sept. 16–20, 2012
27. Petcu D, Macariu G, Panica S and Crăciun C, (2013), Portable cloud applications—from theory to practice, *Future Generation Computer Systems*, Vol. 29, no. 6, pp. 1417–1430, Aug. 2013
28. Petcu D, Craciun C and Rak M, (2011), Towards a cross platform Cloud API - Components for Cloud Federation, *Proc. 1st International Conference on Cloud Computing and Services Science*, pp. 166-169,The Nederlands, May 2011
29. Nguyen B, Tran V and Hluchý L, (2012), Abstraction layer for development and deployment of cloud services, *Computer Science*, vol. 13, no. 3, pp. 79-88, 2012
30. Hill Z and Humphrey M, (2010), CSAL: A cloud storage abstraction layer to enable portable cloud applications, *Proc. IEEE Second International Conference on Cloud Computing Technology and Science*, pp. 504-511, Nov. 2010
31. Mındrut C and Fortis T, (2013), A Semantic Registry for Cloud Services, *Proc. 27th International Conference on Advanced Information Networking and Applications Workshops*, pp. 1247-1252, Spain, March 2013
32. Loutas N, Kamateri E and Tarabanis K, (2011), A semantic interoperability framework for cloud platform as a service, *Proc. IEEE Third International Conference on Cloud Computing Technology and Science*, pp. 280-287, Nov. 2011
33. Di Modica G, Petralia G and Tomarchio O, (2012), A semantic framework to support cloud markets in interoperable scenarios, *Proc. of the IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, pp. 211-214, Nov. 2012
34. Bernstein D and Vij D, (2010), Using Semantic Web Ontology for Intercloud Directories and Exchanges, *Proc. International Conference on Internet Computing*, pp. 18-24, Las Vegas, NV, July 2010
35. Androcec D, Vrcek N and Seva J, (2012), Cloud Computing Ontologies: A Systematic Review, *Proc. Third International Conference on Models and Ontology-based Design of Protocols, Architectures and Services*, pp. 9-14, April 2012
36. Jardim-Goncalves R, Cretan A, Coutinho C, Dutra M and Ghodous P, (2013), Ontology Enriched Framework for Cloud-based Enterprise Interoperability, In *Concurrent*

Engineering Approaches for Sustainable Product Development in a Multi-Disciplinary Environment, pp. 1155-1166, Springer London, 2013

37. Ardagna D *et al*., (2012), MODAClouds: A model-driven approach for the design and execution of applications on multiple Clouds, *Proc. Workshop on Modeling in Software Engineering*, pp.50-56, June 2012
38. Paraiso F, Merle P and Seinturier L, (2013), Managing elasticity across multiple cloud providers, *Proc. of the International Workshop on Multi-cloud Applications and Federated clouds*, pp. 53-60, New York, NY, April 2013
39. Grozev N and Buyya R, (2012), Inter-Cloud architectures and application brokering: taxonomy and survey, *Software: Practice and Experience*, DOI: 10.1002/spe.2168
40. Demchenko Y, Makkes M, Strijkers R and de Laat C, (2012), Intercloud Architecture for interoperability and integration, *Proc. 4th IEEE International Conference on Cloud Computing Technology and Science*, pp.666-674, Taiwan, Dec. 2012
41. Bernstein D, Ludvigson E, Sankar K, Diamond S and Morrow M, (2009), Blueprint for the intercloud-protocols and formats for cloud computing interoperability, *Proc. Fourth International Conference on Internet and Web Applications and Services*, pp. 328-336, Venice, Italy, May 2009
42. Bernstein D and Vij D, (2010), Intercloud directory and exchange protocol detail using XMPP and RDF, *Proc 6th World Congress on Services*, pp. 431-438, July 2010
43. Jardim-Goncalves R, Grilo A, Agostinho C, Lampathaki F and Charalabidis Y, (2013), Systematisation of interoperability body of knowledge: the foundation for enterprise interoperability as a science. *Enterprise Information Systems*, vol. 7, no. 1, pp. 7-32, 2013
44. Ostadzadeh S and Fereidoon S, (2011), An Architectural Framework for the Improvement of the Ultra-Large-Scale Systems Interoperability, *Proc. International Conference on Software Engineering Research and Practice*, Las Vegas, July 2011
45. Guédria W, Gaaloul K, Proper H and Naudet Y, (2013), Research Methodology for Enterprise Interoperability Architecture Approach, *Proc. Advanced Information Systems Engineering Workshops*, pp. 16-29, Springer Berlin Heidelberg, 2013
46. Ahronovitz M *et al*., (2010), *Cloud Computing Use Cases*, version 4.0, white paper from the Cloud Computing Use Case Discussion Group, http://opencloudmanifesto.org/Cloud_Computing_Use_Cases_Whitepaper-4_0.pdf, accessed 16 July 2013
47. Petcu D, (2011), Portability and interoperability between clouds: challenges and case study, *Proc. 4th European Conference Towards a Service-Based Internet*, pp. 62-74, Poland, Oct. 2011
48. Fernando N, Loke S and Rahayu W, (2013), Mobile cloud computing: A survey, *Future Generation Computer Systems*, vol. 29, no. 1, pp. 84-106, Jan. 2013
49. Keyes J, (2013), *Bring Your Own Devices (BYOD) Survival Guide*, CRC Press, 2013
50. Gubbi J, Buyya R, Marusic S and Palaniswami M, (2013), Internet of Things (IoT): A vision, architectural elements, and future directions, *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, Sept. 2013
51. Aggarwal C and Han J, (2013), A Survey of RFID Data Processing, In *Managing and Mining Sensor Data*, pp. 349-382, Springer US, 2013
52. Potdar V, Sharif A and Chang E, (2009), Wireless sensor networks: A survey, *Proc. International Conference on Advanced Information Networking and Applications Workshops*, pp. 636-641, Bradford, UK, IEEE, May 2009
53. Liu Y, Rong Z, Jun C and Ping C, (2011), Survey of Grid and Grid Computing. *Proc. International Conference on Internet Technology and Applications*, pp. 1-4, Wuhan, China, IEEE, Aug. 2011
54. Hughes D, Coulson G and Walkerdine J, (2010), A Survey of Peer-to-Peer Architectures for Service Oriented Computing, In *Handbook of Research on P2P and Grid Systems for Service-Oriented Computing: Models, Methodologies and Applications*, pp. 1-19, IGI Global, 2010

55. Seinstra F *et al*., (2011), Jungle computing: Distributed supercomputing beyond clusters, grids, and clouds, In *Grids, Clouds and Virtualization*, pp. 167-197, Springer London, 2011
56. Buyya R, Ranjan R and Calheiros R, (2010), Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services, *Proc. 10th International Conference on Algorithms and architectures for parallel processing*, pp. 13-31, Busan, Korea, May 2010.
57. Sundareswaran S, Squicciarini A and Lin D, (2012), A brokerage-based approach for cloud service selection, *Proc. IEEE 5th International Conference on Cloud Computing*, pp. 558-565, Honolulu, HI, June 2012
58. Nair S *et al*., (2010), Towards secure cloud bursting, brokerage and aggregation, *Proc. IEEE 8th European Conference on Web Services*, pp. 189-196, December 2010
59. ISO/IEC, (1996), *ISO/IEC 7498-1, Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*, 2nd edition corrected, 15 June 1996, International Standards Office, Geneva, Switzerland, http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html, accessed 16 July 2013
60. Earl T, (2012), *SOA with REST: Principles, Patterns & Constraints for Building Enterprise Solutions with REST*, Prentice Hall PTR, 2012
61. Earl T, (2007), *SOA: Principles of Service Design*, Prentice Hall PTR, 2007
62. Delgado J, (2012), Bridging the SOA and REST architectural styles, In *Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments*, pp. 276-302, IGI Global, 2012
63. Delgado J, (2012), Structural interoperability as a basis for service adaptability, In *Adaptive Web Services for Modular and Reusable Software Development: Tactics and Solutions*, pp. 33-59, IGI Global, 2012
64. Kokash N and Arbab F, (2009), Formal Behavioral Modeling and Compliance Analysis for Service-Oriented Systems, In *Formal Methods for Components and Objects*, pp. 21-41, Springer-Verlag, Berlin, Heidelberg, 2009
65. Adriansyah A, van Dongen B and van der Aalst W, (2010), Towards robust conformance checking, In *Business Process Management Workshops*, pp. 122-133, Springer Berlin Heidelberg, 2010
66. Jeong B, Lee D, Cho H and Lee J, (2008), A novel method for measuring semantic similarity for XML schema matching, *Expert Systems with Applications*, vol. 34, pp. 1651–1658, 2008
67. Euzenat J and Shvaiko P, (2007), *Ontology matching*, Springer Berlin, 2007
68. Mizoguchi R and Kozaki K, (2009), Ontology engineering environments, In *Handbook on Ontologies*, pp. 315-336, Springer Berlin Heidelberg, 2009
69. Satzger B et al., (2013), Winds of Change: From Vendor Lock-In to the Meta Cloud, *IEEE Internet Computing*, vol. 17, no. 1, pp. 69-73
70. Delgado J, (2013), Service Interoperability in the Internet of Things, In *Internet of Things and Inter-cooperative Computational Technologies for Collective Intelligence*, pp. 51-87, Springer Berlin Heidelberg, 2013

Index