Using Cloud Standards for Interoperability of Cloud Frameworks

Thijs Metsch[1] and Andy Edmonds[2] and Victor Bayon[2]

[1] Sun Microsystems, Dr-Leo-Ritter-Strasse 7, 93053 Regensburg, Germany
thijs.metsch@sun.com,
WWW home page: http://www.sun.com
[2] Intel Innovation Centre, Intel Ireland Limited (Branch), Collinstown Industrial Park, Leixlip, Kildare, Ireland. andrewx.edmonds,victorx.m.molino@intel.com,
WWW home page: http://www.intel.com

Abstract. Today's need for Open Frameworks, especially in the Cloud community, demands for standards. These standards are needed to ensure portability, interoperability and integration of Cloud sites. This paper describes how an emerging Cloud standard, the Open Cloud Computing Interface, can be used integrate two major Cloud Frameworks (SLA@SOI and RESERVOIR). Both of the Clouds Frameworks are developed with a focus on elasticity. Within the project different management layers are used to control the life-cycle of the Cloud Services. Still both frameworks can interoperate by using the Open Cloud Computing Interface API.

Key words: Cloud Computing, Standards, Infrastructure as a Service, Hybrid Clouds, Resource Management, Cloud frameworks, Autonomic Cloud Computing platforms, Service Level Agreements, Monitoring

1 Introduction

What today is called Cloud computing comes a long way down the road. It had other names before and many technologies are involved in it. Virtualization, utility computing, and grid technologies are among the most representative.

Cloud offerings can be classified according to the resources they offer 'as a Service' (XaaS): Infrastructure as a Service (IaaS) that allows to allocate virtual machines and storage capacity; Platform as a Service (PaaS) where users are provided with remote software platforms to run their services; and Software as a Service (SaaS) where applications are moved to the Internet and accessed through web interfaces.

Cloud frameworks on the other hand can be seen as the software environment in which Cloud services can be deployed. Most of the frameworks have automatic and elastic management solutions inherited which control the life cycle, placement and Service Level Agreements (SLAs) of a Service.

Different frameworks have come up over the past time, including commercial proprietary and open frameworks like those developed in the European Union

Framework 7 Programme Projects RESERVOIR [1] and SLA@SOI [2]. They consist of similar modules and layers but have different basic architectures. They can be messaging based or client/server, have a focus on IaaS or SLA management. The different foci leads to different architectures.

Still all Cloud frameworks seem to need a layer to deploy Virtual workloads on infrastructure. Having the frameworks interoperate even makes more sense with the background of using each others abilities and functionalities. In this manner services could be moved to the Cloud framework which would best fit their needs and their characteristics.

Having the need to interoperate comes with the demand for standards. Only if the frameworks support certain standardized interfaces, can interoperability be achieved. This paper tries to show the overall setup and ideas behind two Cloud frameworks and includes the description of an upcoming Cloud standard. A architecture which combines all three aspects is proposed as well.

The paper is structured as follows: At first two major Cloud frameworks are presented in sections 2.1 and 2.2. More details and details about interoperability of the frameworks are presented in section 3. The necessary standard for this is presented in section 3.1. A architectural approach is demonstrated in section 4.

2 Cloud frameworks

The two Cloud frameworks described in this paper are part of the development of the RESERVOIR and SLA@SOI projects. Although on a high level view they both manage services based on SLAs there are important differences.

The RESERVOIR framework shows a more layered approach in which the focus is on the management of virtual workloads. These workloads can be of different types like virtual machines or even Java services. Also RESERVOIR tries to break the barriers between Cloud sites which include geographical, security and management issues. Cloud services can be moved between sites to gain Interclouds.

While doing so the services have attached SLAs which for example can state that a Cloud service A should stay on the private Cloud so a Service B must be moved to a public Cloud to free resources for Service A.

The SLA@SOI framework provides a multi-layer, multi-domain SLA management framework. It is targeted at providing these management capabilities to service providers, from business service providers all the way down to infrastructure service providers.

The project tries to create an environment in which Cloud services can be traded as economic goods. For example when deploying a service, a set of dependable conditions can be defined.

Still both frameworks are able to manage virtual machines and the according SLAs. Scaling up and down as well as cloud bursting scenarios can be handled.

2.1 The SLA@SOI framework

The research project SLA@SOI aims at providing a major milestone for the further evolution towards a service-oriented economy, where IT-based services can be flexibly traded as economic goods, i.e. under well defined and dependable conditions and with clearly associated costs. Eventually, this will allow for dynamic value networks that can be flexibly instantiated thus driving innovation and competitiveness.

The project SLA@SOI envisions "a business-ready service-oriented infrastructure empowering the service economy in a flexible and dependable way", where business-readiness requires the following three major characteristics:

Predictability and Dependability The quality characteristics of services can be predicted and enforced at run time.

Transparent SLA management Service level agreements (SLAs) defining the exact conditions under which services are provided/consumed can be transparently managed across the whole business and IT stack.

Automation The whole process of negotiating SLAs and provisioning, delivery and monitoring of services will be automated allowing for highly dynamic and scalable service consumption.

A motivating business scenario highlighting the project idea is a service provider who is enabled to offer services with differentiated, dependable and adjustable SLAs, and can negotiate concrete SLAs with (individual or groups of) customers in an automated fashion, adding value to provisioning systems. This business goal imposes additional requirements on software providers (to provide components with predictable non-functional behaviour) and infrastructure providers (to support an SLA aware management of resources).

This vision maps to the overarching challenge for a service-oriented infrastructure (SOI) that supports consistent SLA management across all layers of an IT stack and across the various stake-holder perspectives. Noteworthy, the SLA characteristics may span across multiple non-functional domains such as security, performance, availability, reliability. This paper focuses on the infrastuctural part of the sla@soi framework.

Figure 1 gives a simplified overview on how such a systematic SLA management process may look like. As today's business systems typically consist of complex layered systems, user-level SLAs cannot be directly mapped onto the physical infrastructure. Services might be composed of other more fundamental services that could be also provided by external parties. Consequently, a stepwise mapping of higher-level SLA requirements onto lower levels and the aggregation of lower-level capabilities to higher levels is crucial for grounding user-level SLAs to the infrastructure.

This vertical information flow must carefully reflect service interdependencies as well as the originating business context. In addition to SLAs, the vertical information flow also covers monitoring, tracking, and accounting data and must support brokering and negotiation processes at each layer. As shown in the figure, the overall SLA management process may include different stake-holders, namely

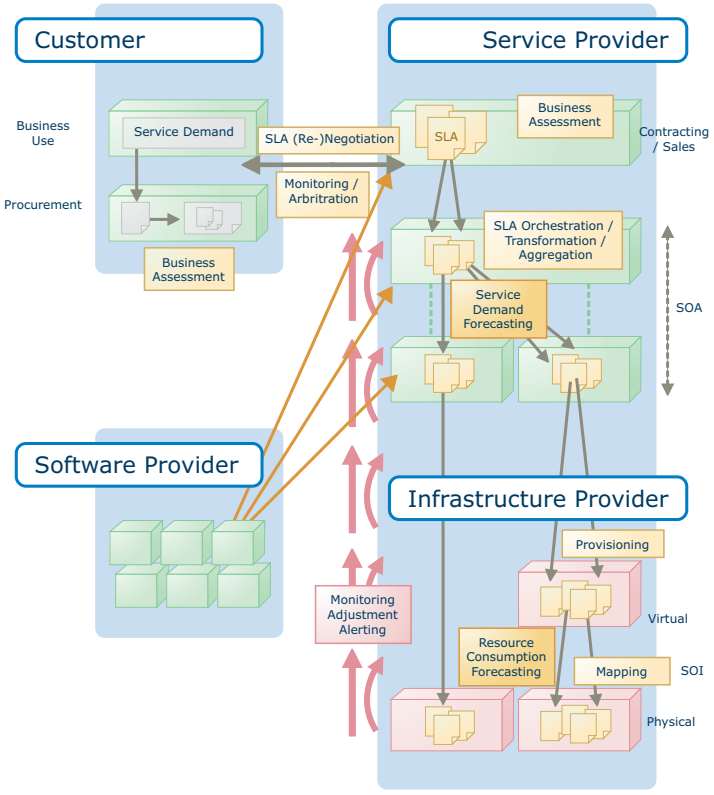

Fig. 1. The high-level SLA@SOI architecture.

customers, service and infrastructure providers, and also various business steps such as business assessment, contracting and sales. The overview is intentionally simplified in the sense that no service chains are visualized. Such chains would represent all cases where service providers rely on other external providers.

Infrastructure SLA Management For the purpose of this paper and discussion, we focus on the infrastructure provider's perspective (see figure 1). Today's IaaS offerings tend not to be automated, what's provisioned is not easily verified [3], typically static, require manual intervention in the case of claims, making for inefficient processes. Many current infrastructure as a service (IaaS) providers consider SLAs to be "best effort". For example Slicehost [5] considers SLAs to be "silly" [6], Amazon EC2 [7] SLAs [4] are not machine readable nor negotiable.

An infrastructure provider client should get what they asked for from that provider. If you have a guarantee and a machine readable SLA that an associated infrastructure framework understands then you can verify that the provider is holding up its end of the agreement.

Infrastructure SLA model and Resource Model An Infrastructure SLA (iSLA) within SLA@SOI is conceptually comprised of:

Set of terms: Terms in an iSLA are the attributes that define the eventual representation of an instantiated infrastructure resource. These within the SLA@SOI framework are broadly split into two categories, functional and non-functional and are in principle immutable. A change of the value of the term will mean an SLA violation. Functional attributes are those that define attributes that are core to the operation of an infrastructure resource. Examples of these include: number of CPU cores, CPU slice (how much percentage of CPU a instantiated resource is allow to take), amount of allocated memory, root partition size, network, etc. Non-functional attributes are those that do not directly affect the operation of an executing infrastructure resource. Examples of these include: resource's geographic region, security, green policy, replication, etc.

Set of SLOs: SLOs are the explicit and possibly implicit attributes that are to be monitored for changes in their value and are associated at runtime properties of the resource. The SLOs are mutable. An example of an SLO is CPU or I/O utilization. With different levels of granularity SLOs could be for example shared among the service provider and the service consumer. The service consumer will know about how much capacity of the current provided solution is using and the service provider could recommend a better updated configuration with less or more resources as required.

Conditional rules: Where iSLA terms and/or SLOs change above or below a particular explicit or calculated values then a set of associated actions are executed. For example, if a provisioned resource unexpectedly terminates the iSLA term state changes and an SLA violation action is executed. Another example is if CPU utilization is too high for a pre-determined period of time, an SLA warning action is executed. Both actions are of the interest of the resource provider and consumer. Currently within the SLA@SOI framework, the consumer of the infrastructure can listen to such events by means of subscribing to a private pub/sub channel.

Relationship between iSLA Registry and Infrastructure Landscape Once an iSLA is agreed and provisioned this action links the associated provisioned resources, reflected in the infrastructure landscape (IL), with the iSLA. iSLAs are persisted in a iSLA registry and if one is to inspect the iSLA registry, a collection of iSLAs can be viewed and their associated resources. This allows management be navigated from the iSLA to the Resource level via the iSLA registry through to the IL.

These iSLAs are managed by the iSLA manager and persisted within an IL, which represents the instantiated resources and immutable SLA terms. The current SLA@SOI infrastructure resource model represent the instantiated SLA plus other running parameters. The main entities considered in the model are Compute, Storage and Network with particular attention paid to the compute entity from a technical and implementation point of view.

When instances of this model are created through the provisioning process, they are registered within the IL. Also registered in the IL are the physical resources on which virtual ones execute upon. The IL is in effect a configuration

management database. Free and used resources can be monitored here. When the running state of a resource changes, it is checked against the IL.

The registration and publishing the information against the IL is done by means of the pub/sub messaging pattern. Each physical resource is responsible for updating its status and the status of the provisioned resources (virtual machines) that it manages.

SLA@SOI Architecture In an ongoing process the existing architecture [8] has been further refined such that the that the separation of concerns btween Resource management and SLA management. This will provide the advantage such that integration with other Resource managers will quite trivial to achieve. The major entities within the SLA@SOI framework are SLA, Service and Resource and their relationships are depicted in figure 2.

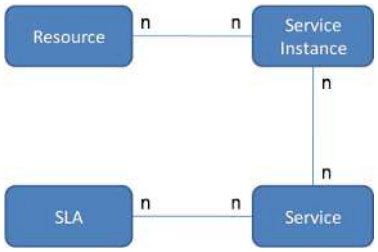

Fig. 2. SLA@SOI Architectural Entity Relationships.

An infrastructure SLA manager is linked with the Resource Manager. The Resource manager offers a number of services and manages service instances. The SLA manager accepts SLA offers and advertises service offerings, otherwise known as SLA Templates. SLA offers are requests for infrastructure, defined by the iSLA model and are the basis of the negotiation that happens between clients and the infrastructure SLA manger. Clients can query the infrastructure SLA manager in order to see what SLA Templates the iSLA manager supports. Once the SLA offer has been successfully negotiated the client can then provision the agreed SLA. In order to support the architecture scenario outlined, it is necessary for the Resource manager to support a number of critical functional APIs.

Required services for infrastructure SLA management:

Services Offer Listing: This service supplies a listing of the services that the Resource manger can offer. These service offers are known as SLA Templates with the SLA@SOI framework. Examples of these include a compute service and storage service. The compute service offers the service to allow for computation to be executed on the provider's infrastructure. This is typically take the form of a virtual machine in the IaaS world. In the example of a storage service, this offers a service to allow storage of data.

Service Instance Creation: For every service offered, an API for creating the related service offers is required. At a minimum, the basic operations of Create, Retrieve, Update and Delete (CRUD) should be offered.

Service Instance Monitoring: This service is in place to guarantee that the terms specified by the iSLA are delivered and that are not changed while the resource is active. In terms of infrastructure, basic parameters such as CPU slice, amount of memory, network capacity are monitored to verify guarantees are met and not violated by the resource provider. For example the resource provider might under-provision the resource during provisioning time and this situation should be reported to the consumer of the resource.

Service Instance Adjustment: Again, for every service offered, a means to change the service instances configuration should be offered. This service, in practical terms, should be simply a proxy to the service instance creation API, where by it utilises the Update functionality of the service instance creation API.

2.2 The RESERVOIR framework

The RESERVOIR project [1] tries to create an environment in which Cloud Services can easily be deployed and managed. This environment provides an abstraction layer from specific resources, platforms and geographical locations.

Beside the major idea of creating abstractions, the project tries to create an environment in which services can be migrated. Migration should be possible across network and storage boundaries. In this case, both a "live" and a "suspend and resume" method of migration are supported.

RESERVOIR encapsulates all of these ideas of an environment, in which different kind of services can be managed. It also tries to create a basic model of how Cloud frameworks can look like when overcoming political, security, geographical and migration boundaries.

The first abstraction the RESERVOIR project uses is the encapsulation of services. One service or a group of services can run inside a Virtual Execution Environment (VEE).

Several VEEs can then run on a Virtual Execution Environment Host (VEEH); meaning one physical resource. Each host has a virtualization technology installed in which the VEEs are deployed and hosted. While one VEE can consist of more than one service, VEEs can be collocated on one or spread across several VEEHs. A small component for management and monitoring is also available on all VEEHs.

The overall management of all VEEHs is realized in the Virtual Execution Environment Management (VEEM) system. It is in charge of the deployment of VEEs on top of VEEHs. It can bootstrap and unload VEEHs. VEEs can be moved around and placed according to the VEEM setup.

To complete the overall system, a Service Manager is needed. It is responsible for the instantiation of service applications. It, therefore, requests the VEEM to create and manage VEEs. Beside this the Service Manager also tracks and

manages the SLAs. It ensures that all SLA policies are satisfied at all times. To do so, it monitors the current state of the overall system and executes the elastic rules. [9]

RESERVOIR can be seen as the perfect environment in which Cloud services can be deployed which have different needs for resources at different times. Overall multi-tier cloud service can be deployed which have a high demand for compute resources can be managed easily.

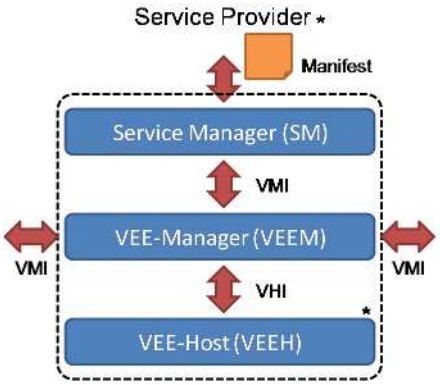

Fig. 3. The high-level RESERVOIR architecture.

Virtual Execution Environment Management Within the RESERVOIR project OpenNebula [10] is used as a VEEM. The Service Manger interface with the VEEM through the VEE Management Interface (VMI). The VEEM itself use a VEE Host Interface (VHI) to communicate with the VEEHs.

OpenNebula is an open source virtual infrastructure engine which provides features to control, deploy and monitor virtual machines. A modular and open architecture is used which support the integration of all kind of hypervisors.

Three central components are present within OpenNebula. The Core is a centralized component which manages the life-cycles of the virtual machines. Next to the core a Capacity Manager adjusts the placement of the virtual workloads. To support multiple hypervisors a third component is used for abstraction. The Virtualizer Access Layer exposes the features of the hypervisors to the Core. [11]

To interface with an OpenNebula system the OpenNebula Cloud API (OCA) supports the development of Cloud interfaces. This interface is used to implement the VMI inside of the RESERVOIR architecture. Next to this OpenNebula also support a subset of Amazon's EC2 queries.

To place the virtual workloads on the resources requires the VMI to be as precise as possible. This means that requests need to be complete and information regarding the placement need to included. The Open Cloud Computing Interface (see section 3.1) will form the base of the VMI.

3 Gaining interoperability.

Interoperability can be reached when two systems use the same interface and so can exchange information. A common interface for both previously described frameworks would be an API which supports the deployment, management and monitoring of virtual workloads like virtual machines.

3.1 The Open Cloud Computing Interface

The Open Cloud Computing Interface (OCCI) is a working group organized within the Open Grid Forum [12]. Motivation for founding this group was the lack of a standard for *Infrastructure as a Service* model based clouds. The open standardization process is driven by the following topics:

Interoperability describes the ability to let different systems integrate with each other. Most known use case for this is the creation of *Interclouds*.
Portability defines the need for easy code reuse in end-user application like cloud clients or portals.
Integration is the idea of wiring up Cloud computing services with legacy resources.

With the focus of providing Infrastructure as a Cloud Service the OCCI group will define a RESTful [13] API. The goal is to create a small interface which can be easily extended. The RESTful approach supports this.

Based on use cases and requirements which define how the end-users interfaces with the Service provider a first API is defined. It is driven by the idea of having a set of nouns on which verbs can operate and a set of attributes bound to them. Moreover the Service provider offers compute, network and storage resources (Nouns) which can by manipulated using CRUD (Create, retrieve, Update, Delete) operations. Each resources has a set of attributes. For a compute resources this includes, but is not limited to:

compute.cpu.arch describing the architecture platform
compute.cpu.speed stating the speed of the CPU
compute.cpu.core number of cores per CPU
...

Each resource is identified by a unique URI. A compute resource might therefore be accessible via `http://abc.com/compute/uid123`.

Since this is a RESTful API the CRUD operations are mapped to the HTTP commands. Create, retrieve and delete are mapped to HTTP POST, HTTP GET and HTTP DELETE. Update would be a combination of HTTP GET followed by a HTTP PUT.

The request to create a compute resource looks like:

```
GET /compute
Host: abc.com
```

```
Authorization: Basic xxxxxxxxxxxxxxxxxxx
...
compute.cores=2
compute.memory=2048
```

If the creation was successful the Service provider will return the *HTTP 200 OK* code.

State transitions of resource to trigger migrations or similar operations can be achieved by a HTTP POST on a URI which ends on */requests* (e.g. POST /compute/uid123/requests).

The rendering of the data format is achieved using different formats like JSON, XML and TXT. Extensions to the API can be freely added by the Service providers. Currently OCCI defines some extensions for monitoring and caching. [14]

4 A hybrid Cloud Framework

In order to allow for IT-services to be traded as a commodities there is a requirement for different infrastructure service providers to interoperate via standardized interfaces that offer both access to provisioning and iSLAs management capabilities that can be managed from the consumer's perspective across domains and in multi-providers settings in a transparent and unified way.

As more services are moved to the Cloud, there will be also a need for the service consumer to retrieve information about all stages of service provisioning and service usage during runtime. Service providers should offer different layers of functionality and granularity, such as monitoring. The service consumer will require this level access in order to perform its own monitoring of the provisioned services.

With the cloud frameworks of SLA@SOI and RESERVOIR presented we now detail a use case scenario that illustrates the benefit that OCCI brings to the scenario. In this scenario the SLA@SOI framework is configured to only provision and manage iSLAs. There are two infrastructure provisioning systems; one that is the "native" SLA@SOI provisioning system and another 3rd party provisioning system, the RESERVOIR VEEM. The topology of such a scenario is illustrated in figure 4. Naturally the SLA management layer could know about more providers but for the purposes of simplicity we have selected these two.

The SLA@SOI framework has the responsibility of acting as a broker. Requests, in the form of SLAs, are supplied to the SLA@SOI SLA management layer. Within the SLA management layer, a number of provisioning systems are known. The iSLAs presented to the SLA manager which then selects the most appropriate provider based on the terms (functional and non-functional parameters) within the iSLA. Once selection is accomplished the SLA manager then negotiates for the required amount of resources, provisions and then begins to monitor the resources based on the iSLA's SLOs and conditional rules.

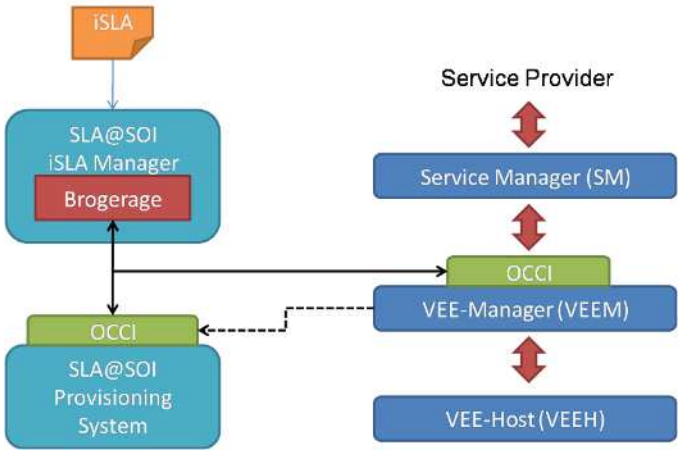

Fig. 4. The Use-Case Scenario Topology.

In this particular use case scenario the iSLA presented to the SLA management is one that requires support for a federated multi-tier Cloud services. This requirement cannot be satisfied by the internal SLA@SOI provisioning system but can be through RESERVOIR. The SLA management layer knows this as it can retrieve the services offered by each provider through the Service Listing Interface.

Provisioning and monitoring is then carried out and mediated by the OCCI interface. The advantage in this approach is that there is no custom provider-specific implementations required to enable firstly these actions and secondly as explained previously provides a way for new providers to be utilized owing to the ease of implementing OCCI. The selection of providers carried out by the SLA manager between the 2 systems is made easy via OCCI, otherwise the selection is made all the more difficult due to the differences in the two systems provisioning APIs. Then SLA@SOI uses OCCI to go straight to our two different infrastructure management layers. In this use case SLA@SOI is used as a means to perform infrastructure provisioning negotiation, brokerage and the management of the agreed iSLAs. OCCI provides the capability for two major European FP projects, RESERVOIR and SLA@SOI, to easily integrate by sharing a common infrastructure resource manager interface (provided by OCCI).

5 Conclusions

The proposed architecture in this paper showed how two cloud frameworks could be coupled to gain interoperability. The usage of OCCI allows the SLA@SOI framework to deploy workloads to the RESERVOIR environment. All this is done in a way that no SLAs are breached.

OCCI plays a major role in this setup. It describes the interface between the two frameworks. Is one of the first standards which arises in the cloud community. But the current specification can already be used to propose the architecture.

Overall this setup showed that it can be possible to let two Cloud frameworks interoperate even with their different architectures. The overall advantage would be that Cloud service can be deployed and managed in a environment which best fits their need. In this case groups of services which intercommunicate can be managed in the SLA@SOI framework and deployed in the RESERVOIR framework.

Notes and Comments. This paper demonstrates a work in progress approach for interoperability of Cloud frameworks. The Open Cloud Computing Interface specification will be released in near future.

The authors would like to thank the members of the Open Cloud Computing Working Group.

The research leading to these results is partially supported by the European Community's Seventh Framework Programme ([FP7/2001-2013]) under grant agreement no.215605 (Reservoir) and no.216556 (SLA@SOI).

References

1. RESERVOIR project website `http://www.reservoir-fp7.eu`
2. SLA@SOI project website `http://sla-at-soi.eu/`
3. John Allspaw: Slides from Web2.0 Expo 2009. (and somethin else interestin), retrieved on the 6th August 2009. `http://www.kitchensoap.com/2009/04/03/slides-from-web20-expo-2009-and-somethin-else-interestin/`
4. Amazon EC2 (see Credit Request and Payment Procedures), retrieved on the 6th of August 2009. `http://aws.amazon.com/ec2-sla/`
5. Slicehost VPS Hosting, retrieved on the 6th of August 2009. `http://www.slicehost.com`
6. Slicehost VPS Hosting SLA, retrieved on the 6th of August 2009. `http://www.slicehost.com/questions/#sla`
7. Amazon EC2, retrieved on the 6th of August 2009 `http://aws.amazon.com/ec2`
8. Wolfgang Theilmann, Ramin Yahyapour, and Joe Butler: Multi-level SLA Management for Service-Oriented Infrastructures, Service Wave (2008)
9. Caceres, J., Montero, R., Rockwerger, B.: RESERVOIR - An Architecture for Services, RESERVOIR project, `http://www.reservoir-fp7.eu/twiki/pub/Reservoir/Year1Deliverables/080531-ReservoirArchitectureSpec-1.0.PDF` (2008)
10. OpenNebula website `http://opennebula.org`
11. Sotomayor, B., Montero, R., Llorente, I., Foster, I.:Capacity Leasing in Cloud Systems using the OpenNebula Engine, Workshop on Cloud Computing and its Applications (CCA08) (2008)
12. Open Grid Forum `http://www.ogf.org`
13. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures, Doctoral dissertation, University of California, Irvine (2000)
14. Open Cloud Computing Interface working group website `http://www.occi-wg.org`