SeaClouds Project
D4.4 Dynamic QoS verification and SLA management approach

Project Acronym	SeaClouds
Project Title	Seamless adaptive multi-cloud management of service-based applications
Call identifier	FP7-ICT-2012-10
Grant agreement no.	Collaborative Project
Start Date	1st October 2013
Ending Date	31st March 2016
Work Package	WP4. SeaClouds run-time environment
Deliverable code	D4.4
Deliverable Title	Dynamic QoS verification and SLA management approach
Nature	Report
Dissemination Level	Public
Due Date:	M18
Submission Date:	03rd April 2015
Version:	1.0
Status	Final
Author(s):	Mattia Buccarella (UPI), Marc Oriol (UPI), Dionysis Athanasopoulos (POLIMI), Javier Cubo (UMA), Roman Sosa (ATOS), Antonio Brogi (UPI)
Reviewer(s)	Elisabetta Di Nitto (POLIMI), Christian Tismer (NURO)

Dissemination Level

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

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	09/03/15	First Table of Contents	Antonio Brogi, Javi Cubo, Francesco D'andria, Elisabetta Di Nitto, Roman Sosa
0.2	18/03/15	First contributions to the document	Marc Oriol, Mattia Buccarella, Dionysis Athanasopoulos, Javier Cubo
0.3	23/03/15	Next contributions in some sections, and checking other sections	Roman Sosa, Dionysis Athanasopoulos, Javier Cubo
0.4	01/04/15	Final contributions in the document	Mattia Buccarella, Marc Oriol, Javier Cubo, Roman Sosa
0.5	02/04/15	Reviews	Elisabetta Di Nitto, Christian Tismer
1.0	03/04/15	Final version	Marc Oriol, Mattia Buccarela

Table of Contents

List of Figures

List of Tables

Executive summary

This deliverable describes the dynamic Quality of Service (QoS) verification and Service Level Agreement (SLA) management performed by SeaClouds.

In this document, we firstly identify and define a list of metrics that are commonly used to represent the properties and the requirements of the services involved in a cloud-based application. Secondly, we specify the syntax of the monitoring rules, which includes the metrics, the entities to be monitored, how data should be aggregated and the monitoring actions. From these rules, the monitor is configured and the monitoring process is able to assess if the QoS is met. Then we describe the SLA Assessment, which is connected to the monitor, receives QoS violations, and performs dynamic verification of the SLAs considering also the Quality of Business (QoB).

Finally, we describe within the architecture of SeaClouds, how the different components responsible of the aforementioned activities are orchestrated and how repairing or replanning is conducted if a violation occurs.

1. Introduction

SeaClouds aims at managing multi-cloud distributed applications. To this aim, it is essential the monitoring, troubleshooting and assurance of the Quality of Service (QoS) at runtime. SeaClouds shall extend the concept of monitoring services from the one-cloud to the multi-cloud perspective, which requires the proper support and implementation. Furthermore, if a violation occurs, repairing or replanning actions should be conducted to restore the QoS.

In this deliverable, we first identify different metrics that are suitable to represent the properties of a multi-cloud application. These metrics have been derived by analyzing the case studies described in D6.1 [1], taking into account the most common and important characteristics, resources and needs that are involved in such scenarios.

These metrics are part of the monitoring rules. The monitoring rules are XML documents, which includes the metrics, the entities to be monitored, how data should be aggregated and the monitoring actions. These rules are used to initialize the monitor, which measures the values of the desired metrics. To do so, the monitor is composed of different data collectors, which are components in charge of collecting data concerning the monitored resources.

Based on the monitoring data, SeaClouds evaluates the monitoring policies. If a violation occurs, SeaClouds tries to fix it by repairing the current deployment plan, or, replanning a new deployment plan.

We also address the Service Level Agreement (SLA) Assessment and describe how SeaClouds perform Quality of Business (QoB) analysis. That is, a long-term analysis on metrics that impact on the business of the application. This QoB evaluation relies on the QoS evaluation: it uses the detected QoS violations from the monitor to assess the QoB constraints, playing a role in a high-level perspective.

Finally, we describe within the architecture of SeaClouds, how the different components responsible of the aforementioned activities are orchestrated.

1.1. Structure of this document

The document is structured as follows: in section 2, we specify the metrics and the monitoring rules. Section 3 describes the monitoring process in detail. Section 4 describes the SLAs assessment. Section 5 details how the orchestration works. Finally, section 6 provides the conclusions for this document.

1.2 Glossary of Acronyms

Acronym	Definition
AAM	Abstract Application Model
ADP	Abstract Deployment Plan
DAM	Deployable Application Model
IaaS	Infrastructure-as-a-Service
PaaS	Platform-as-a-Service
QoB	Quality of Business
QoE	Quality of Experience
QoS	Quality of Service
RTT	Round-Trip Time
SaaS	Software-as-a-Service
SLA	Service Level Agreement
SLO	Service Level Objective
XML	eXtensible Markup Language

Table 1. Acronyms.

2. Specification of Metrics and monitoring rules

In this section, we describe the metrics used to represent the service properties and requirements of the cloud-based application in SeaClouds. These metrics are used in SeaClouds with the purpose of suitably give a quantitative measure of the properties of the cloud services and the cloud application. SeaClouds measures these metrics with the following objectives: (1) detect at runtime any violation of the SLA(s); and (2) provide runtime information to the optimization process if replanning is triggered.

Specification of metrics used in SLAs
SeaClouds is required to monitor the metrics specified in the SLAs. As it will be described later in section 4.1. There are two levels of SLAs:

- Customer - Application Provider SLA (C-AP SLA)
- Application Provider - Cloud Provider SLA (AP-CP SLA)

The C-AP SLA is the SLA that the customer has with the Application Provider (i.e. SLA of the Application), whereas the AP-CP SLA is the SLA that the Application Provider has with the cloud services (e.g. Amazon EC2).

For such a reason, one property is assigned one scope, i.e., it can be associated to one or more specific application modules individually or to the application seen as a whole (e.g., composition of modules). For this reason, the metrics used to measure these properties are also divided in two categories, with the same logic: module metrics and application-wide metrics.

Specification of metrics used for the optimization process
The optimization process requires QoS information in order to generate a correct deployment plan. If a violation on the SLA(s) occur, a replanning is triggered, and results of the monitored metrics are used to generate an accurate plan.

SeaClouds support of metrics

In order to define the metrics that the monitor should support, we may distinguish between compulsory and non-compulsory metrics.

- *Compulsory metrics* are those metrics that may be included in the SLAs as they may be required by the user. SeaClouds should provide support for these type of metrics in order to detect and recover from possible SLA violations. It is worth to mention, that the list of *compulsory metrics* is not intended to define what the SLAs should include, but what SeaClouds should support based on the common properties defined in SLAs in the field of cloud computing. On the other hand, it is not intended to be a complete list of all possible metrics in an SLA, but defines the support that SeaClouds is currently aiming at.
- *Non-compulsory metrics* are those metrics that are not strictly required, as they are not part of SLAs, but can be useful during the replanning process as an augmented source of information. This information is used by heuristic algorithms to infer, in a more precise way, the overall performance of a new deployment plan.

In the following section, we define the list of metrics, using the template shown in Table 2.

Metric Name:	*name of the metric*
Description:	*short description*
Scope:	*Identifies whether it is an application-wide or module metric*
Included in SLAs:	*Identifies if the metric is used in SLAs (Yes/No)*
Optimizer Criterion:	*Identifies if the metric is useful for the optimizer in the replanning phase (Yes/No)*
SeaClouds support:	*Identifies if the metric is compulsory based on its relevance for the optimizer and SLA assessment (Compulsory/Optional)*

Table 2. Template to define the metrics

2.2. List of metrics

In this section, we describe the list of elicited metrics that will be used as part of the *monitoring rules*. These metrics belong to Performance and reliability characteristics [2], which are common characteristics in the field of cloud computing and of interest for the SeaClouds project. It is worth mentioning that the following list is not intended to be the final, and some changes might occur in subsequent phases of the project (e.g. new required metrics can be identified).

Metric Name:	Response time
Description:	It is the expected response time of the deployed application/module to fulfill its functionalities. This response time does not include the network latency between the end-user and the application. But, it is the time interval between the instant in which the request is received by the application/module and the instant in which the corresponding output is produced.
Scope:	application-wide, module
Included in SLAs:	Yes

Optimizer Criterion:	Yes
SeaClouds Support:	Compulsory

Metric Name:	CPU utilization
Description:	It is the percentage showing the load of the CPU hosting the module.
Scope:	Module
Included in SLAs:	Yes
Optimizer Criterion:	Yes
SeaClouds Support:	Compulsory

Metric Name:	Memory utilization
Description:	It is the amount of memory, expressed in megabytes (MB), utilized by a certain machine on the cloud. This metric has both module and application-wide scopes. While the module scope is the most trivial, the application-wide can be expressed by aggregation (i.e., sum) of each machine value.
Scope:	Application-wide, Module
Included in SLAs:	Yes
Optimizer Criterion:	Yes
SeaClouds Support:	Compulsory

Metric Name:	Storage utilization
Description:	It is the amount of disk quota utilized on the cloud. It is expressed in gigabyte (GB). This metric has both module and application-wide scopes. While the module scope is the most trivial, the application-wide can be expressed by aggregation (i.e., sum) of each module values.
Scope:	Application-wide, module
Included in SLAs:	Yes
Optimizer Criterion:	Yes
SeaClouds Support:	Compulsory

Metric Name:	Bandwidth
Description:	The bandwidth provides information about the amount of data that can be handled by the cloud Internet connection. Structurally, this property is represented by means of two values: the *uplink bandwidth* and the *downlink bandwidth*. In our case, the Mbps (Megabit per second) is a suitable unit of measure to represent these two values. With respect to the provider, the uplink bandwidth is the bit rate that the endpoint is able to push towards the outside, while the downlink bandwidth is the bit rate that the endpoint is able to receive from the outside.
Scope:	Module
Included in SLAs:	No
Optimizer Criterion:	Yes
SeaClouds Support:	Optional

Metric Name:	Network latency
Description:	Given two endpoints (e.g., two application nodes, each deployed on its own cloud, communicating with one another) this property is measured by the time, expressed in milliseconds (ms), required by the transmission of one unit of traffic, to reach the recipient, after the dispatch from the sender, and come back. This trip delay is known as *round-trip time* (RTT). It is very important to point out that the RTT is a property that makes sense only when measured with respect to *couples of endpoints*. To measure this value, there must be a clear graph representing all the possible connections involved in a multi-cloud application (e.g., communication relationships across application modules), so that it can be evaluated across two nodes, say N1 and N2, that have at least one path allowing a transmission unit to travel from N1 (N2) to N2 (N1).
Scope:	Application-wide
Included in SLAs:	No
Optimizer Criterion:	Yes
SeaClouds Support:	Optional

Metric Name:	Throughput
Description:	The throughput of a given connection link is the link capacity that is actually used. As well as the bandwidth, this value is measured in Mbps and it is provided with respect to a certain time interval (e.g., 1h).
Scope:	Module
Included in SLAs:	No
Optimizer Criterion:	Yes
SeaClouds Support:	Optional

Metric Name:	Availability
Description:	The availability of the application in a given time interval. This availability is expressed as a percentage value and it should be provided not only with instant granularity, but also for the past times. Availability in past day, availability in past week and availability in past month would be a great level of detail for the evaluation of a cloud to be considered. Each of this value is also a percentage averaged among the instant percentages detected during the past monitorings.
Scope:	Application-wide, Module
Included in SLAs:	Yes
Optimizer Criterion:	Yes
SeaClouds Support:	Compulsory

2.3. From metrics to monitoring rules

To measure the values of a metric, which corresponds to a QoS property of an application as a whole or to an application module, we need a formal way to specify: (i) which entity on the cloud should be monitored, (ii) which kind of data should be collected, (iii) how these data should be aggregated, and possibly, (iv) what monitoring actions should be performed under what conditions. Conceptually, all this information defines a monitoring rule.

Since there is no mature model of monitoring rule based on TOSCA standard, we properly adapt the model of monitoring rule, which has already been proposed in MODAClouds project [3] in SeaClouds.

Firstly, we present the model of monitoring rule, specified in XML schema of Table 3 and explained in subsection 2.3.1. Secondly, we describe the practical usage of this model in measuring the values of the QoS metrics of the performance and the availability of an application in subsection 2.3.2.

2.3.1. Model of Monitoring Rule

Overall, a monitoring rule specifies a set of monitoring entities, the kind of collected data, the aggregation way of collected data, and monitoring actions.

Monitoring Target

In particular, to specify a monitoring entity, a rule includes a set of *monitoring targets* (line 3 in Table 1). Each target is characterized by its *class* and its *type* (lines 38-41 in Table 1). The class of a monitoring target specifies the kind of the monitored entity, which is related to the infrastructure layer of a cloud provider (e.g. monitoring the virtual machines of a cloud for measuring its availability), or to the application layer of the cloud (e.g. monitoring the performance of a method of a user-provided application). The type of a monitoring target specifies the particular instance of an entity that is monitored. It means that a concrete entity is monitored, which can be a specific virtual machine at the infrastructure layer or an instance of the abstract description of an application.

Monitoring Metric

To specify the kind of the monitoring data, a rule includes a *monitoring metric* (line 4 in Table 3). A metric corresponds to the kind of measurement that will be performed on the monitoring target (lines 42-53 in Table 3). As described in section 2.1, a metric may be related to the measurement of the values of a consumed resource in the monitoring target (e.g., CPU consumption), or it may be related to a non-functional property of the monitored application (e.g., response time). A metric is characterized by a set of parameters (line 44 in Table 3), which specify the configuration settings of the software component that collects data/values for this metric. These settings may differ from data collector to data collector. For instance, a data collector, which measures CPU consumption, needs the sampling period. On the contrary, a data collector, which measures SQL query statistics, needs the authentication credentials to a database server.

Aggregation Metric

In case an aggregation of monitoring data is required, a rule contains an *aggregation metric* (line 5 in Table 3). Note that the provision of an aggregation metric is optional. To perform an aggregation, an *aggregation function* is required (line 65 in Table 3). Such a function can be the average, percentile, max, min, and so on of monitoring data. An aggregation metric optionally involves a *grouping class* (line 64 in Table 3) and a set of parameters (line 56 in Table 3). A grouping class specifies the monitoring entity based on which the aggregation will be performed. For instance, if the grouping class is virtual machine then the monitoring data will be aggregated per virtual machine. A parameter refers to restrictions on the value of an aggregation function. For instance, if the aggregation function is the percentile, then a parameter can be the lowest value of the percentile, over which the aggregation will be performed.

Monitoring Action

Regarding what monitoring action and under which condition it should be performed, a rule includes two parts, a *condition* (line 6 in Table 3) and a set of *actions* (line 7 in Table

3). Since the monitoring is not necessarily accompanied by actions, these parts of a rule are also optional. A condition (lines 33-37 in Table 3) is a logical expression that is applied on the value of the aggregation metric, if the latter exists. Otherwise, the condition is applied on the monitoring data values. In case the condition value is true, the set of actions is executed. An action can be the execution of a RESTful invocation, the print of the values of multiple metrics, or even the removal of the rule (lines 79-89 in Table 3).

A monitoring rule is further characterized by the time interval (in seconds) between its two consecutive evaluations, *timeStep* (line 13 in Table 3). Also, a rule is characterized by the time range (in seconds), *timeWindow* (line 14 in Table 1), in which monitoring data are aggregated at every *timeStep*. Another attribute of a rule is the flag, *startEnabled* (line 12 in Table 3), which specifies whether the rule evaluation should start.

Line	XML Schema
1	<xs:complexType name="**monitoringRule**">
2	<xs:sequence>
3	<xs:element name="monitoredTargets" type="mo:monitoredTargets"/>
4	<xs:element name="collectedMetric" type="mo:collectedMetric"/>
5	<xs:element name="metricAggregation" type="mo:monitoringMetricAggregation" minOccurs="0"/>
6	<xs:element name="condition" type="mo:condition" minOccurs="0"/>
7	<xs:element name="actions" type="mo:actions"/>
8	</xs:sequence>
9	<xs:attribute name="id" type="xs:string" use="required"/>
10	<xs:attribute name="label" type="xs:string"/>
11	<xs:attribute name="relatedQosConstraintId" type="xs:string"/>
12	<xs:attribute name="startEnabled" type="xs:boolean"/>
13	<xs:attribute name="timeStep" type="xs:string" use="required"/>
14	<xs:attribute name="timeWindow" type="xs:string" use="required"/>
15	</xs:complexType>
16	<xs:simpleType name="**probability**">
17	<xs:restriction base="xs:float">
18	<xs:maxInclusive value="1"/>
19	<xs:minInclusive value="0"/>
29	</xs:restriction>
21	</xs:simpleType>
22	<xs:complexType name="**monitoredTargets**">
23	<xs:sequence>
24	<xs:element name="monitoredTarget" type="mo:monitoredTarget" maxOccurs="unbounded">
25	<xs:annotation>
26	<xs:appinfo>
27	<jaxb:property name="monitoredTargets"/>
28	</xs:appinfo>
29	</xs:annotation>
30	</xs:element>
31	</xs:sequence>
32	</xs:complexType>

```
<xs:complexType name="condition">
  <xs:simpleContent>
    <xs:extension base="xs:string"/>
  </xs:simpleContent>
</xs:complexType>
```

```
<xs:complexType name="monitoredTarget">
  <xs:attribute name="type" type="xs:string" use="optional"/>
  <xs:attribute name="class" type="xs:string" use="required"/>
</xs:complexType>
```

```
<xs:complexType name="collectedMetric">
  <xs:sequence>
    <xs:element name="parameter" type="pa:parameter" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:appinfo>
          <jaxb:property name="parameters"/>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="metricName" type="xs:string" use="required"/>
</xs:complexType>
```

```
<xs:complexType name="monitoringMetricAggregation">
  <xs:sequence>
  <xs:element name="parameter" type="pa:parameter" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:appinfo>
          <jaxb:property name="parameters"/>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="groupingClass" type="xs:string"/>
  <xs:attribute name="aggregateFunction" type="xs:string" use="required"/>
</xs:complexType>
```

```
<xs:complexType name="actions">
  <xs:sequence>
    <xs:element name="action" type="mo:action" maxOccurs="unbounded">
      <xs:annotation>
        <xs:appinfo>
          <jaxb:property name="actions"/>
       </xs:appinfo>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

```
<xs:complexType name="action">
  <xs:sequence>
    <xs:element name="parameter" type="pa:parameter" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:appinfo>
          <jaxb:property name="parameters"/>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required"/>
</xs:complexType>
```

Table 3. The XML specification of a monitoring rule.

2.3.2. Using the Model of Monitoring Rule in Measuring Metric Values

In a first place, except for the specification of the syntax of a monitoring rule, a common glossary of metric names is required. This glossary contains the performance and availability metrics that are commonly used by the Planner (to specify the monitoring rules), the Monitor (to execute the monitoring rules), and the Deployer (to install the data collectors, required for gathering the metric values).

Based on the list of metrics defined in section 2.2, we provide in Table 4 the commonly accepted terms that we use on the different components to refer to them.

Metric	**Agreed term**
Compulsory metrics	
Response time	ResponseTime
CPU utilization	CPUUtilization
Memory utilization	MemoryUtilization
Storage Utilization	Queries
Availability	Availability
Optional metrics	
Bandwidth	Bandwidth
Network Latency	Latency
Throughput	Throughput

Table 4. Glossary of the metric terms agreed for SeaClouds.

In a second place, instances of monitoring rules, suitable for measuring the aforementioned metrics, can be formed by following the monitoring ontology [4], used by the MODAClouds monitor [3]. Taking an example, assume that we would like to calculate the average CPU consumed by an application per cloud virtual machine. To this end, MODAClouds monitoring ontology defines the class *VM* that corresponds to a cloud virtual machine. Thus, we should write a monitoring rule that specifies the class *VM* (*monitoredTarget class="VM"*) as a monitoring target, the *Average* as an aggregation function (*aggregationFunction name="Average"*), and the class *VM* as a grouping class, based on which the aggregation will be performed (*groupingClass= "VM"*). In this way, as required, the average percentage of CPU utilization over multiple virtual machines is calculated.

3. The monitoring process

The SeaClouds platform helps end-user in the design, the deployment, the management, and the reconfiguration of complex applications across clouds. In other words, the SeaClouds platform supports end-user at the design and the runtime of the lifecycle of an application. The monitoring part of the SeaClouds platform supports the runtime part of the application lifecycle. In this section, we focus on describing the runtime process followed by the SeaClouds monitoring. The monitoring process is depicted in the sequence diagram of Figure 1 and is further described as follows.

In particular, the monitoring process can be divided into two main parts. The first part concerns the initiation and the initialization of the Monitor. In a nutshell, this part includes the starting up of the internal components of the Monitor and its initialization with the necessary information (e.g. monitoring rules) for acting about the monitoring data. The second part includes the actions that the Monitor performs when the rules are violated.

Concerning the first part, by the time an application has been designed and its deployment plan has been provided to the Deployer, the latter initiates the Monitor. In this way, the internal servers of the Monitor (Monitoring Manager, Knowledge Base, and Data Analyzer), which are able to collect data from the monitored cloud machines and to analyze them, are started up. After the initiation of the Monitor, the Deployer provides to the Monitor the description of the current deployment plan for the current application and the monitoring policies in the form of monitoring rules.

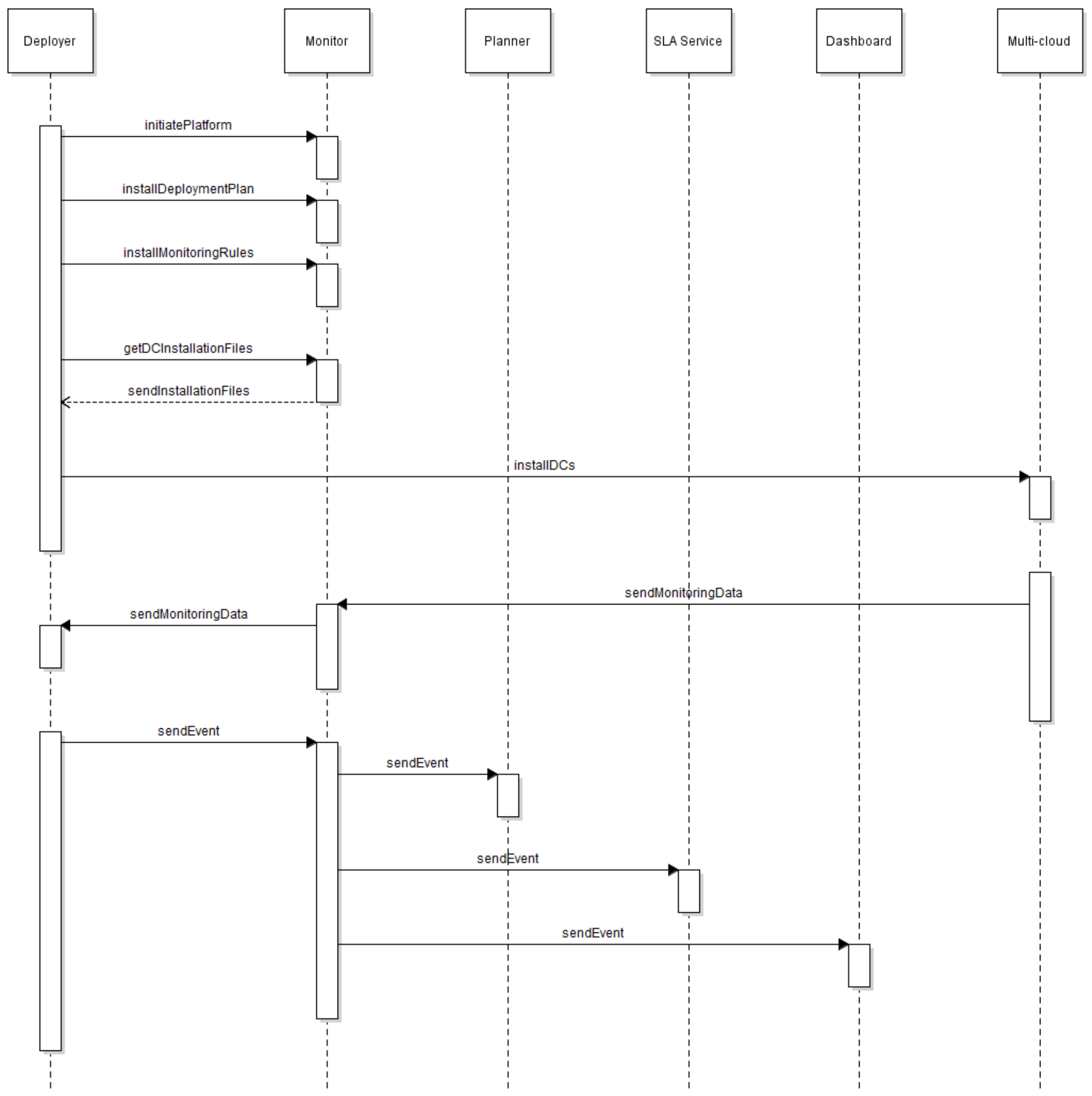

Figure 1. The sequence diagram that specifies the monitoring process.

The first part of the monitoring activities is finalized when the Deployer asks and retrieves from the Monitor the installation files of the data collectors that measure the values of the desired infrastructure-layer metrics. The latter metrics are specified in the monitoring rules.

Having acquired the necessary data collectors, that is, the components in charge of collecting data concerning the monitored resources, the Deployer continues with their installation on the cloud machines, which are specified in the current deployment plan. The installation of the data collectors means that the Monitor enters into the second part of the monitoring process. In this part, the data collectors send data to the Monitor. These raw monitoring data are also acquired by the Deployer.

Based on the monitoring data, the Deployer evaluates the monitoring policies. If a violation happens, then the Deployer tries to fix this by repairing the deployment plan. If a repairing activity is not possible, then the Deployer sends a notification event to the

Monitor informing the latter that the replanning of the current application is required. In this case, the Monitor informs about the replanning necessity the subscripted components to the monitoring rule, which corresponds to violated policy. This process will be described more in detail in Section 5.

Note that the methods invoked on the monitoring platform in the sequence diagram of Figure 1, have been specified in the API of the SeaClouds monitoring platform in the deliverable D4.5 [5].

4. Service Level Agreements assessment

The SLA Service enables the Service Level Agreements (SLA) management of business oriented policies. The main responsibilities of the SLA service are: generating and storing WS-Agreement templates and agreements, and assessing that all the agreements (SLA guarantees) are respected by evaluating the business rules.

4.1 The two SLA levels

In SeaClouds, two levels of SLA have been identified (see Deliverable D2.4 [6], depending of the involved parties of the agreement:

- Customer - Application Provider SLA (C-AP SLA)
- Application Provider - Cloud Provider SLA (AP-CP SLA)

The Customer-Application Provider level describes the service offered by the Application Provider to their users. In this SLA level, there is one template per full application.

The guarantee terms in this Customer-Application Provider SLA should only watch observable metrics by the customer. The purpose is to measure the Quality of Experience (QoE) that the user perceives when he/she is using the application; in the case of SeaClouds, these metrics are, for example, availability and response time. The application provider have the possibility to add business values to the guarantee terms, in order to describe the penalties that the application provider will incur if the term is violated. A possible penalty is a discount in the monthly fee.

The Application Provider-Cloud Provider SLA level describes the service offered by the cloud provider to the Application Provider. For example, the Amazon offers are described in a set of templates, where the QoS assured and the corresponding penalties if not fulfilled are stored. These templates are filled with the SLA offers advertised by each cloud provider.

At this SLA level, there is one agreement for each cloud provider where the application is deployed.

The application designer can enrich the agreements based on the cloud provider templates with business values. Due to the fact that the cloud provider enforces its own SLA, and therefore, SeaClouds can not impose any penalty to the cloud provider, the actions that make sense to be specified here are unilateral actions. A possible action of this type is a migration of the modules in the affected cloud provider to another cloud provider. In SeaClouds, this is achieved with a replanning trigger generated by the SLA Service.

This SLA level performs a second assessment of the actual SLA enforced by the cloud provider, although monitored by the Monitor component. In this way, the application provider can check at the end of the billing cycle the QoS violations incurred by the cloud provider, and verify that the corresponding discounts have been applied.

An external accounting/billing component may interact with the SLA Service in order to obtain the business penalties that have occurred.

4.2 The relation with the Monitor

The SLA Service has a strong dependency with the Monitor: while the Monitor performs the QoS Assessment, the SLA Service performs a QoB (Quality of Business) Assessment. This QoB evaluation relies on the QoS evaluation: it uses the detected QoS violations to assess the QoB constraints, playing a role in a high-level perspective.
We can consider the QoS evaluation as a short-term evaluation. This means that when a condition is detected, an immediate action is expected. The action may be a repairing or a replanning. If the action solves the issue, no more action is delivered.

On the other hand, QoB evaluation is a long-term analysis on metrics that impact on the business of the application, and the business actions to apply in case of violation. For example, if a violation of Response Time has been detected by the Monitor, the deployer will try to add a virtual machine in order to solve the situation. But consider a QoB policy defining that when the Response Time is violated 3 times in a hour, we are not interested in that cloud provider anymore, and force a replanning, even when the issue is solved by a repairing. QoB policies also can define penalizations (like discounts) to the provider of the agreement if the QoB constraint is not fulfilled.

In order to perform its task, the SLA Service needs to be subscribed to the Monitor component to receive the QoS violations observed in an agreement.

4.3 From the user requirements to the SLA

The user requirements are expressed in the TOSCA AAM. They will be translated into WS-Agreement. Thus, the SLA Service is an implementation of the WS-Agreement specification [7], which defines schemas for SLA Templates and SLA Agreements. According to WS-Agreement:

- A template is a document used by the service provider to advertise the types of offers it is willing to accept.
- An agreement defines a dynamically-established and dynamically-managed relationship between a provider and a customer, where the object of this relationship is the delivery of a service by the provider to the customer.

A template or agreement contains functional and nonfunctional terms that describes the service being delivered. In SeaClouds, we are mostly interested in non-functional terms (Guarantee Terms), comprised of a Service Level Objective (SLO), defined as a constraint on a metric, and a list of business values describing the result of not fulfilling the objective.

The templates may be used as a base to create the actual agreements. Also, an agreement may contain additional terms not found in a template. For example, in SeaClouds, the agreements will contain Quality of Business (QoB) policies specified by the application designer, but not specified in a cloud provider template.
A QoB rule is defined like this:

- A constraint over a metric provided by sensors (e.g. runtime < 2000ms). A non fulfilled constraint is considered a QoS violation.
- The business action that takes place in the case of violation. This action may also be defined inside a time window. For example, 3 violations of a constraint in a day is penalized with a discount, and 5 violations in a day is penalized with a trial of a service during one month.

Let us consider a simple web application deployed on Amazon. We want to assure a Response Time of 500 ms to the user, but impose 200 ms to the cloud provider. If the constraint is violated three times in an hour, we want to migrate. The corresponding SLA agreements for this scenario are shown in Figure 2. For readability purposes, we expressed the SLAs in YAML, although the formal specification is in WS-Agreement.

Application Provider - Cloud Provider
```
Context:
        Provider: Amazon
        Consumer: Seaclouds
        Service: Compute
        Resources: webapplication
Guarantee Terms:
        -Availability:    # from amazon template
                Constraint: uptime_month > 0.995
        -ResponseTime:
                Constraint: response_time < 200
        BusinessValues:
                -(5, hour): migrate
```
User - Application Provider
```
Context:
        Provider: Seaclouds
        Consumer: user
        Service: gold
Guarantee Terms:
        -ResponseTime:
        Constraint: response_time < 500
        BusinessValues:
        -(5, hour): discount 5% hour
        -(10, hour): discount 10% hour
```

Figure 2. SLAs in the Web Application in YAML

5. Dynamic orchestration and verification of the QoS and SLA

Once an application has been deployed, its management is accomplished by the Deployer, and the Monitor is checking the status of possible violations, connected to the SLA Service, as explained in Section 5.1. Whether a violation occurs, then a reconfiguration process is required, which will be described in Section 5.2. Note an initial description of this process is given in Deliverable 4.3 [8]. Revisions of this process may be presented in the next deliverables of WP4, towards improving the performance of the process, specifically in Deliverable D4.6 [9].

5.1 The Deployer as orchestrator of the Monitoring and SLA assesment

In order to understand how the deployer acts like an orchestrator of the monitoring and SLA assessment, we present and describe the interaction flow of SeaClouds in this section, following the sequence diagram presented in Figure 3. This figure represents the necessary steps to carry out an **application deployment** from the initial stage where the Application Developer (end-user) provides the Abstract Application Model consisting of the Module Profile and the Topology representing the connections among the modules of the cloud application to be deployed (other elements as the SLA restrictions and policies are considered by SeaClouds), as described in detail in Deliverable D3.1 [10], related to the design-time. And the corresponding information to the deployment, monitoring and reconfiguration of the application, related to the run-time, please, refer to Deliverables D4.1 [11] and D4.3 [8]. A brief explanation of this process is given in the following.

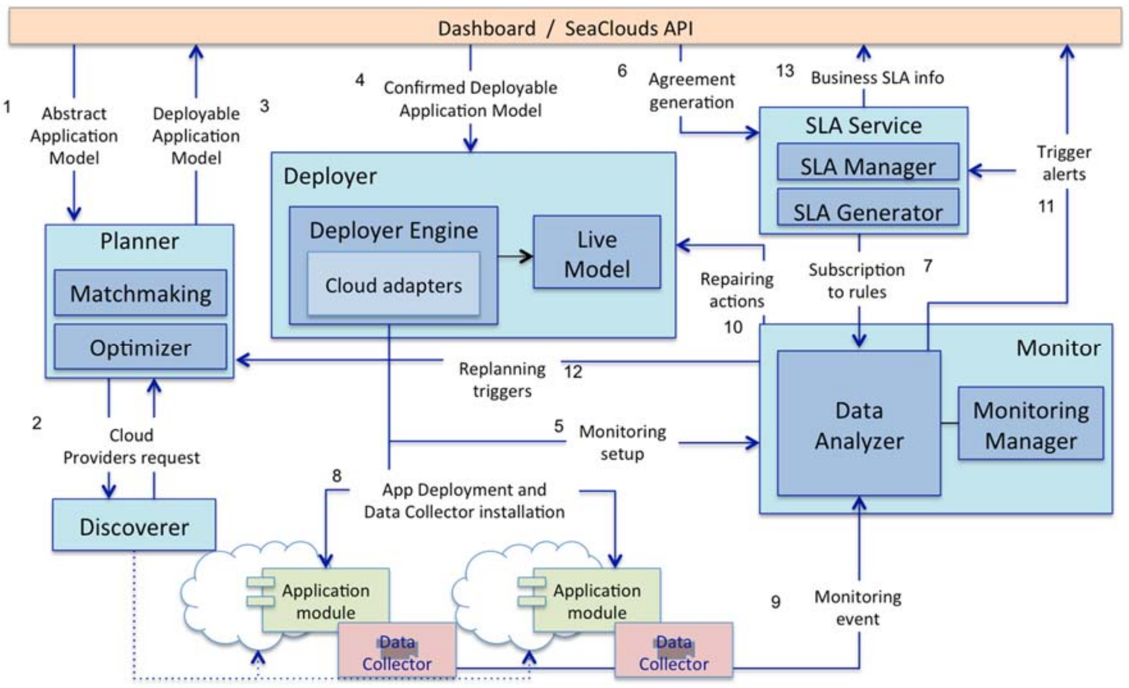

Figure 3. Interaction flow between the SeaClouds components

After the Abstract Application Model (AAM) has been specified (step 1 in Figure 3), SeaClouds kicks off the Discovering and Planning stage. Once the cloud providers have been discovered (step 2), the Planner acts with two subprocesses: Matchmaking and Optimizer (described in D3.2 [10]).

The Planner generates a Deployable Application Model (DAM), which specifies the concrete cloud services used to distribute the application (step 3), using an internal and intermediate model, the Abstract Deployment Plan (ADP), between the AAM and the DAM.

Then is the time in which the **Deployer component** acts, as detailed in Deliverable D4.1 [11]. The Deployer executes the confirmed DAM (step 4), while the monitoring is configured, by means of the Deployer, with the corresponding monitoring rules taken from the user requirements (step 5) and the SLA agreements are initialized with the user inputs (step 6). Also, the SLA service is subscribed to rules or alerts in connection with the Monitor, enforcing the policies of the agreements (step 7). The Deployer also allows the deployment of the application’s modules over heterogeneous IaaS and PaaS (step 8, which can be executed together with step 5), and a Live Application Model, which tracks the dynamic evolution of the deployment and management of the application modules themselves.

Once the application is deployed, the Deployer manages it and instruments the Monitor and SLA assessment. In particular, the Deployer installs Data Collectors in the cloud machine(s), in which the application modules have been deployed. A Data Collector component gathers raw monitoring data and pushes them to the Monitor (step 9). The latter component interacts with the Deployer and the SLA service to jointly manage violations of properties, QoS and QoB. A Live Application Model maintains a track of the dynamic evolution of the deployment and management of the application modules. Whether a violation issue occurs, and it can be fixed without replanning, then the Monitor and the Deployer interact to repair the issue (step 10). Otherwise, then the Monitor interacts with the Planner (steps 11, 12). In this case, the Planner generates a new plan, a replanning action, which will be executed by the Deployer. Business SLA info is exposed by means of the Dashboard (step 13).

5.2 From monitoring results to repairing and replanning

Monitoring and managing applications is an arduous task. An application management solution should consider multiple runtime metrics to understand effectively and possibly efficiently the status of the application in order to manage it correctly, using the monitoring results in the corresponding reconfiguration action.

Depending on the step of the management process, the monitoring results include different kind of information.

- During the evaluation of the management policies by the Deployer, the monitoring results include raw monitoring data, used by the Deployer for performing this evaluation. The monitoring data are collected based on the metrics specified in the monitoring rules.
- In case a reconfiguration plan (esp., replanning) is decided, the monitoring results include the sending of monitoring events to the external SeaClouds components (Planner, Dashboard, and SLA Service) about the necessity of replanning. This kind of monitoring result is realized by the actions, also specified in the monitoring rules.

In SeaClouds, “the source of truth” is the Deployable Application Model (DAM), the concrete plan that is executed by the Deployer Engine. The Deployer Engine, in fact, executes DAM concrete plans produced by the planning stage. In the planning stage, SeaClouds assembles a plan that contains the modules of the application and the topology, the concrete services that execute the modules, and the policies to fulfill user requirements, like SLA or cost-constraints. Just to give a very basic example, a SeaClouds user can ask to deploy a 3-tier web application in EU ensuring that its response time will always be under 300 ms but without spending more than x euros/month. This (simple) example shows the complexity of managing applications: those are reasonable constraints, but the SeaClouds platform will have to translate this high-level constraints into a DAM concrete plan that the platform can run and manage. As already proposed in Deliverable D4.3 Design of the runtime reconfiguration process [8], after an alert is triggered, as monitoring result, we differentiate two reconfiguration types: Repairing and replanning.

Repairing occurs when the violation can be fixed without the need of generating a new plan, because it was previously considered by the user or even it is dynamically ordered by the user. In this strategy, the Deployer and the Monitor is directly connected to check the actions which could be fixed.

Repairing reconfiguration is based on the management of deployment resources, which allows SeaClouds to adjust the deployed application according to the runtime information and related monitoring rules (previously defined by the user, or even requested at runtime), using the Deployer Engine effectors. Then, it involves dynamic changes or fixed fails of some components or the entire application. The applicable scenarios mainly include replacing/restarting a failed component, scaling to meet the demand, and applying a follow-the-sun policy.

Replanning reconfiguration will try to handle the cases that cannot be solved by repairing. Replanning is a more complex process than repairing since it needs to involve the Planner to generate a new plan. It needs to modify the plan specified in the DAM, that describes the distribution of the application modules, and do a redeploying. Thus, replanning cannot be completed independently by only the Deployer Engine, but also needs the work of planner to update the DAM which may contain new distribution of the application modules. Replanning reconfiguration will be more complicated also because migration may happen in this process.

A main reason of triggering replanning is a QoS violation. In this case, the Monitor forwards the event of such a violation to interested components, subscribed to the Monitor. Another reason of replanning is a QoB violation. In particular, the SLA Service component is listening for violations that impact on the business of an application, performing a long term QoB analysis. If a QoB policy states that a migration should occur if violated, then the SLA component triggers a replannification alert to the Planner.

6. Conclusion

This document addressed the dynamic QoS verification and SLA management approach of SeaClouds. First we have defined a set of the metrics used for the representation of runtime QoS properties of the services, where each metric is also defined whether applying to a module-restricted scope or to the application seen as a whole.

Then, we have formally specified these metrics in *monitoring rules*, which also include the entities to monitor, and the actions to perform if a violation occurs. As described, these rules are used to initialize the SeaClouds monitor, which measures the different properties by means of different data collectors. We have also described the monitoring process in terms of the interactions between the core components of the SeaClouds platform.

Based on the monitoring data, we have shown how SeaClouds evaluates the monitoring policies, and how, if a violation occurs, SeaClouds tries to fix it by repairing the current deployment plan, or, replanning a new deployment plan.

Finally, we have also described how SeaClouds perform Quality of Business (QoB) analysis of the SLAs.

As future work, we plan to consolidate the design decisions of the aforementioned activities, implement them in the prototype and validate the approach through the case studies.

References

1. SeaClouds Project. Deliverable D6.1. Case study extended description (SeaClouds Consortium), March 2015.

2. ISO/IEC. ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models, 2010.

3. ModaClouds Project. MOdel-Driven Approach for design and execution of applications on multiple Clouds, 2015. https://github.com/deib-polimi/modaclouds-monitoring-manager/wiki.

4. ModaClouds Project. User's Manual, 2015. https://github.com/deib-polimi/modaclouds-qos-models/blob/master/doc/user-manual.md#actions.

5. SeaClouds Project. Deliverable D4.5. Unified dashboard and revision of Cloud API (SeaClouds Consortium), March 2015.

6. SeaClouds Project. Deliverable D2.4 Final SeaClouds Architecture (SeaClouds Consortium), February 2015.

7. Open Grid Forum, Web Services Agreement Specification (WS-Agreement), 2006. https://www.ggf.org/Public_Comment_Docs/Documents/Oct-2006/WS-AgreementSpecificationDraftFinal_sp_tn_jpver_v2.pdf

8. SeaClouds Project. Deliverable D4.3. Design of the run-time reconfiguration process (SeaClouds Consortium), February 2015.

9. SeaClouds Project. Deliverable 4.6. Prototype and detailed documentation of the SeaClouds run-time environment (SeaClouds Consortium), June 2015 (to be published).

10. SeaClouds Project. Deliverable D3.2. Discovery, Design and Orchestration Functionalities (SeaClouds Consortium), March 2015.

11. SeaClouds Project. Deliverable D4.1. Definition of the multi-deployment and monitoring strategies (SeaClouds Consortium), October 2014.