SeaClouds Project

D4.5 Unified dashboard and revision of Cloud API

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	D4.5
Deliverable Title	Unified dashboard and revision of Cloud API
Nature	Report
Dissemination Level	Public
Due Date:	M18
Submission Date:	March 3rd , 2015
Version:	1.0
Status	Final
Author(s):	Dionysis Athanasopoulos (POLIMI), Miguel Barrientos (UMA), Jose Carrasco (UMA), Javier Cubo (UMA), Elisabetta Di Nitto (POLIMI), Adrián Nieto (UMA), Marc Oriol (UPI), Diego Pérez (POLIMI), Román Sosa (ATOS)
Reviewer(s)	Dionysis Athanasopoulos (POLIMI), Andrea Turli (Cloudsoft), Javier Cubo (UMA)

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	08/03/15	First ToC	Adrián Nieto, Jose Carrasco
0.2	09/03/15	Second ToC and assignment of sections	Javier Cubo, Miguel Barrientos, Jose Carrasco
0.3	19/03/15	First contributions	Adrián Nieto, Javier Cubo, Miguel Barrientos, Jose Carrasco, Marc Oriol, Diego Pérez, Román Sosa, Dionysis Athanasopoulos
0.4	20/03/15	Internal revision to check missing contributions	Adrián Nieto
0.5	26/03/15	New contributions and last checkings	Adrián Nieto, Jose Carrasco, Miguel Barrientos, Román Sosa, Javier Cubo, Leonardo Bartoloni, Simone Zenzaro, Diego Pérez, Marc Oriol
0.6	27/03/15	Ready to be reviewed	Adrián Nieto, Jose Carrasco, Miguel Barrientos, Javier Cubo, Marc Oriol
0.7	30/03/15	Revised version	Andrea Turli, Román Sosa, Javier Cubo
1.0	31/03/15	Final version	Adrián Nieto, Miguel Barrientos

Table of Contents

Table of Figures

Table of Tables

Executive Summary

This document presents a revisited version of the SeaClouds Unified Application Programming Interface (API) and the Dashboard, based on the exposed work on the previous Deliverable D4.2. It describes the common language that every SeaClouds component will use to interact between each other. Also, it exposes the advances on the Dashboard design, where we simplify the user interaction by the usage of SeaClouds Assistants, which will guide the user during the usage of the Dashboard. Finally, it includes an example based on the Nuro Storyboard that we introduced in Deliverable D2.4 in order to show a real example of usage of the Dashboard and SeaClouds API.

1. Introduction

This document describes the design of the unified SeaClouds API, which allows to develop and use the different SeaClouds components in an isolate way in order to facilitate the reusability and modularity. Thus, the API provides the necessary mechanisms to expose and use the functionalities of each component.

Therefore, we describe the expected Dashboard functionalities providing a usable mockup, related to the high-fidelity prototype described in the Deliverable D5.2.2 Final Design of the User Interface [1] . This tool works as a SeaClouds API client and it implements an environment to use the SeaClouds functionalities using all the operations performed by the SeaClouds components that are provided by the API. The dashboard inherits the life-cycle to deploy and manage the applications on the target locations using the SeaClouds platform.

1.1. Structure of this document

The structure of this document is the following.

Section 2 provides a specification of the technology aspect of the SeaClouds API and an exhaustive description of the performed operations by each SeaClouds component.

In Section 3, we present the SeaClouds Dashboard starting, again, with the technical details and requirements. In order to describe the design and user experience we provide a functional mockup of the user interface.

Section 4 briefly illustrates the platform functionalities using the Nuro Use Case to show how the dashboard use the SeaClouds components API and how they interact with each other.

Finally, Section 5 presents some conclusions for this document.

1.2. Glossary of Acronyms

Here we list the different acronyms which will be used in this document.

Acronym	Definition
SaaS	Software-as-a-Service
PaaS	Platform-as-a-Service
IaaS	Infrastructure-as-a-Service
QoS	Quality of Service
QoB	Quality of Business
SLA	Service Level Agreement
GUI	Graphical User Interface
API	Application Programming Interface
APP	Application

AAM	Abstract Application Model
DAM	Deployable Application Model
ADP	Abstract Deployment Plan
RDF	Resource Description Framework
URI	Uniform Resource Identifier
MVC	Model View Controller
YAML	YAML Ain't Markup Language
XML	eXtensible Markup Language
REST	Representation state transfer

Table 1. Acronyms.

2. SeaClouds API

This section describes the different parts that compose the SeaClouds unified API, by defining the methods that will be available from each of the different SeaClouds components. The goal of this section is to give a common global interface for the SeaClouds dashboard to execute the primary objectives of the platform, such as designing, managing and re-configuring applications.

In Table 2, we present the template used in this document to describe the SeaClouds API corresponding to each component of the platform: Discoverer, Planner, Deployer, Monitor and SLA Service.

Basically, we are considering a Method identification, and provide a Description of the method. Also, the Parameters of the method are listed, as well as the Response of the method.

ID	MethodID (e.g: getApplications)
Description	(e.g: returns a list of applications)
Parameters	• **(Type) Param1**, a description of Param1 • **(Type) Param2**, a description of Param1
Response	• **(Type) MethodResponse**, a description of the response

Table 2. API method definition template.

2.1. Technical overview

From a technical point of view, SeaClouds unified API will be accessible from a single endpoint, and it will be internally organized in the same way that SeaClouds components are designed. In other words, each SeaClouds component (Discoverer, Planner - including the Matchmaker and Optimizer processes -, Deployer, Monitor, SLA Service) will implement its own REST API, all being gathered under the same REST hierarchy (Figure 1).

From a developer point of view, the usage of the API is homogeneous and transparent to the component implementation as the developer only has to concern about the external details of the SeaClouds Platform, which are defined in this deliverable.

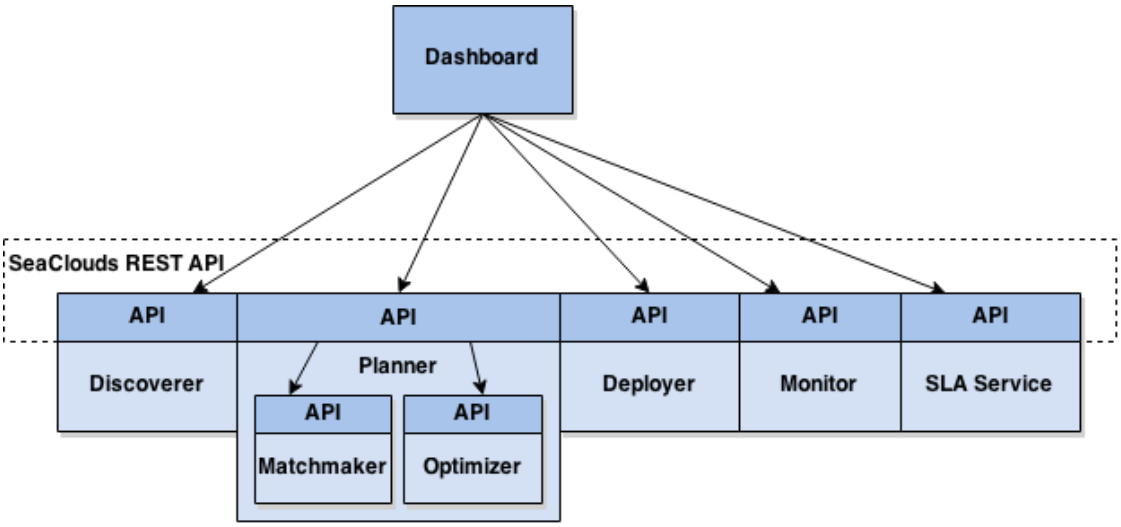

Figure 1. SeaClouds unified API interaction.

2.2. Discoverer API

The Discoverer is composed of several pluggable modules and a core Information system. The core information system stores the different cloud offerings and their properties in TOSCA YAML standard. This information is obtained by the different modules, which follow different strategies and have different capabilities according to the strategy.

2.2.1. Terminology

- **TOSCA document**: a document in TOSCA format - either as a string or file or as a parsed Java object representation.
- **CloudOfferingDocument**: a TOSCA document containing one or more node type definitions corresponding to cloud offerings. The format for this document is described in Deliverable D3.2 [2].
- **CloudOfferingID**: the name of a TOSCA node type referring to a cloud offering, expressed as a string.
- **CloudOfferingEnumerator**: the head of a linked list of CloudOfferingDocument representing the offers in the database. It has two methods: get() to retrieve the CloudOfferingDocument currently being pointed and next() to get a CloudOfferingEnumerator pointing to the next element in the enumeration. next() will return NULL if the element being pointed is the last element of the enumeration.

2.2.2. Interface

The Discoverer provides the following API:

- Add/remove cloud offering: adds/removes a cloud offering of the core information system (used by the modules of the discoverer)
- Update cloud offering properties: updates the properties of the cloud offering (used by the modules of discoverer)
- Enumerate cloud offerings: allows iteration of the cloud offerings and their properties from the information system. (used by the matchmaker)

ID	update
Description	Adds or replace a list of cloud offering to the discoverer database. Used by discoverer modules.
Parameters	● **(CloudOfferingDocument) cloudOfferings**, a set of cloud offerings to be included into the database. ● **(Boolean) overwrite**, if true cloud offerings with the same name will be updated, otherwise the update will fail if an offering with the same name is specified
Response	● **(Boolean) Success**, true if the database was updated successfully.

ID	getDefinition
Description	Get the definition of a cloud offering given its identifier.
Parameters	● **(CloudOfferingId) cloudOfferingId**, the unique id of the cloud offering whose definition is to be.
Response	● **(CloudOfferingDocument) cloudOffering**, a TOSCA document containing the node type definition corresponding to the required cloud offering ID. If no cloud offering exists for the given ID NULL is returned .

ID	removeOffer
Description	Removes a cloud offering from the database .
Parameters	● **(CloudOfferingId) cloudOfferingId**, the unique id of the cloud offering to be removed.
Response	● **(Boolean) Success**, true if the cloud offering was found and correctly removed.

ID	enumerateOffers
Description	Get an enumerator pointing to the first cloud offering. Used by the matchmaker to enumerate all the available offerings and fetch them one at a time.
Parameters	
Response	● **(CloudOfferingEnumerator) enumerator**, the head of a linked list of the offers in the database.

The different modules of the Discoverer provide different APIs depending on their nature. Examples of modules of the discoverer are:

- Brokering crawlers (e.g. CloudHarmony, PaaSify): provides an API to start the interaction with CloudHarmony, PaaSify.
- Service Provider's advertisements: provides an API for the Service Provider's to advertise directly their services to SeaClouds.
- Monitored information module: provides an API to be notified of QoS information of the cloud offerings
- Manual input module: provides an API to manually include cloud offering

2.3. Planner API

In this section we describe the available methods that are offered from the Planner API for example obtaining the best plan deployment for an application or plan a re-planification.

2.3.1. Terminology

- **TOSCA document**: a document in TOSCA format - either as a string or file or as a parsed Java object representation.
- **AAM**: a TOSCA document containing the Abstract Application Model as described in other deliverables [2*].
- **ADP**: a TOSCA document containing the Abstract Deployment Model, which is similar to the Abstract Application Model but it has cloud offerings associated with executable modules [2*].
- **DAM**: a TOSCA document containing the Deployment Application Model, which is similar to the Abstract Deployment Plan but it is augmented with the required information to perform the deployment.
- **LiveModel**: a TOSCA document containing topology, deployment, and runtime information of a running application.

2.3.2. Interface

ID	plan
Description	Implements the process of requiring application planning. Given the Abstract Application Model as TOSCA YAML input, the planner performs matchmaking and optimization by invoking the methods match and optimize respectively. The output of this process is a set of optimized deployment proposal for the given application.
Parameters	● **(AAM) model**, the Abstract Application Model for which planning is required
Response	● **(Set<ADP>) deploymentModels**, a set of optimized deployment proposal models

ID	match
Description	The planner offers to the user the option of performing matchmaking (i.e only the first step of the planning process). This method invokes the internal component Matchmaker, which implements the functionality.
Parameters	● **(AAM) model**, the Abstract Application Model in TOSCA YAML for which matchmaking is required
Response	● **(Map<ModuleName, CloudOfferingDocument>) matchingOffers**, a map associating a set of possible cloud offerings to e each module in the input Abstract Application Model.

ID	optimize
Description	The planner offers to the user the option of performing optimization (i.e. only the second step of the planning process). This method, invokes the internal component Optimizer, which implements the functionality.
Parameters	● **(AAM) model**, Abstract Application Model in TOSCA YAML that contains the information of application modules, application topology, QoS requirements, QoS properties and names of cloud services that can be used for each module in an AAM ● **(Map<ModuleName, CloudOfferingDocument>) suitableCloudOffers**, a map from the Abstract Application Model modules to the set of matching cloud offers containing the information retrieved by the Discoverer module of suitable cloud services and information regarding communication capabilities of

	clouds
Response	● **(Set<ADP>) candidatePartialPlans**, the output is an set of candidate partial plans where, in each plan, each module is associated to one and only one cloud service. The internal optimization problem aims at satisfying the performance and/or availability requirements while minimizing the expected expenses on using computing cloud resources assuming that they are used in a “pay-as-you-go” settings

ID	generateDAM
Description	Generates the Deployment Application Model from the Abstract Deployment Plan. To do so, it interacts with the user to obtain additional information regarding credentials,policies, etc. that is required to perform the deployment.
Parameters	● **(ADP) deploymentModel**, the Abstract Deployment Plan in TOSCA YAML.
Response	● **(DAM) deploymentModel**, The Deployable Application Model in TOSCA YAML with the required information to perform the deployment.

ID	replan
Description	Implements the replanning phase for a running application. The Planner takes the Abstract Application Model and the current Live Model for the user application. The Live Model provides also the information about replanning cause. The output of this process is a set of optimized deployment proposal for the given application.
Parameters	● **(AAM) abstractModel**, the Abstract Application Model in TOSCA YAML for which replanning is required ● **(LiveModel) liveModel**, the current Live Model containing also the informations about violations and replanning causes
Response	● **(Set<ADP>) deploymentProposals**, a set of optimized deployment proposal models

2.3.3. Matchmaker

The Matchmaker iterates the list of available cloud offerings from the Discoverer and selects those which are suitable to implement the modules of the application given the requisites from the user.

2.3.3.1. Terminology

- **TOSCA document**: a document in TOSCA format - either as a string or file or as a parsed Java object representation
- **AAM**: a TOSCA document containing the Abstract Application Model as described in D3.2
- **Module**: the smallest deployable entity of the application, described in TOSCA by a Node Template (see D3.2)
- **ModuleName**: the unique name of the node template used to specify a module in the AAM
- **CloudOfferingDocument**: a TOSCA document containing one or more node type definitions corresponding to cloud offerings. The format for this document is described in D3.2

2.3.3.2. Interface

ID	match
Description	Implements the matching process. Given the Abstract Application Model for the application, each module is matched with available cloud offers according to its functional properties. The mapping between modules and matching offers is returned.
Parameters	● **(AAM) model**, the Abstract Application Model in TOSCA YAML for which matchmaking is required
Response	● **(Map<ModuleName, CloudOfferingDocument>) matchingOffers**, a map associating a set of possible cloud offerings to each module in the input Abstract Application Model.

2.3.4. Optimizer

The Optimizer module within the planner is mainly composed of an optimization problem solving[1] component. It provides interfaces for two different problems: a) deciding the cloud services to use in the initial deployment (called *optimize*) and b) deciding the cloud services to use in subsequent deployments when the application has to migrate, at least partially, from its original deployment (called *reoptimize*).

2.3.4.1. Terminology

- **TOSCA document**: a document in TOSCA format - either as a string or file or as a parsed Java object representation.
- **AAM**: a TOSCA document containing the Abstract Application Model as described in other deliverables.
- **Module**: the smallest deployable entity of the application, described in TOSCA by a Node Template [2*].
- **ModuleName**: the unique name of the node template used to specify a module in the AAM.
- **CloudOfferingDocument**: a TOSCA document containing one or more node type definitions corresponding to cloud offerings. The format for this document is described in Deliverable 3.2 [2*]
- **ADP**: a TOSCA document containing the Abstract Deployment Model, which is similar to the Abstract Application Model but it has cloud offerings associated with executable modules [2*]
- **LiveModel**: a TOSCA document containing topology, deployment, and runtime information of a running application.

2.3.4.2. Interface

ID	optimize
Description	It produces a set of candidate partial plans where, in each plan, each module is associated to one and only one cloud service.
Parameters	● **(AAM) model**, Abstract Application Model in TOSCA YAML that contains the information of application modules, application topology, QoS requirements, QoS properties and names of cloud services that can be used for each module in an AAM. ● **(Map<ModuleName, CloudOfferingDocument>) suitableCloudOffers** , a map from the Abstract Application Model modules to the set of matching cloud offers containing the information retrieved by the Discoverer module of suitable cloud services and information regarding communication capabilities of clouds.
Response	● **(Set<ADP>) candidatePartialPlans**, the output is an set of candidate partial plans where, in each plan, each module is associated to one and only one cloud service. The internal optimization problem aims at satisfying the performance and/or availability requirements while minimizing the expected expenses on using computing cloud resources assuming that they are used in a “pay-as-you-go” settings.

ID	reOptimize

Description	It produces a set of candidate partial reconfiguration plans where each plan specifies the modules to migrate and their target cloud service
Parameters	● **(AAM) model**, Abstract Application Model in TOSCA YAML that contains the information of application modules, application topology, QoS requirements, QoS properties and names of cloud services that can be used for each module in an AAM. ● **(Map<ModuleName, CloudOfferingDocument>) suitableCloudOffers** , a map from the Abstract Application Model modules to the set of matching cloud offers containing the information retrieved by the Discoverer module of suitable cloud services and information regarding communication capabilities of clouds. ● **(ADP) oldModel**, deployment model that was used to deploy the application before replanning triggered. ● **(LiveModel) liveModel**, model containing real time information about the application currently deployed, including the violation which triggered the replanning.
Response	● **(Set<ADP>) candidatePartialReconfigurationPlans**, the response is a set of candidate partial reconfiguration plans, where, in each of these candidates it is specified the information for changing from the current DAM that is no longer valid to a computed alternative deployment that overcomes the current DAM problems. The internal optimization problem aims at satisfying the performance and/or availability requirements while minimizing both the expected expenses on using computing cloud resources assuming that they are used in a "pay-as-you-go" settings and the application modules that need to be migrated from their current deployment.

2.4. Deployer API

The Deployer API provides the resources to deploy and manage application modules. It contains the methods for managing the application deployments.

2.4.1. Terminology

- **Application**: represents an application which was deployed by the Deployer.
  - *ID*: unique identifier.
  - *Name*: given name for the application.
  - *Status*: current lifecycle status of the application (*"Starting", "Running", "On fire", "Stopped",* etc.)

- *ConfigParameters*: list of configured application parameters.
- *Policies*: list of attached policies.
- *Modules*: modules that compose the application.
- *Effectors*: operations that can be invoked on application.

- **Module**: represents each of the modules that compose an application.
  - *ID*: unique identifier.
  - *Name*: given name for the module.
  - Status: current lifecycle status of the entity ("Starting", "Running", "On fire", "Stopped", etc.).
  - *ConfigParameters*: list of parameters configured on the module. This includes environment variables, endpoint description, domain, ports, etc.
  - *Policies*: list of attached policies.
  - *Effectors*: operations that can be invoked on module.
- **Effector**: represents the possible actions that can be performed on each application module.
  - *Action*: the action that will be performed.
  - *Description*: a description of the action.
- **Location**: represents the cloud provider where the managed application modules will be deployed.
  - Provider.
  - Region.

2.4.2. Interface

ID	getApplication
Description	Returns the details of an existing application (modules, status, location, etc).
Parameters	• **(String) applicationId**: ID of the application.
Response	• **(Application) application**: found application

ID	getModule
Description	Returns the details of an existing application module (status, location, policies, configuration, etc).
Parameters	• **(String) applicationId:** ID of the application. • **(String) moduleId:** ID of the module.
Response	• **(Module) module:** application module details.

ID	getApplications

Description	Returns the list of deployed applications.
Parameters	
Response	● **(Application[]) applications:** list of available deployed applications

ID	createApplication
Description	Creates and deploys a new application, given a Deployable Application Model.
Parameters	● **(String) deployableApplicationModel**: application description
Response	● **(String) applicationId:** ID of the created application

ID	deleteApplication
Description	Removes a running application, releasing all cloud resources associated to it.
Parameters	● **(String) applicationId**: ID of the application to be removed.
Response	

ID	getEffectors
Description	Returns the list of available effectors for an Application Module.
Parameters	● **(String) applicationId**: ID of the application. ● **(String) moduleId**: ID of the module where to retrieve the effectors.
Response	● **(Effector[]) effectors**: List of effectors available on the module.

ID	callEffector
Description	Triggers an effector action associated to a module.
Parameters	● **(String) applicationId**: ID of the application. ● **(String) moduleId**: ID of the module that contains the target effector. ● **(String) effector**: ID / action to be triggered ● **(String[]) effectorParameterList**: additional parameters

	required by the effector.
Response	● (**Application) application**: found application

ID	getSupportedLocations
Description	Retrieves the available locations in the deployer.
Parameters	
Response	● **(Location[]) locations**: list of currently supported locations.

ID	getAvailablePolicies
Description	Retrieves the available policies that can be attached to certain kind of module.
Parameters	● **(String) moduleType**: module type.
Response	● **(Location[]) locations**: list of currently supported locations.

2.5. Monitor API

SeaClouds monitoring platform encapsulates and exploits the functionality offered by MODAClouds monitoring platform [4]. The latter platform uses four core components, the Monitoring Manager, the Knowledge Base, the Data Analyzer, and one or more Data Collectors, as specified below:

- The Monitoring Manager is the coordinator of MODAClouds platform.
- The Knowledge Base contains an ontology and a permanent RDF database. The ontology is a formal specification of the common abstractions needed to represent and monitor cloud applications and to describe the mutual relationships among them. The permanent RDF database contains information about the running system and the current monitoring platform configuration.
- A Data Collector is responsible for collecting monitoring data from cloud resources and applications and to associate semantic information to the data.
- The Data Analyzer processes monitoring data coming from data collectors and tries to detect on-the-fly patterns that emerge directly from the data, without the need of major transformations of the data itself.

By using the internal components, SeaClouds monitoring platform offers to the external SeaClouds components (Deployer, Planner, SLA Service, and Dashboard) the following overall functionality: it collects raw monitoring data, it offers these data to Observers interested on them (i.e.: the Deployer), and it sends events to the

Planner, the SLA service, and the Dashboard in case a set of monitoring rules are violated. The notion of a monitoring rule has been specified in previous deliverables [5].

To realize this functionality, the external SeaClouds components interact with the internal components of the monitoring platform. For instance, the Deployer initiates all the internal components of the monitoring platform, while the Planner, the SLA service, and the Dashboard subscribe to the Monitoring Manager in order to retrieve events.

To reduce the coupling between the external and the internal components of the monitoring platform, we adopt the mediator design pattern. Based on this pattern, the interaction between components is encapsulated with a mediator component, which we call here "controller". In this way, components no longer interact directly with each other, but instead interact through the controller. In other words, the controller defines the API for the interaction between the external and the internal components of the monitoring platform.

2.5.1. Terminology

- **Endpoint**: URIs of (internal and external) servers with which the Monitor interact.
- **MetricName**: the name of a monitoring metric, which is commonly used among SeaClouds components (e.g., Planner, Deployer), and corresponds to the kind of the data that will be measured by the Monitor.
- **DataCollector**: the executable file (e.g., .jar) of a data collector.
- **DescrPlan**: the low-level representation (e.g., String) of the deployment plan of an application.
- **Rule**: the low-level representation (e.g., String) of a monitoring rule, according to the XML schema [5].
- **mID**: the identifier of a monitoring rule, as specified in the XML schema [5].
- **Callback**: URI of SeaClouds component that is interested to retrieve raw monitoring data.
- **ReplanningEvent**: the low-level representation (e.g., String) of the information, needed by a SeaClouds component, in order to be informed about violations of a monitoring rule that lead to a replanning.

2.5.2. Interface

The controller API includes the methods of the following tables, which prototype the core functionalities offered by SeaClouds monitoring platform.

ID	initialize
Description	It provides URIs of MODAClouds servers and SeaClouds

	components with which SeaClouds Monitor interact.
Parameters	• **(URI[]) endPoints**, a set of endpoint addresses.
Response	

ID	getAvailableMetrics
Description	It returns the names of the available metrics, for which data collectors are available in the SeaClouds monitoring platform.
Parameters	
Response	• **(String[]) metricNames**, a set of available metric names.

ID	getDataCollectors
Description	It accepts as input a set of metrics and returns the data collectors that measure the values of these metrics.
Parameters	• **(String[]) metricNames**, a set of metric names.
Response	• **(File[]) dataCollectors**, a set of executable files of data collectors.

ID	installDeploymentPlan
Description	It accepts as input the description of the deployment plan for an application and uploads this plan to the Monitoring Manager.
Parameters	• **(String) descrPlan**, the description of the deployment plan of an application.
Response	

ID	installMonitoringRules
Description	It accepts as input the monitoring rules for an application, uploads these rules to the Monitoring Manager, and activates them.
Parameters	• **(String) rules**, the set of the input monitoring rules.
Response	

ID	uninstallMonitoringRule

Description	It accepts as input the id of a previously installed monitoring rule and deletes the corresponding rule from the platform.
Parameters	● **(Integer) mID**, the identifier of a monitoring rule.
Response	

ID	addObserver
Description	It accepts as input the name of a metric and a callback URI and sends the collected monitoring data for the input metric to the callback URI.
Parameters	● **(String) metricName**, the name of a metric. ● **(URI) callback**, the endpoint address of component that is interested in receiving raw monitoring data.
Response	

ID	sendReplanningEvent
Description	It accepts as input a re-planning event,produced by the Deployer, and forwards it to SeaClouds components (Planner, SLA Service, and Dashboard) that have subscribed to the violated monitoring rule.
Parameters	● **(String) replanningEvent**, the information, needed by a SeaClouds component, in order to be informed about violations of a monitoring rule that lead to a replanning.
Response	

2.6. SLA Service API

The SLA Service API provides the methods to manage templates and agreements of the two SLA levels identified in SeaClouds.

2.6.1. Terminology

- **Agreement**: document that describes the delivered service, the involved parties, and the non-functional properties that the service must fulfill.
- **AgreementId**: Unique identifier of the agreement.
- **Template**: document that describes a provider offer. Actual agreements may be based on templates.

- **TemplateId**: Unique identifier of the template.
- **Guarantee term**: Term that express service guarantees in an agreement, define how guarantees are assessed and which compensation methods apply in case of meeting or violating the service guarantees.
- **ServiceId**: Identifier of a service being delivered. In the case of Customer - Application Provider level, it corresponds to the application id; otherwise, it corresponds to an identifier of the actual service offered by the cloud provider.
- **Resource**: Identifier of the entity using a service. Used in the Application Provider - Cloud Provider level, corresponding to the moduleId(s) being hosted.
- **Enforcement**: Process that evaluate the guarantee terms are being fulfilled.
- **QoB (Quality of Business)**: express a constraint over business-related metrics and the penalties and recovery actions that are applied in case this constraint is violated.

2.6.2. Interface

ID	getAgreement
Description	Retrieves the agreement identified by its id.
Parameters	• **(AgreementId) Id**, WS-Agreement Id of agreement.
Response	• **(Agreement)** WS-Agreement representation of agreement.

ID	getAgreements
Description	Retrieves all the agreements that match the filter.
Parameters	• (**ProviderId**) provider, Id of a Provider. • (**ServiceId**) service, Id of a Service. • **(ResourceId)** resourceId, Id of a Resource. • (**ConsumerId**) consumer, Id of a Consumer. • **(TemplateId) templateId**, WS-Agreement template id of agreements based on this template.
Response	• **(Agreement[])** WS-Agreement representation of agreements matching the filter.

ID	createAgreement
Description	Creates an agreement for a given application.
Parameters	• **(AAM) aam**, Abstract Application Model of the application. • **(DAM) dam**, Deployable Application Model of the application.

Response	● **(Agreement[])** WS-Agreement (including generated AgreementId) representation of the created agreements: - 1 agreement in Customer - Application Provider level. - n agreements in Application Provider - Cloud Provider level, one per each cloud service used in the plan.

ID	updateAgreement
Description	Updates an existing agreement.
Parameters	● **(Agreement) description**, WS-Agreement representation of the agreement. It may be internally modified, so the parameter should not be taken as the agreement finally stored.
Response	● **(Agreement)** WS-Agreement representation of the agreement.

ID	terminateAgreement
Description	Changes an agreement state to "Terminated" and stops any enforcement.
Parameters	● **(AgreementId) Id**, WS-Agreement Id of agreement.
Response	

ID	getTemplate
Description	Retrieves the template identified by its id.
Parameters	● **(TemplateId) Id**, WS-Agreement Id of template.
Response	● **(Template)** WS-Agreement representation of template.

ID	getTemplates
Description	Retrieves all the templates that match the filter.
Parameters	● (ProviderId) provider, Id of a Provider. ● (ServiceId) service, Id of a Service.
	● **(Template[])** WS-Agreement representation of templates matching the filter.

ID	createTemplate

Description	Creates a template.
Parameters	● **(Template) description**, WS-Agreement representation of the template. It may be internally modified, so the parameter should not be taken as the template finally stored.
Response	● **(Template)** WS-Agreement representation of template (including generated TemplateId).

ID	getAgreementStatus
Description	Retrieves the status (violated, not violated) of service level objectives and the overall agreement.
Parameters	● **(AgreementId) Id**, WS-Agreement Id of agreement.
Response	● **(AgreementStatus)** status of agreements and its respective guarantee terms.

ID	startEnforcement
Description	Starts the enforcement of an agreement.
Parameters	● **(AgreementId) Id**, WS-Agreement Id of agreement.
Response	● **(Boolean)** status of enforcement.

ID	stopEnforcement
Description	Stop the enforcement of an agreement.
Parameters	● **(AgreementId) Id**, WS-Agreement Id of agreement.
Response	● **(Boolean)** status of enforcement.

ID	createProvider
Description	Creates a provider.
Parameters	● **(Provider) description**, name and description of provider to create.
Response	● **(Provider)** SLA Service representation of provider (including generated ProviderId).

ID	getQoSViolations
Description	Get a list of QoS violations that match the filter.
Parameters	• (**AgreementId**) agreementId. • (**ProviderId**) provider, Id of a Provider. • (**ServiceId**) service, Id of a Service. • **(ResourceId)** resourceId, Id of a Resource. • **(xs:datetime[]) dateInterval**, if provided, violation must be in the interval.
Response	• **(QoSViolation[])** Violations matching the filter.

ID	getQoBViolations
Description	Get a list of QoB violations that match the filter.
Parameters	• (**AgreementId**) agreementId. • (**ProviderId**) provider, Id of a Provider. • (**ServiceId**) service, Id of a Service. • **(ResourceId)** resourceId, Id of a Resource. • **(xs:datetime[]) dateInterval**, if provided, violation must be in the interval.
Response	• **(QoBViolation[])** Violations matching the filter.

ID	receiveQoSViolation
Description	Notifies the SLA Service a QoS violation.
Parameters	• **(QoSViolation) violation**, violation sent by the Monitor.
Response	

ID	receiveHealingNotification
Description	Notifies the SLA Service that the Policy Action was already done.
Parameters	• (**AgreementId**) agreementId. • (**PolicyId**) policyId. • (**QoSViolationId**) violationId.
Response	

3. SeaClouds Dashboard

The main goal of the SeaClouds Dashboard is to provide a simple interface to the application administrator, where the description in this Deliverable is more focused on the back-end, then considering the internal connections, related with the user interface described in Deliverable D5.2.2 [1], more focused on the front-end.

3.1. Technical overview

The Dashboard is a pure HTML5 + JavaScript application. It uses REST calls to interact with the SeaClouds Platform (Figure 2). The SeaClouds Dashboard is based on the Bootstrap library [6], which allows adapting the website to the size of the screen, providing a nice user experience with independence of the device (mobile, tablet or traditional desktop). It also uses technologies like AngularJS [7] as a client side MVC to provide all the functionality.

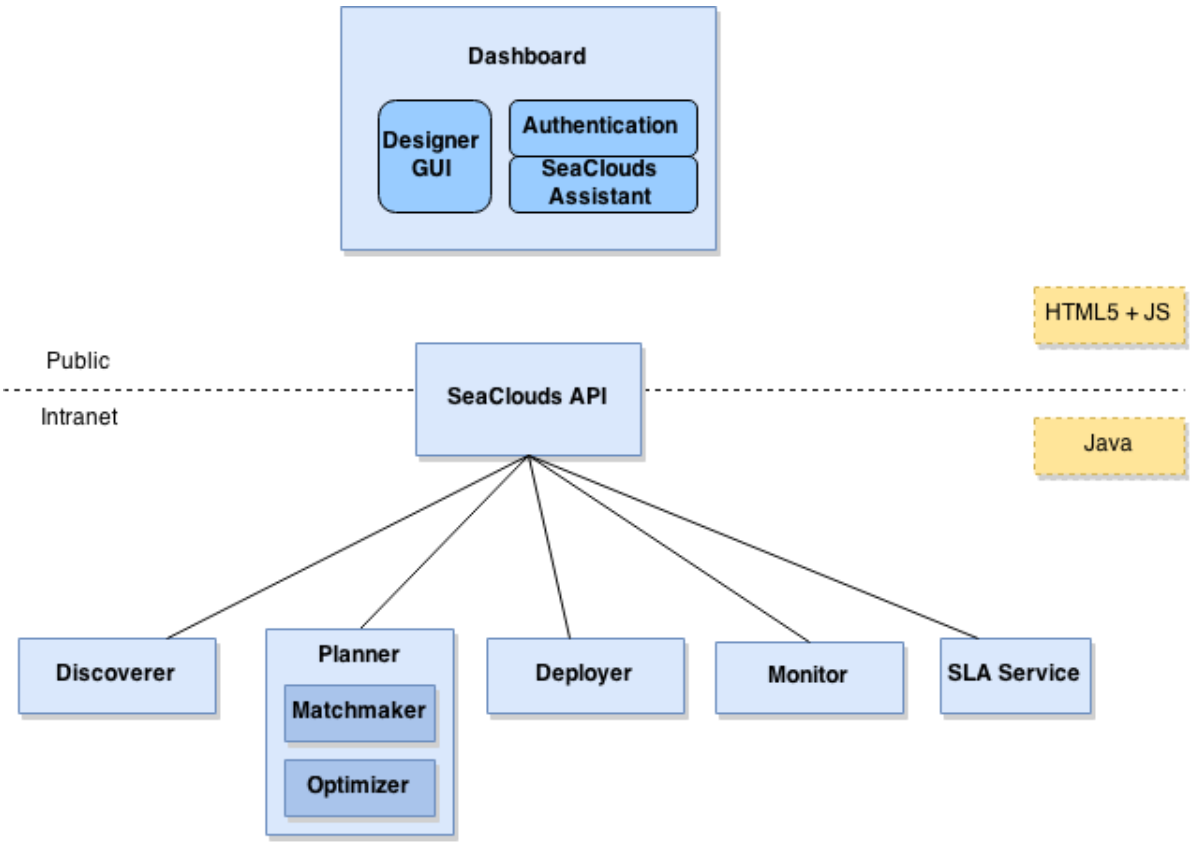

Figure 2. Dashboard interaction diagram.

3.2. Design and user experience

In this section we present the Dashboard user interface which will be employed to use the services that are provided by SeaClouds. We describe the available functionalities, like create and configure an application and its post-management.

An important goal of this section is to specify the interaction among the SeaClouds components for carrying out the different supported operations.

In the previous version of the dashboard, in Deliverable D4.2 [8], we proposed an individual usage of the SeaClouds components, as they all were not really connected. Although the consortium is thinking on the possibility of exploiting SeaClouds functionalities as a whole and also using the components individually (considering also the exploitation of design-time and runtime toolkits, or even intermediate brokers - several components used for several platforms at the same time), here we focus on the use of the dashboard following a wizard style, guiding the user through all the components.

We are currently working on analyzing the dependencies among components, so that single functionalities can be offered separately to the user (Discover component, Deployer component, etc.). Since we do not have real conclusions on this yet (that could be published in other future deliverables, such as “Deliverable 4.6 Prototype and detailed documentation of the SeaClouds run-time environment” [9]), we prefer to focus on the design and user experience on the option "SeaClouds as a whole", in order to avoid incoherencies between the wizards and the usage of individual components.

Therefore, we plan to consider the generation of an advanced mode which allows the user to interact with the modules independently (for example, discovering Cloud Offerings without deploying an application, or deploying an existing deployment plan specified in TOSCA YAML).

3.2.1. Main View

Once the user has logged in to SeaClouds Platform, the dashboard shows the existing applications managed with SeaClouds and an option to add new ones (Figure 3).

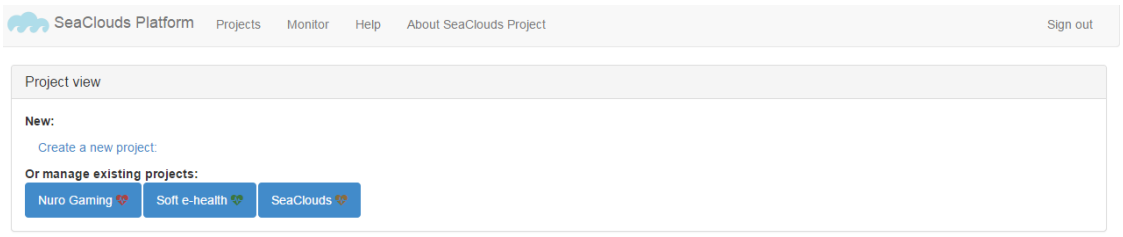

Figure 3. SeaClouds Main View.

Internally, the dashboard will ask the deployer about which applications are already deployed by using the Deployer API, together with the global status of the application (getApplications).

3.2.2. Status View

In Figure 4, once the application is selected the SeaClouds Dashboard shows the status of the application, and the current topology with the status of each node (getApplication)

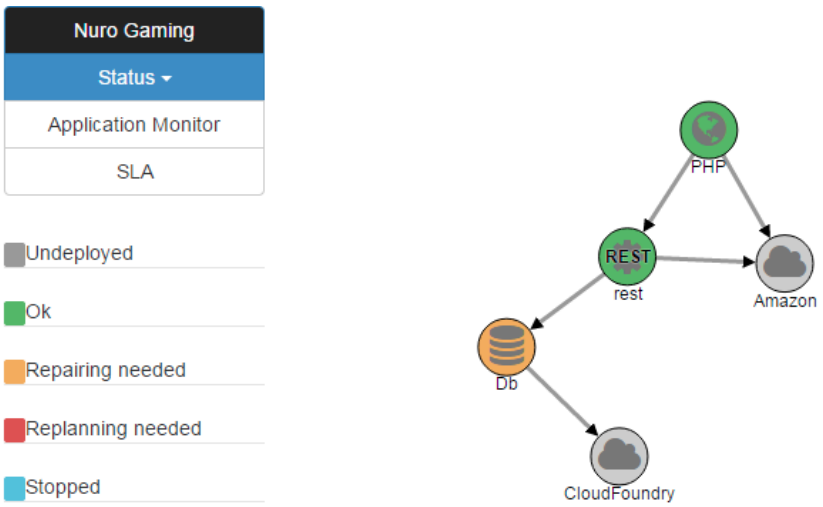

Figure 4. SeaClouds status view.

3.2.3. SeaClouds Assistants

SeaClouds Dashboard will ease to the Application Administrator / Designer the interaction with the Platform thanks to the SeaClouds Assistants, which hide unnecessary information about the processes or displaying the information in a simple way (for example the user may not need to see the DAM before deploying it) as it can provide the most common features.

3.2.3.1. Add new application

"Add new application wizard" (Figures 5, 6, 7, 9 and 10) hides most of the details about the representation and the transmission of the data across the entire platform. It's designed to hide to the user low level details like TOSCA files or how the modules interact between them.

The user only has to design the application by using a simplified version of the Designer GUI, which does not output any TOSCA file. After the Application Topology is defined, the user inputs the requirements of his application. Then, SeaClouds will suggest the ideal Cloud Providers, and will provide a one-click solution to deploy the application. Until the user confirms the deployment, the dashboard saves the new application information retrieved from each step in the client, waiting for the confirmation to send the data to SeaClouds platform.

"Add new application wizard" starts asking the user for the name of the application and the global properties which should be optimized as we can see in Figure 5.

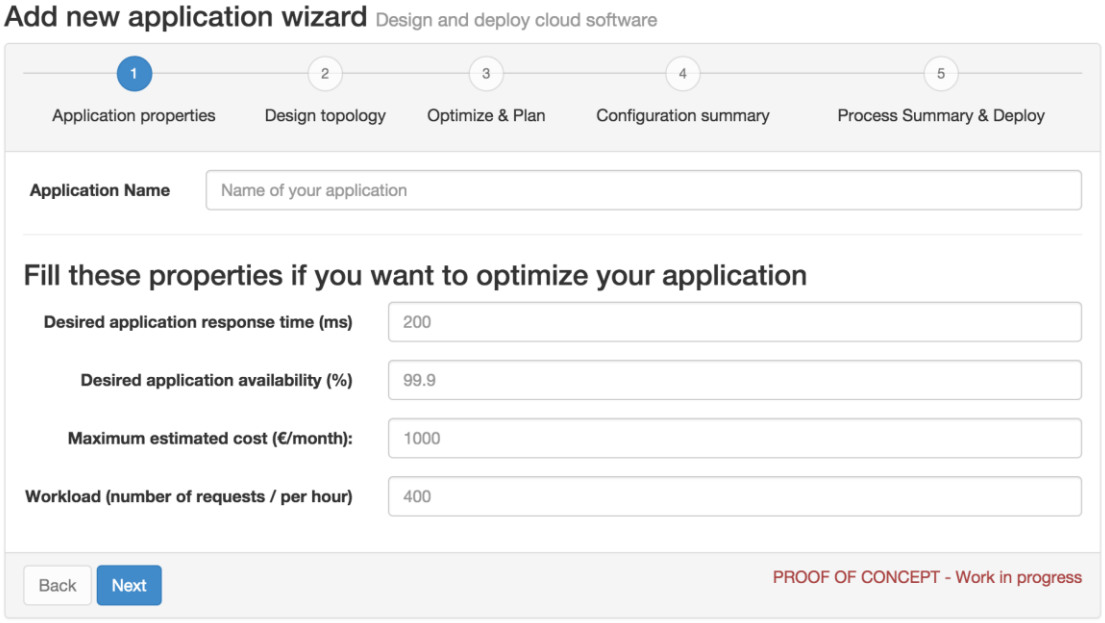

Figure 5. Add a new application wizard, step 1.

Once the step 1 is completed, the user proceeds to design graphically the topology of the application (Figure 6).

The user can add each of the modules that compose the application, configuring each of them through an individual interface (see Figure 7). This configuration includes technological requirements (like application language) and non-functional requirements, which includes information about the cost, location, reconfiguration policies and QoS constraints. The user also chooses between using IaaS or PaaS infrastructure.

Once all this information is filled, the user can keep adding more modules in the same way. When all application modules are configured, the Step 2 concludes.

In Figure 8 we can see how the Dashboard interacts with the SeaClouds Components during the module creation process.

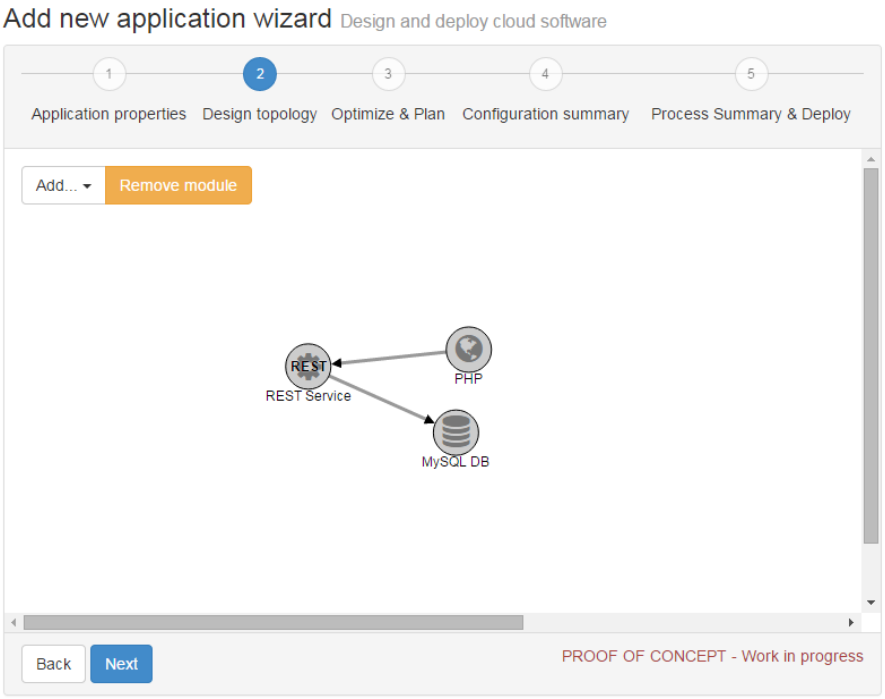

Figure 6. Design an application topology, step 2.

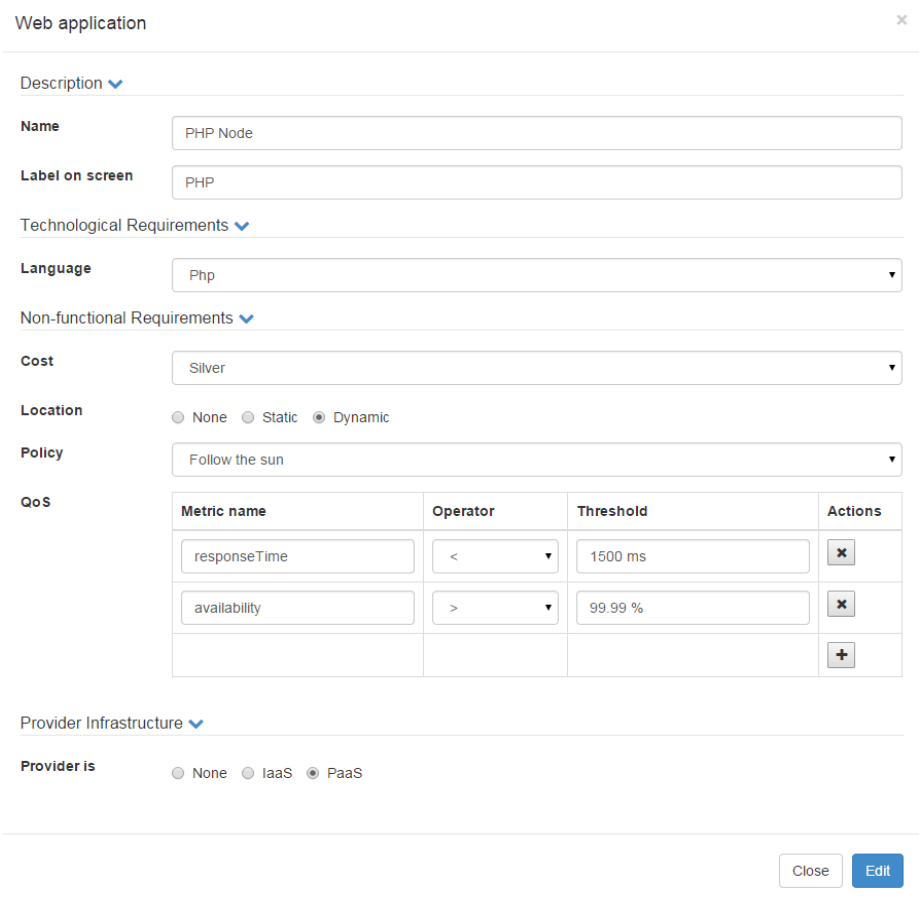

Figure 7. Configuring an application module, step 2.

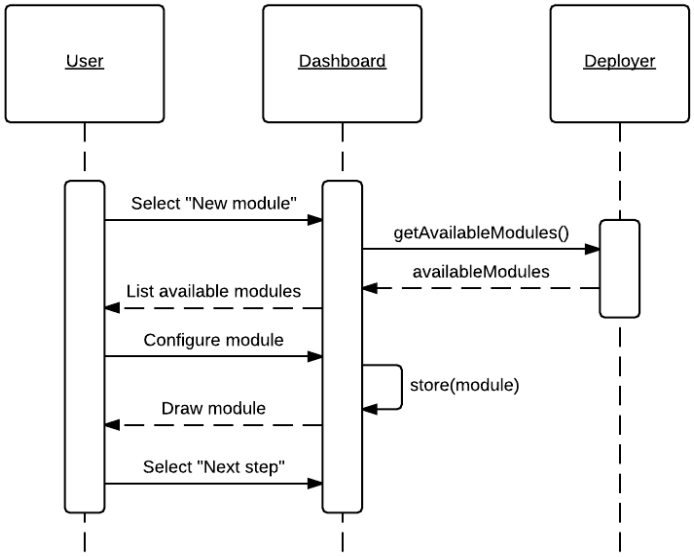

Figure 8. Designing the application topology, step 2. Sequence diagram.

Before the third step of the wizard is visible to the user, the Dashboard sends the AAM to the Matchmaker & Optimizer to retrieve the Cloud Offerings. When the Dashboard has all the required information it shows it to the user (step 3), as we can see in Figure 9.

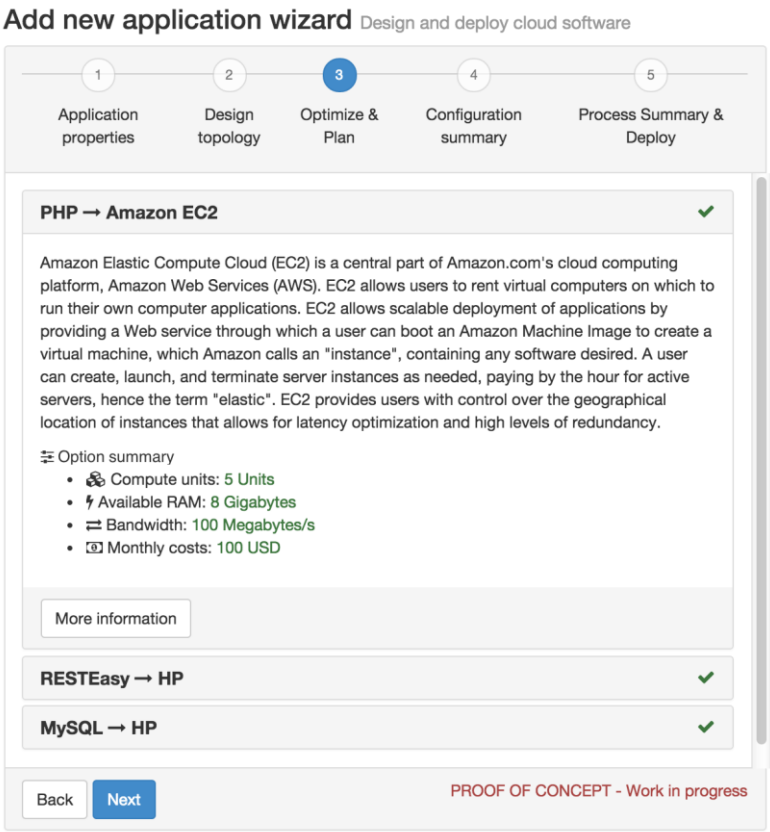

Figure 9. Optimized plan selection, step 3.

The step 4 (Figure 10) is just a summary where the user can review what will be the result after the deployment process finishes. In this step, the Dashboard checks the topology and shows the information about the selected providers, cloud resources, estimate cost and other key properties of the application. Also, the user may modify the SLA. Once the user checks and approves the generated profile, final step 5 shows the result of deploying the application.

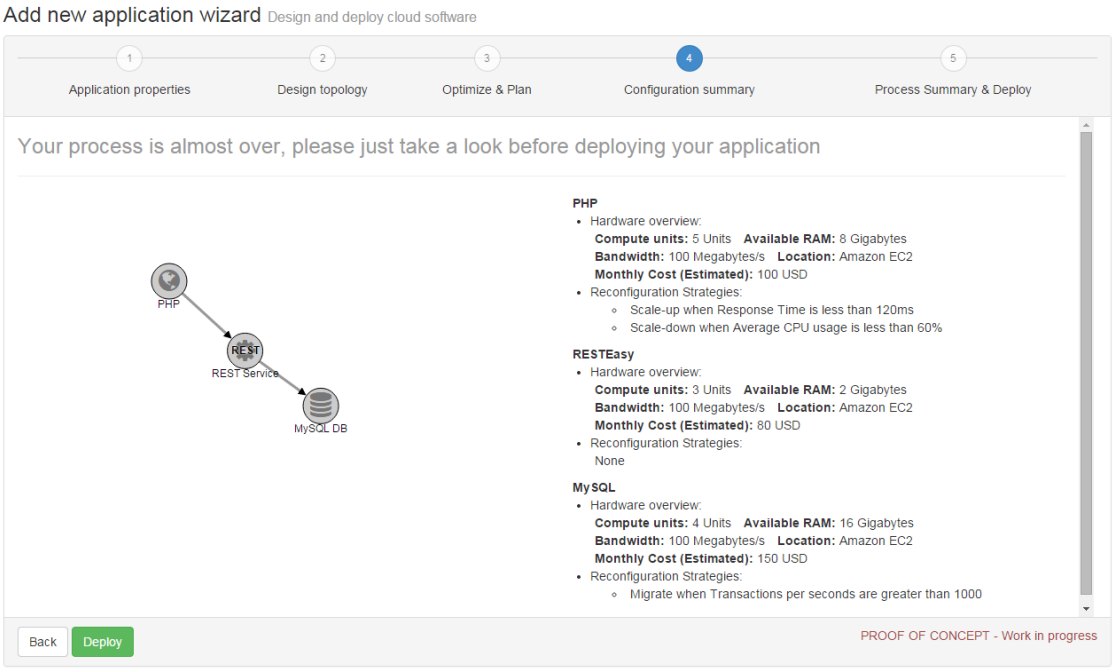

Figure 10. Configuration summary, step 4.

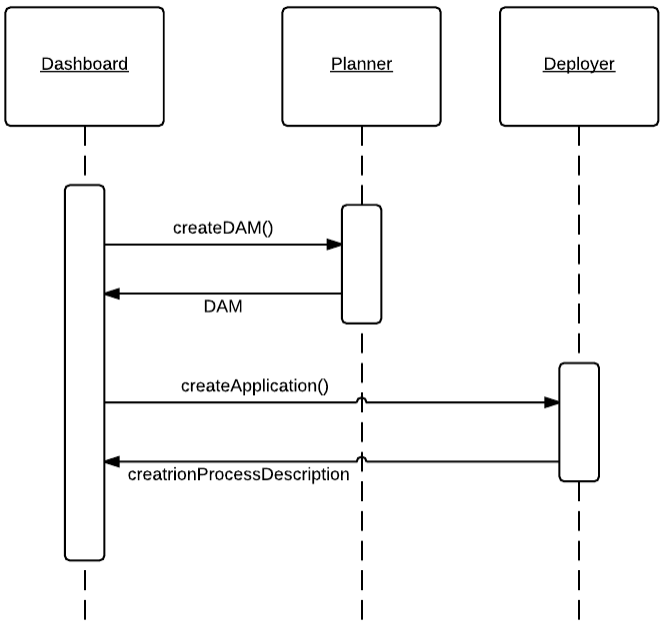

Figure 11. Generation and deployment of a DAM. Sequence diagram.

The Dashboard requests a DAM to the Planner (method), and sends it to the Deployer (createApplication) who will finish the deployment autonomously (Figure 11), notifying the description process based on the result of the deployment.

3.2.3.2. Remove an existing application

The remove application assistant is a very simple wizard. It allows the user to remove an application in three steps.

The first step (Figure 12) uses the Deployer API to retrieve from the live model which applications are currently running on SeaClouds (getApplications).

Then, after the user selects one of them, the second step (Figure 13) of the wizard shows the information about the application that the user wants to remove in order to allow him to check if he selected the right application.

During the third step (Figure 14) is where the deletion process occurs. First, the dashboard calls the SLA Service to remove the agreements associated to the application.

Next, it calls the Deployer API to remove the application itself (deleteApplication), notifying the result based on the result of the expunge process (Figure 15).

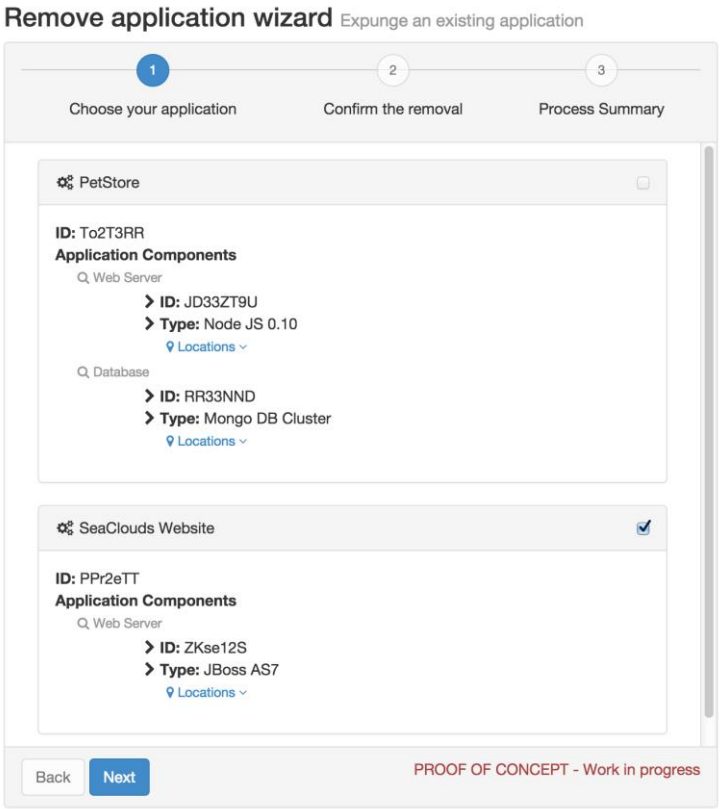

Figure 12. Selecting the application which will be removed, step 1.

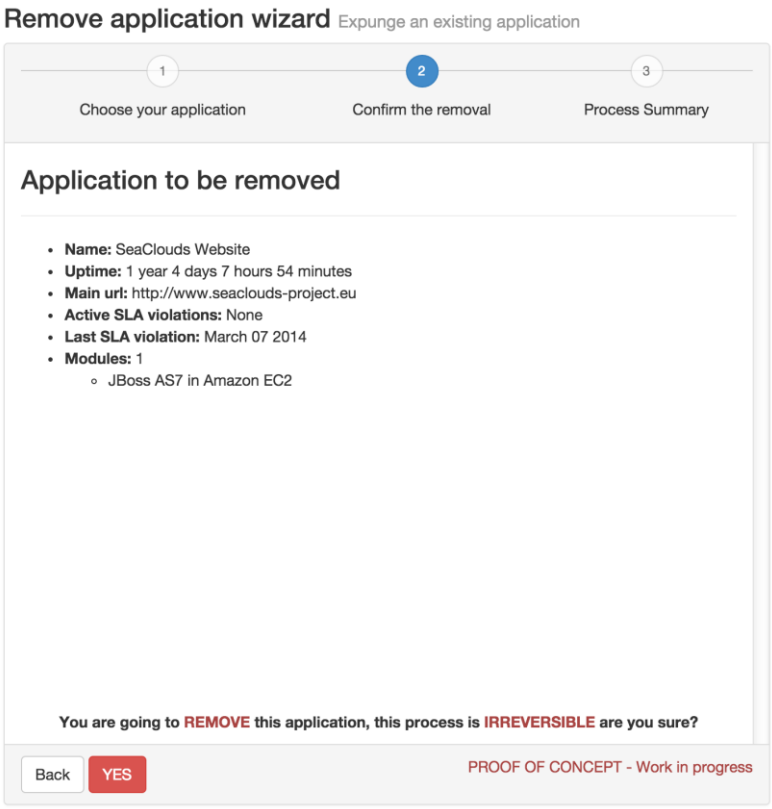

Figure 13. Removal operation confirmation, step 2.

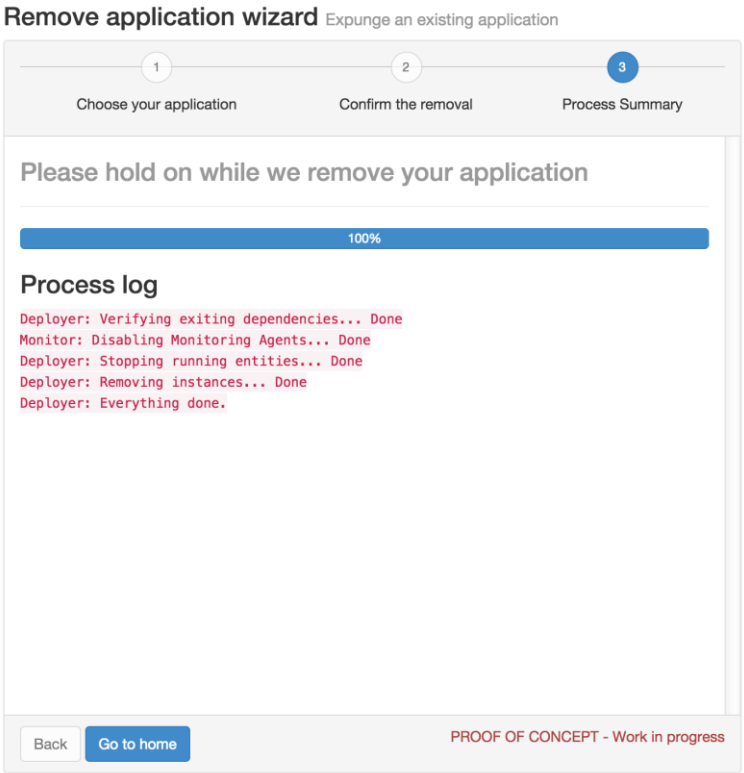

Figure 14. Remove process summary, step 3.

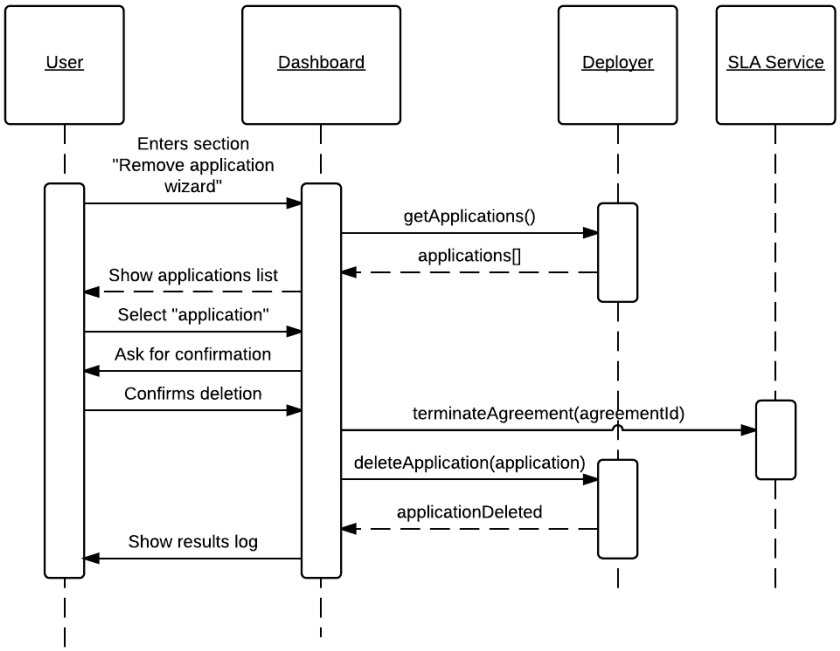

Figure 15. Remove application wizard. Sequence diagram.

3.2.4. Monitor section

Once an application is deployed and running, it is able to be monitored using the Monitor interface. In Figure 16 we can see an example where several aspects of an application are being followed and shown.

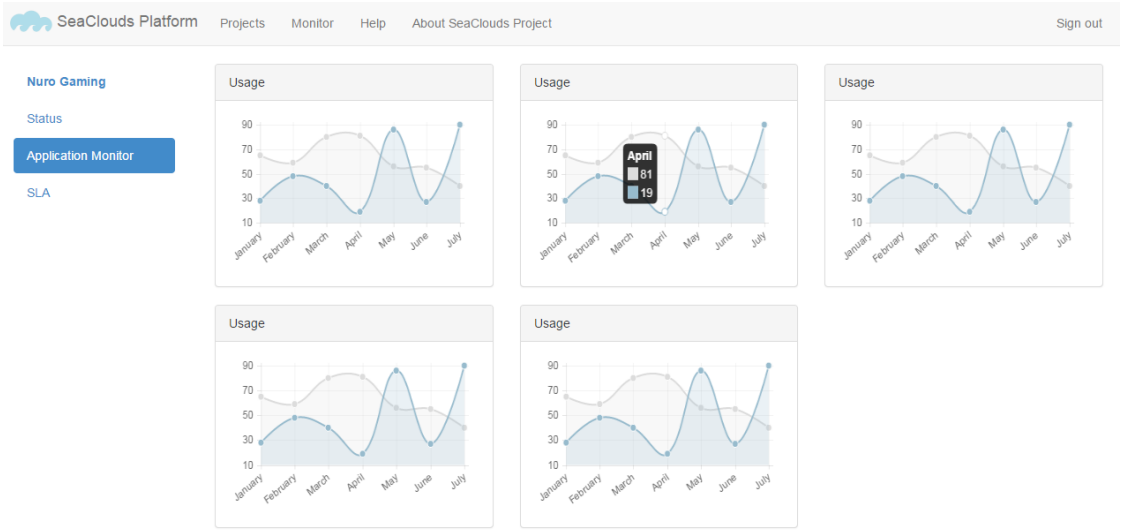

Figure 16. Application monitor.

In order to retrieve the data associated with this view, the dashboard internally queries the API to retrieve what metrics are available for the selected application (getAvailableMetrics). After that, the monitor has all the available metrics and the

dashboard registers himself as an observer of the metrics to receive notifications (addObserver) from the Monitor, and updates the graphs based on the metric values (Figure 17).

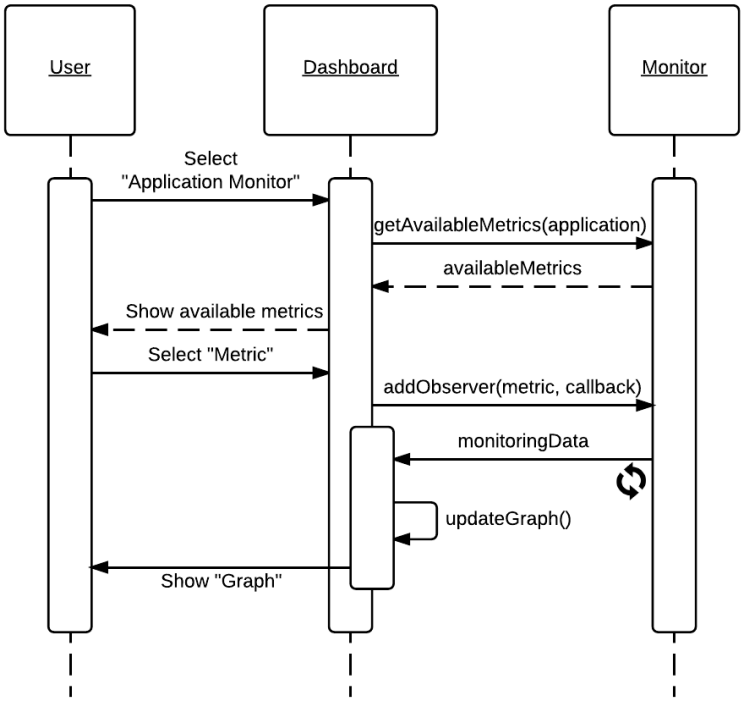

Figure 17. Dashboard application monitoring process.

3.2.5. SLA section

The SLA view allows to check at a glance how your application has been working. It shows the agreements for each provider, as we can see in Figure 18. Therefore, it details the current SLA accomplishment. It maintains a list of the succeed rule violations and a list of penalties as consequence of the aforementioned violations. In Figure 19 we can see how the Dashboard does it internally using the SLA.

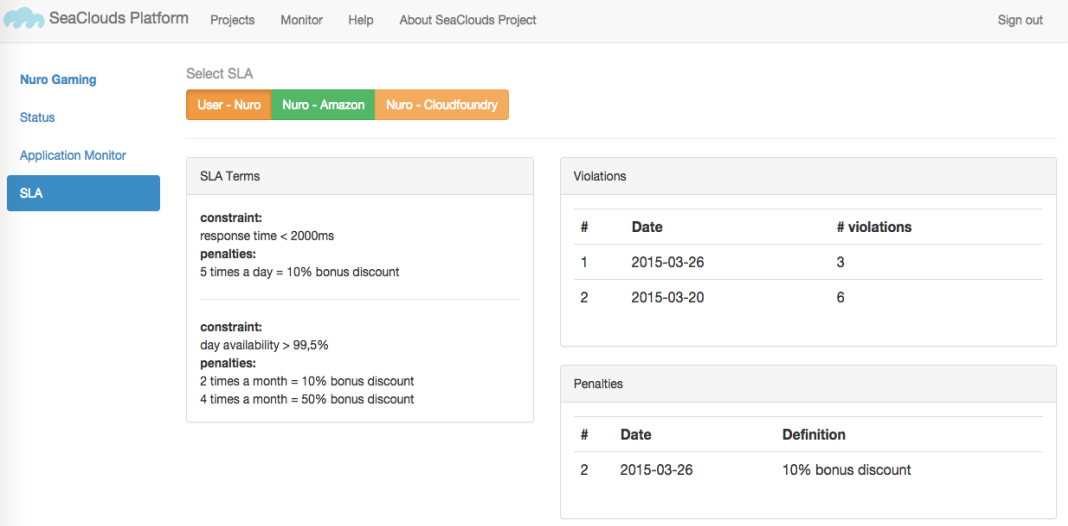

Figure 18. SLA usage and status summarize.

In order to generate the view the dashboard retrieves all the available agreements for the current application (getAgreements). Once the agreements are in the client, per each agreement the dashboard queries the SLA about the global status of the agreement (getAgreementStatus), together with historical data of the violations of QoB (getQoBViolations) and QoS (getQoSViolations) and their associated penalties (getPenalties).

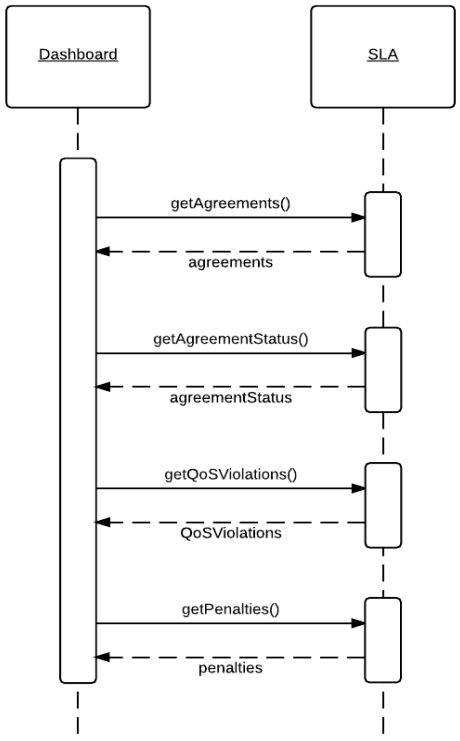

Figure 19. Interaction between the Dashboard and the SLA API.

3.3. Nuro Storyboard as use example of the API and Dashboard

Based on the Nuro Storyboard presented on previous deliverables [10], here we briefly mention how interacting with the Dashboard during the use case affects to the API. Note we are introducing here a brief explanation of the application of the API and Dashboard, since we consider the images provided in the previous sections (mentioned here but not included as regards the case study to not repeat them) are enough explanatory.

In the Nuro Storyboard the user, Christian, wants to deploy his existing application by using SeaClouds. The process starts opening the New Application Wizard where the user deploys his application following the procedure illustrated in the Section 3.2.3.1.

The deployment plan is generated by SeaClouds, and after the deployment process finishes, Christian could check how the application is working by using the Monitor view inside the project view (Section 3.2.43.2.2). This contains an overview of the most important application metrics for Christian.

Some violations of any QoS, SLA may occur at runtime. Christian also can plan to expand his business, so he will probably need to be aware of this problem.

In order to ensure that the application will work as expected, Christian goes to the Status view (Section 3.2.2), click the desired module and edit the module properties. He can add a new policy associated with a reconfiguration in case of happening some violation of the application.

4. Conclusions

This document has been structured in two main topics, the first one introduced the SeaClouds API from a high level point of view, by describing the functionality that each SeaClouds Component exposes via REST. This description has been focused on explaining the most important functionality as it is not possible to define all the helper methods because the API is still under development. The second one showed a brief overall of the SeaClouds Dashboard, focused on how the Dashboard interacts with the API.

In order to illustrate the relationship between the API and Dashboard the last part of the document showed the application of the API and Dashboard ovre the Nuro Storyboard.

5. References

1. SeaClouds Project. Deliverable D5.2.2. Final Design of the User Interface (SeaClouds Consortium), March 2015 (to be published).

2. SeaClouds Project. Deliverable D3.2. Discovery, design and orchestration functionalities (SeaClouds Consortium), March 2015 (to be published).

3. Hromkovic, Juraj, Algorithmics for Hard Problems: Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics. 2010, Springer-Verlag.

4. Model-driven approach for design and execution of applications on multiple clouds, Project is partially Funded by European Commission Grant no. FP7-ICT-2011-8-318484, 2013-2015, https://github.com/deib-polimi/modaclouds-monitoring-manager/wiki

5. SeaClouds Project. Deliverable D4.4. Dynamic QoS Verification and SLA Management Approach (SeaClouds Consortium), March 2015 (to be published).

6. Bootstrap: A framework for developing responsive, mobile first projects on the web, http://getbootstrap.com/ 2015.

7. Angular JS: HTML enhanced for web apps, https://angularjs.org/ 2015.

8. SeaClouds Project. Deliverable D4.2. Cloud Application Programming Interface (SeaClouds Consortium), September 2014 http://www.seaclouds-project.eu/deliverables/SEACLOUDS-D4.2-Cloud_Application_Programming_Interface.pdf

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 D2.4 Final SeaClouds Architecture (SeaClouds Consortium), February 2015 (to be published).