
# SeaClouds Project

# D2.3.1 - Periodic Standardization report

| | |
|---|---|
| Project Acronym | SeaClouds |
| Project Title | Seamless adaptive multi-cloud management of service-based applications |
| Call identifier | FP7-ICT-2012-10 |
| Grant agreement no. | 610531 |
| Start Date | 1st October 2013 |
| Ending Date | 31st March 2016 |
| | |
| Work Package | WP2 Requirements Analysis, overall Architecture and Standardization |
| Deliverable code | D2.3.1 |
| Deliverable Title | Periodic Standardization report |
| Nature | Report |
| Dissemination Level | Public |
| Due Date: | M10 |
| Submission Date: | 31st July 2014 |
| Version: | 1.0 |
| Status | Final |
| Author(s): | Alex Heneveld (Cloudsoft), Roman Sosa (ATOS), Jose Carrasco (UMA) |
| Reviewer(s) | Francesco D'Andria (ATOS), Ernesto Pimentel (UMA) |

Dissemination Level

| Project co-funded by the European Commission within the Seventh Framework Programme | | |
|---|---|---|
| | Public | X |
| | Restricted to other programme participants (including the Commission) | |
| | Restricted to a group specified by the consortium (including the Commission) | |
| | Confidential, only for members of the consortium (including the Commission) | |

## Table of Contents

## Executive Summary

This deliverable, D2.3.1, summarises the main standards relevant to our work in SeaClouds, in particular outlining the areas where we are consuming the standards, contributing to the standards, or fostering adoption of the standards.

The structure of this document is the following:

- Section 2: This section outlines why we view standards as important to our research and to the wider community
- Section 3: This section outlines the key standards in four areas:
    - Application Topology
    - Infrastructure-as-a-Service
    - Platform-as-a-Service
    - Service Level Agreements

## 2. The Importance of Standards in SeaClouds

Cloud computing is an emerging topic of the distributed computing that may offer many benefits to organizations by making information technology (IT) services available as a commodity and accessible from the web.

In general, the cloud-computing community sees the lack of cloud interoperability as a barrier to cloud-computing adoption because organizations fear “vendor lock-in.” Vendor lock-in refers to a situation in which, once an organization has selected a cloud provider, either it cannot move to another provider or it can change providers but only at great cost.

On the other hand, SeaClouds wants to reduce time-to-market and provides on-demand scalability at a low cost, managing, in an efficient and adaptive way, complex multi-services applications over technologically heterogeneous Clouds environments.

Below some tangible examples of how standards can benefit cloud computing:

- Standards-based descriptions are attractive to users -- especially in enterprise -- as they don't develop on proprietary formats.
- Easy to try our system with existing descriptions from others - we win based on functionality, not lock-in of the description format
- Can tie in to existing documentation and conventions -- we are investing work in features, not reinventing description syntax

# 3. Relevant Standards and Open Source Systems, and Our Work in SeaClouds

## 3.1 Application Topology

The leading standards in representing application topology are OASIS CAMP and TOSCA.

### 3.1.1 CAMP

OASIS CAMP — Cloud Application Management for Platforms — describes a REST API for runtime management of applications. It supports investigating topology, sensors, and operations, and it supports deployment of applications. Deployments are presented as "plans" based on a declarative YAML syntax consisting of artifacts and services together with their requirements and characteristics.

### 3.1.2 TOSCA

OASIS TOSCA — Topology and Orchestration Specification for Cloud Applications — describes how an application should be deployed. In its original form it specifies an XML syntax describing nodes, services, and relationships, along with the type definitions and implementational details. A simpler YAML "profile" is in development.

**Why Relevant**

The description of an application's topology is essential as input to SeaClouds and useful as intermediate representations and presentation back to a user. By following open standards, we increase the potential for SeaClouds to interoperate with other tools on the inbound and outbound sides.

**Our Contributions**

One or more SeaClouds members have been involved with both technical committees, suggesting improvements to the specifications based on our activity. Many of the SeaClouds memers have been contributing to Apache Brooklyn which provides an implementation of CAMP (earlier version as of this writing) and is starting to develop an implementation of TOSCA.

## 3.2 Infrastructure-as-a-Service

The IaaS providers, APIs, standards, and technologies are too numerous to list exhaustively, but we will give a summary of several of the main ones.

### 3.2.1 OpenStack

Although its focus is on being a standard implementation rather than an API, OpenStack, driven by the OpenStack Foundation, far and away has the largest momentum of any standard in the area. Its core is a set of APIs and implementations for compute, storage, and networking, with a very broad set of adjacent projects (including Heat which starts to touch on Application Topology and is related to TOSCA through the HOT templating language). It is worth noting that although the official OpenStack project defines APIs and implementations, most distributions (Red Hat, Piston, Mirantis) and service offerings (Rackspace, HP Cloud) come with extensions and variations.

### 3.2.2 CloudStack

Apache Cloudstack is one of the most mature IaaS platforms. It has widespread adoption and a broad API covering compute, network, and storage.

### 3.2.3 Other Cloud Systems

There are dozens of other clouds; the two above are the leading open-source systems, but it is worth mentioning Amazon Web Service's EC2 platform, Google Compute, IBM SoftLayer, Azure, GoGrid, Digital Ocean, and Eucalyptus (EC2-compatible on-premise deployment). It is worth noting that these all have quite different APIs.

### 3.2.4 Other Standards

Various IaaS standards have been proposed, including OCCI and CIMI, but these have not been adopted as much as the systems mentioned above.

### 3.2.5 Multi-Cloud Client Bindings

To solve the problem of deploying multi-cloud, several client libraries have emerged. Apache Jclouds provides JVM bindings; Fog provides Ruby bindings; and libcloud provides Python bindings. These all support a wide range of the systems mentioned previously.

**Why Relevant**

SeaClouds will in some cases be consuming IaaS, and it is a requirement that this be done in a portable, multi-cloud manner.

**Our Contributions**

SeaClouds is actively using Apache Jclouds via Apache Brooklyn to support all of the systems identified above. SeaClouds members have been contributing directly to Apache Jclouds and to Jclouds support in Apache Brooklyn.

## 3.3 Platform-as-a-Service

### 3.3.1 The State of the Art and the Need for Standards

Standardization is the natural process in some enterprise evolution contexts. It is very useful in terms of quality control, interoperability, business expansion, etc. In the a cloud computing environment, several providers offer PaaS level services such as Google App Engine (https://appengine.google.com/), AWS Elastic Beanstalk (http://aws.amazon.com/elasticbeanstalk/), OpenShift Origin (http://openshift.github.io/) Heroku (https://www.heroku.com/), RackSpace (http://www.rackspace.com/), etc.

However, as we have mentioned above each provider defines its own API to display its services, non-functional requirements, QoS, add-ons, etc. As a result, cloud developers are often locked into a specific set of services from a specific cloud environment.

Thus, it is difficult to maintain the interoperability and portability between services of the different providers, which may be used in the deployment of an application. In this regard, the standardization PaaS gains importance, as it ensures the portability and interoperability of the applications and any system in general, which uses the services offered through different clouds.

Currently, PaaS services are conceived as a way to ensure the vendor lock-in, due to the aforementioned constraints. However, a correct standardization (or at least unification) of the cloud providers industry could help to prevent vendor lock-in issues and allow users to benefit from the principal advantages of cloud features such as the elasticity and high availability. Thus, we find a lot of organizations and enterprises which propose their own approaches, standards, technologies and other methodologies to avoid part of the vendor

lock-in issues. Typically, to achieve this they need to be defined in such a way as to ensure the portability of the applications and their dependencies.

Dependencies are one of the key aspects of PaaS which have to be standardized to ensure the portability of the applications. Typically, the dependencies are composed by the language runtime, data store, external services, framework, etc. If these dependencies are maintained, a developer will be able to move applications to a new PaaS target without critical changes. We have already mentioned the CAMP and TOSCA standards, which describe their own application modelled methodology. In this case, these standards could be very useful because they both describe the necessary elements for deploying and running the application: Types and Templates in TOSCA and the definition of the application's component (plan) through the common API in CAMP.

Although we consider that standards are defined by a consortium formed by cloud experts, we must also bear in mind the current available technologies.

### 3.3.2 Heroku and Build Packs

Heroku defines Buildpack [1] to package the application dependencies and to port the application runtime and frameworks. This methodology was designed by Heroku maintaining a separation with its PaaS services, and determining how to build up the application and the necessary resources. Thus, Buildpacks are becoming a standard used by the greater part of the PaaS ecosystem and some providers allow the applications to be deployed without any modification, for example as Cloud Foundry does (http://docs.cloudfoundry.org/buildpacks/). Other providers may need to perform some modifications of the standardized Buildpacks, which does not guarantee full portability, but they can be easy to modified.

### 3.3.3 OpenShift

OpenShift (Red Hat) also defines its own methodology for packaging the applications and their dependencies, Cartridges [2]. They provide the necessary commands and control for the functionality of the software that is running the users' applications (source [2]). It allows new elements to be defined and added to the OpenShift platform, in order to extends its functionality. A new software (server, database or language support) which may not exist in OpenShift could be defined through this methodology. In a similar way to Builtpacks,

Cartridges enable the modelling of the applications and the dependencies needed for it to work. Typically, it is composed by a root directory which contains all the logic for components that an application could need. Furthermore, OpenShift was built to work over public and private clouds in a similar way so, although this methodology is not as standardized as Buildpack, it may be a good option to take advantage of hybrid clouds.

### 3.3.4 Stratos

Similary, Apache Stratos (http://stratos.apache.org/) is a PaaS framework that defines another packaging format for applications and their dependencies. It has been designed to avoid the aforementioned problems in the context of PaaS. Moreover, it also works over IaaS services and it could be extended to support Apache jclouds (http://jclouds.apache.org/) (source [3]).

### 3.3.5 Cloud Foundry

Cloud Foundry (http://cloudfoundry.org/) is rapidly emerging as the leading player in the PaaS space. Cloud Foundry is an open source PaaS, proposed by VMWare (http://www.vmware.com/), spun out as the company Pivotal, and now brought to a new foundation, the Cloud Foundry Foundation. The Cloud Foundry Core defines a baseline of common capabilities to promote Cloud portability across different instances of Cloud Foundry. We have mentioned that this approach is a cloud computing PaaS, however it needs an IaaS layer over deployed applications. Currently, Cloud Foundry supports AWS, OpenStack (mentioned above), and VMware clouds [4], through the BOSH Ruby tool chain. So, it is very easy to define an application and its dependencies using the aforementioned Build Packs and to deploy it in a **portable way** over the IaaS supported, so long as Cloud Foundry and the Build Packs are supported in that IaaS. (Note that Cloud Foundry canbe used over public, private and hybrid clouds).

In this sense, Cloud Foundry provides an abstraction of the IaaS infrastructure through the PaaS level.

### 3.3.6 Engine Yard

Engine Yard (https://www.engineyard.com/) is another leading PaaS, focused on management of Ruby enterprise application, but it is support several languages and technologies too. In this case, Engine Yard wraps the AWS and Azure Infraestructura above

a PaaS layer, providing a independence of the platform mechanisms which, based over IaaS, ensures scaling, high availability, replication, monitoring, and so on.

### 3.3.7 App Fog

Also, we can include AppFog (https://www.appfog.com/) within this scope. It is a kind of Cloud Federation, whose system enables the user to select the infrastructure on which his software is going to be deployed from among several commercial solutions. AppFog is based on the Cloud Foundry Open Source Project and can used to deploy an application on serveral infrastructures such as AWS (different regions are supported), Azure, HP OpenStack and RackSpace. Moreover, AppFog provides a management layer from which it is possible to monitor the status of the resources in use during the deployment and the rest of the lifecycle, e.g. number of instances used by our system, or the amount of memory used. AppFog supports several languages and provides some extremely useful templates and add-ons to automatize the initialization and the configuration of the frameworks and other dependencies needed by the applications.

### 3.3.8 OpenStack Solum

OpenStack Solum aims at streamlining how OpenStack components can be leveraged as part of a PaaS ecosystem. It follows the common model of language packs (build packs) and services, together with deployable archives. Solum is adopting the CAMP standard for deployment description and a standards-compliant REST API.

### 3.3.9 A Note on Portability

With respect to the standardization and vendor lock-in, although when the software is deployed, it can be migrated from one IaaS to another with minimal user interaction , it is however, it is centred on IaaS portability and does not address the lock-in at level of PaaS.

We have mentioned upon the Cloud Federation. With regard to inter-cloud operation, it is a platform where the user can select the infrastructure in which to deploy his software across a set of third-party solutions. As an example, Paraiso et al [5] propose to define the PaaS level allowing selection on the level of IaaS. The Open Source Cloudware (http://www.ow2.org/view/Cloud/) community (OW2) has motivated many projects in the area of Cloud interoperability. One example is The Open Cloudware project (http://www.opencloudware.org/), which aims to buid an open software engineering

platform for the collaborative development of distributed applications to be deployed across multiple Cloud Infrastructures.

OpenShift Origin (already mentioned) is an open source project developed by the Red Hat foundation and defines a Platform as a Service (PaaS). Although, it maintains the baseline described in the approach that we have already analyzed, it can be run on top of several public providers, AWS, OpenStack, HP OpenStack, Rackspace, etc. Moreover, it provides private support to use on top of our own data center including vSphere, KVM (http://www.linux-kvm.org/), vCloud…

Also, it can be used over local hypervisor installations such as VirtualBox (https://www.virtualbox.org/). Note that typically, in the same way as its other competitors (e.g. Heroku), it runs over AWS.

In this regard, we can include AppScale (http://www.appscale.com/). Again, it is an open source project which defines a Platform as a service over several public and private IaaS such as AWS EC2, Google Compute Engine, Rackspace, OpenStack and CloudStack and Eucalyptus. Also, it supports virtualized cluster such as vSphere and hypervisors such as Virtual Box (source [6]). One of the goals of the aforementioned project is to provide an API for the developers, which allows portable applications to be built in the different IaaS. However, it principally focuses on the portability from Google App Engine applications to the aforementioned IaaS, preventing part of the vendor lock-in. It is very useful because, Google App Engine is one of the providers which define the most restrictions to work properly. In this regards, if an application has to be migrated from Google App Engine to another target platform, it has to critically modify part of the application structure.

### 3.3.10Relevant Research and Interoperability Projects

In a research context, we find several projects which try to wrap the offered services of the different providers in order to make them compatible and to mitigate the vendor lock-in issues.

The Broker@Cloud project is (http://www.broker-cloud.eu/) an EU FP7 funded collaborative project aiming to assist enterprises to move to the cloud while enforcing quality control on developed services. Broker@Cloud supports IaaS and SaaS services too. Capabilities for cloud service governance and quality control such as lifecycle management, dependency

tracking, policy compliance, SLA monitoring, and certification testing are included in the project. Brooker@Cloud has a matchmaking algorithm based on semantic models. So, it finds and selects the services of the available brokers whose properties best match those needed by the developers.

The MODAClouds (http://www.modaclouds.eu/) is also an EU FP7 project. It aims provide quality assurance during the application's life-cycle, supporting migration from cloud to cloud when needed, and supplying techniques for data mapping and synchronization between multiple clouds, mitigating part of the portability vendor lock-in issues. Furthermore, services of any nature (IaaS, PaaS, SaaS) can be adapted, enabling their management from MODAClouds. These services can be displayed by public and private clouds (so hybrid clouds are supported too). To do so, MODAClouds requires software developers to adopt a Model-Driven Engineering approach in order to obtain an automatic deployment on multiple providers while hiding the technology stack. MODAClouds provides systems support and risk analysis methods and proper guidelines in order to select the environment to carry out the application (or system) deployment. In this sense, MODAClouds takes into account the resource prices over time, performance, geographic location, etc. Moreover, this project defines a monitor methodology independent of any provider API. For this purpose, MODACloud describes the data collertors which are little programs that are executed in the deployment environment. These are run in a cyclical manner auditing the run-time environment status to detect the performance and these data are sold to MODACloud platform so it can manage and check the performance, availability, failures, and other applications and deployment environment aspects.

In addition, the PaaSage European project (http://www.paasage.eu/) has Quality of Service assurance as one of its goals. It principally focuses on providing a standardized, open and integrated platform to design and to deploy cloud applications according to the dependencies specifications. Please, note that MODAClouds and PaaSage teams have joined forces to carry out this task. Analogously to MODAClouds (mentioned above), it also requires the developers to adopt a modeling language in order to specify the model of the application. This methodology allows them to develop, configure and deploy applications by

means of an independent layer which provides an abstraction of the infrastructures used. PaaSage is intended to match application requirements against platform characteristics and makes deployment recommendations and dynamic mapping of components to the platform(s) selected for the application instantiation.

The mOSAIC project (http://www.mosaic-cloud.eu/) also shares some of SeaClouds' goals. More precisely, mOSAIC aims at developing an open-source platform that enables applications to negotiate Cloud services as requested by their users. Using a Cloud ontology, applications will be able to specify their service requirements and communicate them to the platform via a vendor agnostic API, so that the resulting applications can be deployed on different IaaS using a sort of mOSAIC virtual machine.
The platform will implement a multi-agent brokering mechanism that will search for services matching the applications' requests, and possibly compose the requested service if no direct lead can be found.
As in other projects, the final result is a platform that supports the user when he develops her code. Our proposal avoids the creation of a middleware platform by allowing the user to select at runtime the platform(s) which better suits his application's preferences and requirements in order to deploy or migrate the corresponding module(s). mOSAIC also plans to support SLA negotiation (with monitoring to detect SLA violations) and the application's life-cycle, but requires developers to adopt the project's API.

The 4CaaSt (http://4caast.morfeo-project.org/) is a project with the highly generic goals of solving the lock-in of IaaS and PaaS services. The 4CaaSt project indeed defines an advanced PaaS platform which will support the optimised and elastic hosting of internet-scale applications. This platform will facilitate programming of rich applications and enable the creation of a business ecosystem where applications coming from different providers can be tailored to different users, mashed up and traded together. This project proposes a marketplace in order to facilitate the development and deployment of the applications which present framework and other elements of run-time dependencies.

The EU FP7 Cloud4SOA project (http://www.cloud4soa.eu) provides an open source interoperable framework for application developers and PaaS providers. We identify several supported providers: Openshift, CloudControl (https://www.cloudcontrol.com/), CloudBees (http://www.cloudbees.com/), CloudFoundry, Amazon WS, Heroku, Google App Engine, Windows Azure, Red Hat, dotCloud (https://www.dotcloud.com/), VMWare.
Cloud4SOA facilitates developers in deployment and lifecycle management and monitoring of their applications on the PaaS, offering the best matches to their computational needs, and ultimately reduces the risks of a vendor lock-in. In this case, Cloud4SOA defines a model for describing an application profile obtaining the knowledge needed to manage applications. Moreover, this profile contains the data necessary to carry out matchmaking among the application components' constraints and find the best match to enable the user to target providers. In the same way as Broker@Cloud, the providers are described based on semantic models, something which is very useful for selecting the likely PaaS services. So, Cloud4SOA aims to be a solution to the interoperability and portability problems of Platform as a Services. In fact, it provides mechanisms to migrate applications through the supported providers (as a long as the new provider supports the application restrictions and dependencies). The monitoring is based on unified metrics, but Cloud4SOA monitors each application separately and it is not able to aggregate monitoring results of multi-component applications.

Cloud standardisation is one of the most active lines in Cloud research. Relevant associations such as IEEE or OASIS are working on standards in order to tackle the interoperability and portability between Cloud platforms. In fact, TOSCA and CAMP are two OASIS standards concentrating efforts in reducing the complexity of deployment and management of cloud applications. These and other Cloud standards, can be found in the Cloud Standards Wiki (http://cloud-standards.org/).

The Open Cloud Computing Interface (http://occi-wg.org/) is emerging standard being developed by the Open Grid Forum comumunity (https://www.ogf.org/). It is intended to provide a common API that defines cloud providers' interaction in an openly and independtly. OCCI defines a protocol for working, this allows the creation of a remote driver

(a remote management entity) to abstract and unify the IaaS service management and orchestration. So, through these drivers the developer could use the services in a portable and interoperable way. Currently, the standard defines the IaaS nature aspect of the providers only. However, it isstill a work in progress, and it will support PaaS and SaaS services in the future.

**Why Relevant**

SeaClouds will be deploying to several PaaS systems, and the preferred architecture is one which allows us to abstract across those systems. Cloud4SOA is particularly useful here. Nevertheless it remains important that we have a good understanding of the underlying PaaS primitives and the differences between them, both so that SeaClouds can do its planning appropriately and so that the deployment and management technically works.

**Our Contribution**

Currently, SeaClouds could use Brooklyn to deploy applications components over the selected providers. However, at the moment Brooklyn does not support PaaS deployments, because the aforementioned technology is based over jclouds and this library does not support the management of these kinds of cloud services. In this regard, SeaClouds could take advantage of Cloud4SOA which allows the management the PaaS services of several different providers (already mentioned). Moreover, Cloud4SOA provides a set of cloud independent monitoring mechanisms which could be very useful to achieve the monitoring and reconfiguration targets of the SeaClouds project. Additionally, Cloud4SOA displays a matchmaking methodology and a provider's modelling, however we are defining our own mechanisms for selecting the target services that better adapt to the application's components. One of SeaClouds goals is to extend the OASIS standard used and currently we use a CAMP declarative layer top to define and deploy the applications over IaaS services. Thus, we will have to wrap and adapt services exposed by Cloud4SOA below the CAMP layer if we want to maintain a unified API. Cloud4SOA shows some examples of migrations of the application's components, which is a goal of the SeaClouds also.

Our contribution will thus be primarily as a consumer of the PaaS APIs, feeding abstractions and characteristcs back to the CAMP and TOSCA standards. However we envision the likely need to extend or update PaaS support in Cloud4SOA, and we recognize the value of extending Brooklyn to support the PaaS systems, all as part of growing the ecosystem.

### 3.4 Service Level Agreement Languages

There are several techniques to describe service level agreements between Cloud Service Provider (CSP) and a Cloud Consumer (CC). QoS is usually defined in static manually managed SLAs.

However, systematic and dynamic languages to represent these contracts are becoming increasingly necessary because they make it possible to interpret these contracts so that, for instance, service providers can be chosen on the fly. SLAs define acceptable service levels to be provided by the CSP to its customers in measurable terms. The ability of a CSP to perform at acceptable levels is consistent among SLAs, but the definition, measurement and enforcement of this performance varies widely among CSPs. A cloud consumer should ensure that CSP performance is clearly specified in all SLAs, and that all such agreements are fully incorporated, either by full text or by reference, into the CSP contract [7].

In the context of SeaCloluds the SLA management framework provides a generic end-to-end solution for SLA definition and operational management embracing multi-clouds services at IaaS and PaaS level. It provides an operational management with SLA composition and decomposition across functional and organizational Cloud domains; and covers the complete SLA and service lifecycle with consistent interlinking of planning and runtime management aspects; and can be applied to a large variety of industrial domains and use cases.

Standardization Needed:

- SLA description language
- SLA evaluation and Penalties

## 3.5 WS-Agreement

WS-Agreement (GFD.107) [8] defines a language and a protocol for advertising the capabilities of service providers and creating agreements based on creational offers, and for monitoring agreement compliance at runtime. This is supported by WS-AgreementNegotiation (OGF), which defines a protocol for automated negotiation of offers, counter offers, and terms of agreements defined under WS-Agreement-based service agreements.

SeaClouds is also willing to explore additional SLA description models as well as negotiation strategies and protocols to enable the an end-to-end SLA creation, the development of monitoring and feedback mechanisms to observe the commitments met by an SLA, and the development of adaption strategies to mitigate the effects of possible SLA infringements.

All these aspects will be analyzed both on a design and implementation level. SeaClouds aims at active participating in relevant standardization bodies and working groups (e.g. ETSI, OGF GRAAP, OGF OCCI or DMTF Open Cloud Standards Incubator) in the evolution of a standardized model of end-to-end Service Level Agreement procedures for Clouds, which will allow precise description of Quality of Service (QoS), an effective governance and audit processes, and lifecycle management of complex systems in heterogeneous and multi-cloud environments.

## 3.6 WSLA

Since SLA monitoring and enforcement become increasingly important in a Web Service environment where enterprises rely on services that may be subscribed dynamically and on demand, WSLA (Specification denied by IBM [9], [10] and [11]), similarly to WS-Agreement, provides a framework for specifying and monitoring Service Level Agreements (mostly defined) for the Web Services.

For economic and practical reasons, IT entreprises wants an automated provisioning process for both the service itself as well as the SLA management system.

WSLA provides an approach to be able to measure and monitor QoS parameters, checks the agreed-upon service levels, and reports violations to the authorized parties involved in the SLA management process.

Although WSLA has been designed for a Web Services environment, it is applicable as well to any inter-domain management scenario such as business process and service management or the management of networks, systems and applications in general.

The WSLA framework consists of a flexible and extensible language based on XML Schema and a run-time architecture comprising several SLA monitoring services, which may be outsourced to third parties to ensure a maximum of objectivity.

### 3.7 SLA*

A further promising language that describes SLA terms and agreement is SLA* [12]. SLA* has beed developed as a generalization and refinement of the web service-specific XML standards: WS-Agreement, WSLA, and WSDL.

SLA* uses much of the WS-agreement constructs. The SLA* model has been developed as part of the SLA@SOI project [12].

The most significant difference between WS-Agreement and SLA* is the dependency on a certain metalanguage or rendering. WS-Agreement is inherently dependent on XML (and XML Schema) as a description language. References within an SLA document in WS-Agreement, for example, are therefore most conveniently realized using XPath [13].

Another important difference between both languages is that WS-Agreement does not offer any domain-specific expressions that could be used to define ranges or constraints. These are left entirely to the user of the specification. SLA*, on the other hand, can be used to directly describe a large number of resources and constraints on those resources [13].

### 3.8 SLAng

The primary aim of SLAng is to provide a language that enables the specification of contractual relationships between consumers and providers, and by that allows for a clear definition of obligations on all involved partners with respect to the provided QoS [14]. The SLAng white paper defines an SLA as an arrangement between a customer and a provider, describing technical and non-technical characteristics of a service, including QoS requirements and the related set of metrics with which the provision of these requirements should be measured [15].

The SLAng syntax is defined using an XML Schema, which favours the integration with existing service description languages. For example, SLAng can be easily combined with WSDL.

The content of an SLA varies depending on the service offered, and incorporates the elements and attributes required for the particular negotiation. In general, an SLA includes:

1. An end-point description of the contractors (e.g., information on customer/provider location and facilities).
2. Contractual statements (e.g., start date, duration of the agreement, charging clauses).
3. Service Level Specifications (SLSs), i.e. the technical QoS description and the associated metrics [15]**Error! Reference source not found.**.

SLAng places emphasis on semantics, providing formal notions of SLA compatibility, monitorability and constrained service behavior. It is, however, targeted at electronic services and provides only a limited set of domain-specific QoS constraints [16].

Even though the language seems to be very well designed, activities on this topic seem to have slowed down or probably stopped, as the last activities on the repository have occurred in 2009.

**Why Relevant**

SeaClouds requires a language for expressing quality of service, to be embedded in the abstract model supplied by the user and passed to the monitoring systems. Thus an understanding of the standards and available systems is very relevant to our work.

**Our Contribution**

SeaClouds will primarily be a consumer of these standards. We do not envision attempting to advance them, although we see some promise in using the YAML lightweight expressive style used by CAMP and TOSCA to formulate the semantics of these standards, and are open to the opportunity to bring the SLA communities and the Application Topology communities closer together.

## 4. Conclusions

This deliverable has described the standardization strategy and plans of the SeaClouds project. These planned activities aims at ensuring the widest possible impact of the SeaClouds concepts, results and outputs.

SeaClouds architecture will be alinghed with two major standards for cloud interoperability: the OASIS' CAMP (Cloud Application Management for Platforms) and TOSCA (Topology and Orchestration Specification for Cloud Applications), promoting them in research and industrial communities.

Additionally, by leveraging these novels OASIS standards, SeaClouds aims at attracting a significant user base (as these standards have a lot of interest but no reference implementations, so far) and advance the standards, ensuring the long-term viability of the benefits delivered by this proposal, i.e. management and monitoring of underlying providers, performance optimization, low impact on the code, formal methods support, flexibility to include new services and react to problems at runtime.

It will also contribute to Cloud Service Level Agreements by providing the tools and required info that the decision makers needs in order to know what to expect and what to be aware of as they evaluate SLAs from their cloud computing providers, enhancing the role that standards play to improve interoperability and comparability across different cloud providers .

The main challenges to be addressed within the SeaClouds project also include the enhancement of existing SLA description models as well as respective negotiation strategies and protocols to enable the an end-to-end Service Level Agreements creation, the development of monitoring and feedback mechanisms to observe the commitments met by an SLA, and the development of adaption strategies to mitigate the effects of possible SLA infringements.

All these aspects will be analyzed both on a design and implementation level. SeaClouds aims at active participating in relevant standardization bodies and working groups (e.g. ETSI, OGF GRAAP, OGF OCCI or DMTF Open Cloud Standards Incubator) in the evolution of a standardized model of end-to-end Service Level Agreement procedures for Clouds, which will allow precise description of Quality of Service (QoS), an effective governance and audit

processes, and lifecycle management of complex systems in heterogeneous and multi-cloud environments.

## 5. References

[1]. Heroku BuildPacks: https://devcenter.heroku.com/articles/buildpacks

[2]. Definition of OpenShift Cartridges: http://openshift.github.io/documentation/oo_cartridge_developers_guide.html

[3]. Features of Apache Stratos: https://cwiki.apache.org/confluence/display/STRATOS/4.0.0+Introducing+Apache+Stratos

[4]. Cloud Foundry documentation page: http://docs.cloudfoundry.org/

[5]. F. Paraiso, N. Haderer, P. Merle, R. Rouvoy, and L. Seinturier, \A federated multi-cloud PaaS infrastructure," in IEEE 5th International Conference on Cloud Computing, 2012, pp. 392-399.

[6]. Description of the IaaS supported by AppScale: https://github.com/AppScale/appscale/wiki

[7]. Creating Effective Cloud Computing Contracts for the Federal Government - Best Practices for Acquiring IT as a Service

[8]. https://www.ogf.org/documents/GFD.107.pdf

[9]. http://domino.watson.ibm.com/library/cyberdig.nsf/1e4115aea78b6e7c85256b360066f0d4/cdedb79080f59ee285256c5900654839?OpenDocument

[10]. http://clip.dia.fi.upm.es/Projects/S-CUBE/papers/keller03:wsla_framework.pdf

[11]. http://domino.watson.ibm.com/library/cyberdig.nsf/papers/CDEDB79080F59EE285256C5900654839/$File/RC22456.pdf

[12]. Keven T. Kearney and Francesco Torelli and Constantinos Kotsokalis, “SLA*: An Abstract Syntax for Service Level Agreements,” 2010.

[13]. Peter Chronz and Philipp Wieder, “Integrating WS-Agreement with a Framework for Service-Oriented Infrastructures,” 2010.

[14]. Optimus, “Analysis of Existing solutions and Requirements for SLAs,” 2009.

[15]. D.Davide Lamanna and James Skene and Wolfgang Emmerich, “SLAng: A Language for DefiningService Level Agreements,” 2003.

[16]. Keven T. Kearney and Francesco Torelli and Constantinos Kotsokalis, “SLA*: An Abstract Syntax for Service Level Agreements,” 2010.