SeaClouds Project

D3.3 - SeaClouds discovery and adaptation components prototype

Project Acronym	SeaClouds
Project Title	Seamless adaptive multi-cloud management of service-based applications
Call identifier	FP7-ICT-2012-10
Grant agreement no.	610531
Start Date	1st October 2013
Ending Date	31st March 2016
Work Package	WP3 SeaClouds Design-Time modelling and orchestration
Deliverable code	D3.3
Deliverable Title	SeaClouds discovery and adaptation components prototype
Nature	Prototype
Dissemination Level	Public
Due Date:	M22
Submission Date:	31st August 2015
Version:	1.0
Status	Final
Author(s):	Marc Oriol (UPI), Diego Perez (Polimi), Simone Zenzaro (UPI), Mattia Buccarella (UPI), Javi Cubo (UMA)
Reviewer(s)	Francesco D'Andria (ATOS)

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	08/07/15	First ToC	Marc Oriol (UPI)
0.2	10/07/15	First contributions	Marc Oriol (UPI)
0.3	14/07/15	Second contributions	Marc Oriol (UPI)
0.4	15/07/15	Added the optimizer part	Diego (Polimi)
0.5	20/07/15	contributions/check on technical details and synchronization	Diego (Polimi), Simone Zenzaro (UPI), Mattia Buccarella (UPI), Javi Cubo (UMA)
0.6	09/08/15	Final contributions before internal review	Marc Oriol (UPI)
0.7	25/08/15	Revision	Francesco D'Andria (ATOS)
1.0	01/09/15	Final changes and formatting.	Marc Oriol (UPI)

Table of Contents

List of Figures

Executive Summary

This deliverable describes the implementation of the discovery, design and orchestration functionalities. Firstly, we describe and architectural overview of the SeaClouds platform for both design time and execution time. Then we go in depth in the components involved on these activities, namely the Discoverer and the Planner, describing first its architecture and then their technical details accompanied with sequence diagrams and their API specifications.

The code of such implementation is released under Apache 2.0 license and can be downloaded from the SeaClouds Platform github repository https://github.com/SeaCloudsEU.

1. Introduction

1.1. Scope and outcome of the Deliverable

This deliverable describes the implementation of the discovery and adaptation components of the SeaClouds Platform, accompanied with the architectural definition, the relationship between components, and their technical details.

The SeaClouds project is an open source software released under Apache 2.0 license and the released prototype can be downloaded from the SeaClouds Platform github repository https://github.com/SeaCloudsEU.

1.2. Glossary of Acronyms

Acronym	Definition
AAM	Abstract Application Model
ADP	Abstract Deployment Plan
API	Application Programming Interface
DAM	Deployable Application Model
IaaS	Infrastructure-as-a-Service
PaaS	Platform-as-a-Service
QoB	Quality of Business
QoS	Quality of Service
SLA	Service Level Agreement
SLO	Service Level Objective
TOSCA	Topology and Orchestration Specification for Cloud Applications
UML	Unified Modelling Language
URL	Uniform Resource Locator
WP	Work Package
YAML	YAML Ain't a Markup Language

Table 1. Acronyms

2. Architectural Overview

This section describes the architectural overview of the SeaClouds platform. In Figure 1 we illustrate the current version of the architecture which was refined from Deliverable D5.1.2 [1] considering the implementation and design decisions taken during the development of the platform.

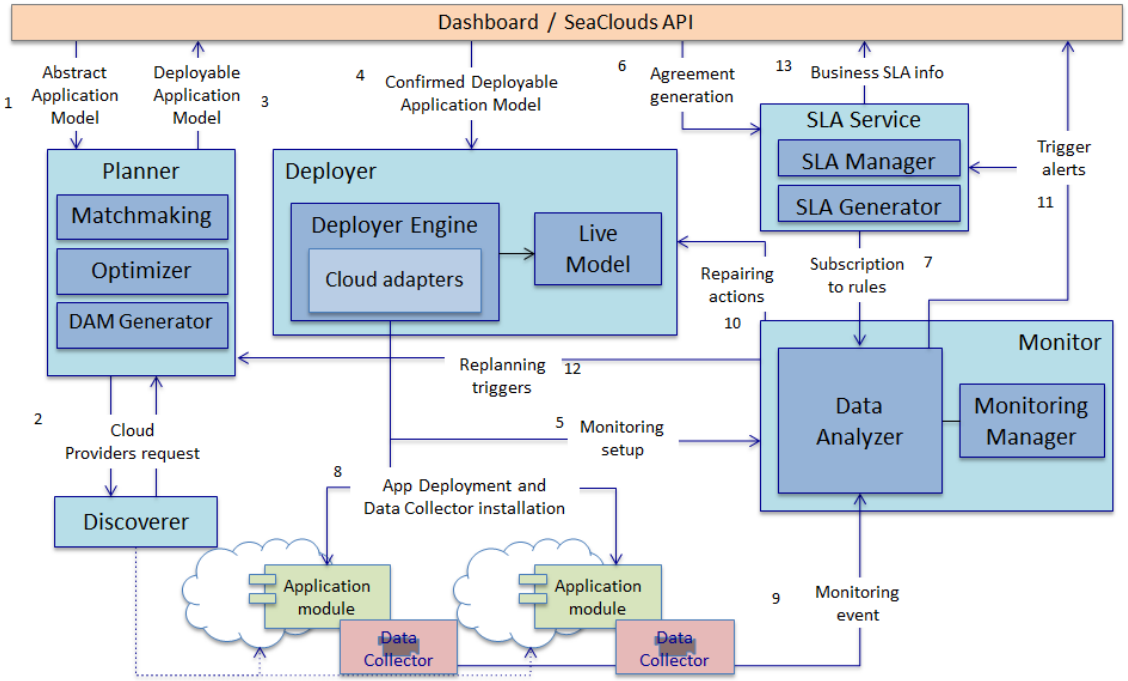

Figure 1. SeaClouds Architecture

Below we describe the architecture focusing on the Planner and Discoverer components for both design and runtime execution.

Design time

SeaClouds is orchestated by the Dashboard/SeaClouds API. The initial input for SeaClouds is an abstract application, which is instantiated by the user and described through an Abstract Application Model (AAM) [2]. This model contains the definition of all the modules of the application, their relationships, and the user's requirements. These requirements are both technical and QoS requirements that may apply to the whole application and/or to the constituent modules. Then, the planner interacts then with the Discoverer, which provides a list of Clouds Offerings from service providers with information regarding their technical characteristics and QoS information.

These models are processed by internal components of the Planner, namely the Matchmaker and the Optimizer, which generate as output a list of Abstract Deployment Plans (ADPs). In an ADP, the different modules of the cloud application are instantiated by concrete services that provide the functionality required, meeting the technical and QoS requirements.

The list of ADPs are sent back to the user in order to let her choose the most suitable cloud composition to her interests. Once the ADP has been selected, it is sent to the DAM Generator which augments the information specified in the ADP and generates a Deployable Application Model (DAM). The DAM contains the information needed by the SeaClouds Deployer to deploy, configure and execute the application (e.g. with all the required information about credentials).

Execution time

During execution, a Live Model keeps track of the status of all application's modules and it is used for supporting the dynamic evolution of the application. The Live Model exposes the deployment status process to the Deployer API. If there is a violation on the QoS, SeaClouds executes a repairing action (e.g. by scaling Virtual Machines). If such repairing action is not capable or restoring the required QoS a replanning action is triggered. In this case, the Planner generates a new DAM. Details about repairing policies at execution time of the SeaClouds platform are described in [3].

In the following sections we describe in detail the implementation of the Discoverer and Planner components

3. Discoverer

3.1. Architecture & Design

The discoverer component of SeaClouds is responsible of providing the planner with information about the available cloud offerings for both IaaS and PaaS from several cloud providers. In order not to rely on a single source of information, the architecture of the discoverer is modular.

The main module of the Discoverer, the core, aggregates the information from several modules and exposes it to the rest of the SeaClouds platform. The different modules are responsible of converting the information from heterogeneous sources in different formats

into the standard TOSCA YAML format for cloud offerings, which will then be consumed by the core. The modular approach allows also for easy extension of the discoverer if some of the used sources of information becomes unavailable and must be replaced. Figure 2 depicts the High-level architecture of the Discoverer as defined in [2].

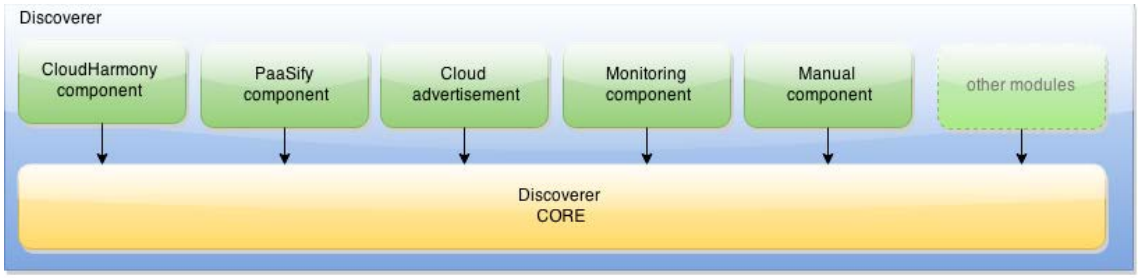

Figure 2. High-level architecture of the Discoverer

The different modules are described below:

3.1.1. CloudHarmony component

CloudHarmony [4] is a repository of cloud offerings of different nature. The information stored in CloudHarmony is mapped to the required properties that are defined in the cloud offering TOSCA YAML model used by SeaClouds. CloudHarmony provides a query API which allows the information to be fetched in JSON format. The discoverer CloudHarmony spider fetches this data, converts it into TOSCA YAML format and stores it in the Discoverer CORE.

3.1.2. PaaSify component

PaaSify [5] is the second directory that SeaClouds relies on for populating the local repository of cloud offerings. The PaaSify web site focuses mostly on two main strength points: i. searching the repository can be performed by adding filters that shrink the size to the suitable offerings according to the specified characteristics; ii. the accuracy of the fetched data is hardly wrong because the repository is filled manually by users within the community or by the administrator of the offered clouds himself.

From the SeaClouds perspective, PaaSify will be used to populate the local repository of cloud offerings, basically by getting the JSON files from to the PaaSify profile directory, then convert them in TOSCA YAML, in such a way that they can be used locally by SeaClouds. The PaaSify JSON files can be retrieved through Github.

3.1.3. Manual component

Even if automated systems are available, manually introducing cloud offerings information should be kept as an option. Manually maintaining a list of cloud offerings may in some cases have lower cost than maintaining the software for automatic detection. On the other hand, some technical information may not be available in a machine readable format for automatic detectors to read. Under these circumstances, the manual intervention of an administrator could be preferred.

3.1.4. Cloud advertisement

After launching the SeaCloud platform, we expect the cloud providers to be interested in having their own offers listed in the SeaCloud discoverer directory, in order to increase their visibility. We propose a model for which cloud providers can notify the SeaCloud platform of updates to their offering, and provide the information in the TOSCA YAML format for the Cloud Advertisement component. We see two possible strategies for this interaction: the first one is the PUSH approach, where the cloud provider submits a list of their updated services whenever changes occur, the second is the PULL approach, where the cloud providers register a URL for their offers, with the updated list of cloud offering that can be fetched when needed.

The PUSH approach has the advantage that the cloud providers do not have to setup a URL with the purpose of advertising to SeaClouds, whereas the PULL approach allows the cloud providers to advertise automatically to other frameworks beyond the scope of SeaClouds.

3.1.5. Monitoring component

SeaClouds monitoring components will be deployed on various cloud services. The monitoring results of these components can be used to as an alternate source of data for some of the properties of the cloud offerings stored in the repository, in order to complete or validate the information already present. The monitoring component of the discoverer can get this information from the live model and update the data of the repository accordingly.

3.2. Implementation

During the implementation phase, the aforementioned high-level architecture of the discoverer has been refined to a technical low-level architecture, which is depicted in Figure 3. The code of the Discoverer is available at https://github.com/SeaCloudsEU/SeaCloudsPlatform/tree/master/discoverer.

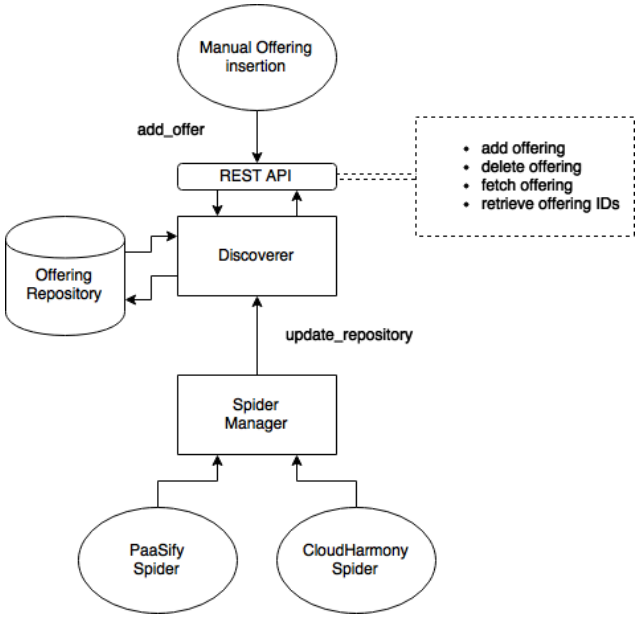

Figure 3. Discoverer technical architecture

The different modules are implemented as spiders that crawls the information from the different sources and converts them into TOSCA YAML format. Currently the ones implemented are the PaaSify Spider and CloudHarmony Spider. These spiders are managed by a Spider Manager, which continuously updates the repository of the discoverer. The Discoverer also offers a REST API which is used for both (1) manually insert cloud offerings directly in TOSCA YAML and (2) retrieve the cloud offerings of the repository.

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. In Figure 4 we

show the sequence diagram of how the Spider Manager interacts with the different components of the discoverer.

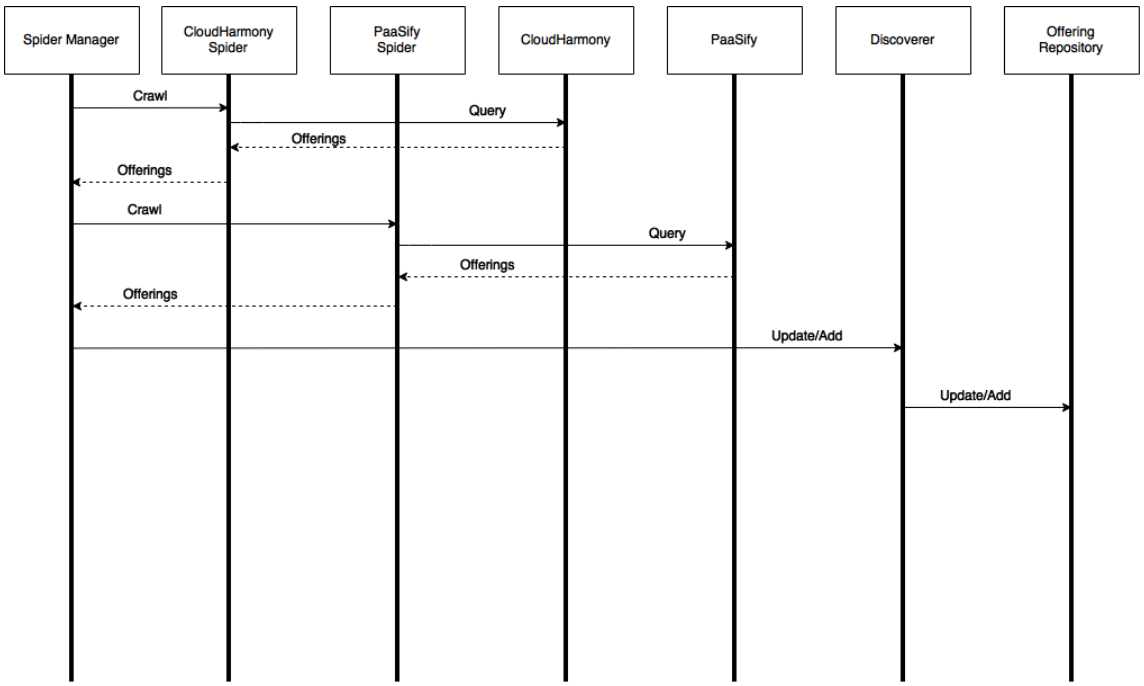

Figure 4. Sequence diagram of the Discoverer modules

3.3.1. Discoverer API

The Discoverer provides the following API though its web service layer:

- Add cloud offering: adds a cloud offering to the discoverer directly in TOSCA YAML format.
- Delete cloud offering: removes a cloud offering from the discoverer
- Fetch offering: obtains the properties of the cloud offering
- Retrieve offering IDs: obtains the list of offerings IDs to allow the iteration of the cloud offerings and their properties from the information system. (used by the matchmaker)

ID	addOffer
Description	Adds a cloud offering to the discoverer.
Parameters	**(CloudOfferingDocument) cloudOffering**, a cloud offerings in YAML format to be included into the discoverer.
Response	**(String) offer_id**, the identifier of the cloud offering added in the discoverer.

ID	delOffer
Description	Removes a cloud offering of the discoverer.
Parameters	**(String) offer_id** a cloud offering ID to remove from the discoverer.
Response	**(Boolean) Success**, true if the database was updated successfully.

ID	fetchOffer
Description	Get the definition of a cloud offering given its identifier.
Parameters	**(CloudOfferingId) cloudOfferingId**, the unique id of the cloud offering.
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	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.

4. Planner

4.1. Architecture & Design

The Planner component is in charge of providing a set of deployment plans that define where each application module will be deployed. Given an AAM the Planner generates a set of ADPs that meet the requirements specified by the user.

The generation of a deployment plan consists of two steps:

1. matchmaking the suitable offerings for each module;
2. optimizing the set of suitable offerings reducing the number of possible configurations.

The planner architecture is composed, as shown in Figure 5, by three modules: Matchmaker, Optimizer and DAMGenerator.

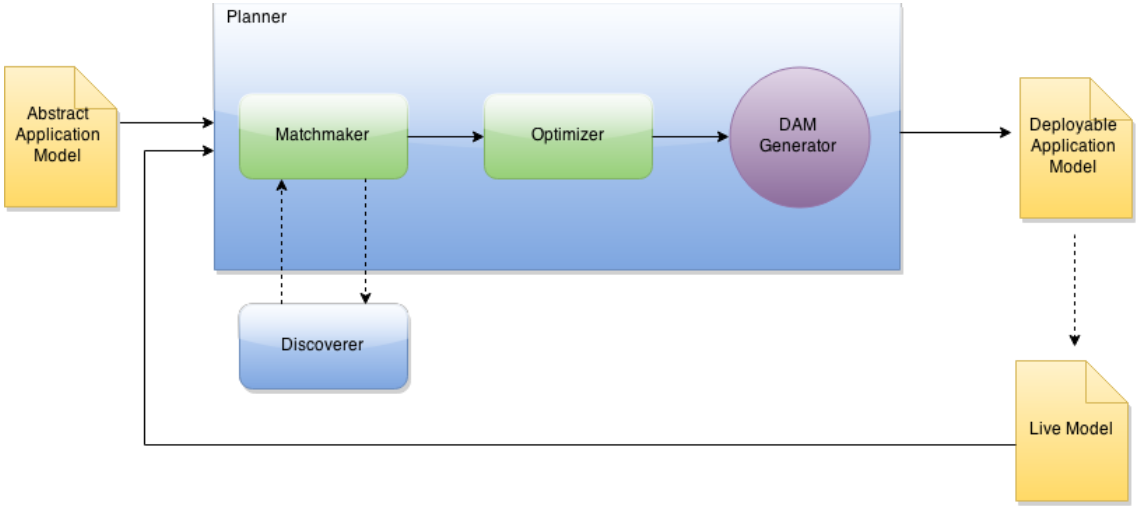

Figure 5. Planner Architecture.

At design time, in order to plan the deployment of an application, the Planner component requires as input an AAM. At run time, for replanning, the Planner requires also the information about the Live Model. The planner expose *plan* and *replan* as the two main functionalities that, respectively address the first planning of an application and the replanning process.

Figure 6 and Figure 7 show the sequence diagram for the planning process. The dashboard triggers the planner requiring its planning functionality and giving the AAM for the application to be planned. The planner calls the Matchmaker that gets (a stream of) cloud offerings from the discoverer and looks for the suitable offerings for each AAM module. When the matchmaking process ends, the Planner requires the optimization step from the Optimizer that will select a set of optimal deployment proposal. Finally this set of deployment proposal, instrumented as ADPs, are sent back to the user who is in charge of choosing the most promising plan among the set of optimized ADPs. Finally,the DAM generator generates the DAM that has all the information required to deploy the system. Currently the DAM Generator is in charge of augmenting the ADP with information regarding: cloud credentials, monitoring information and SLAs by invoking different services that provides such information.

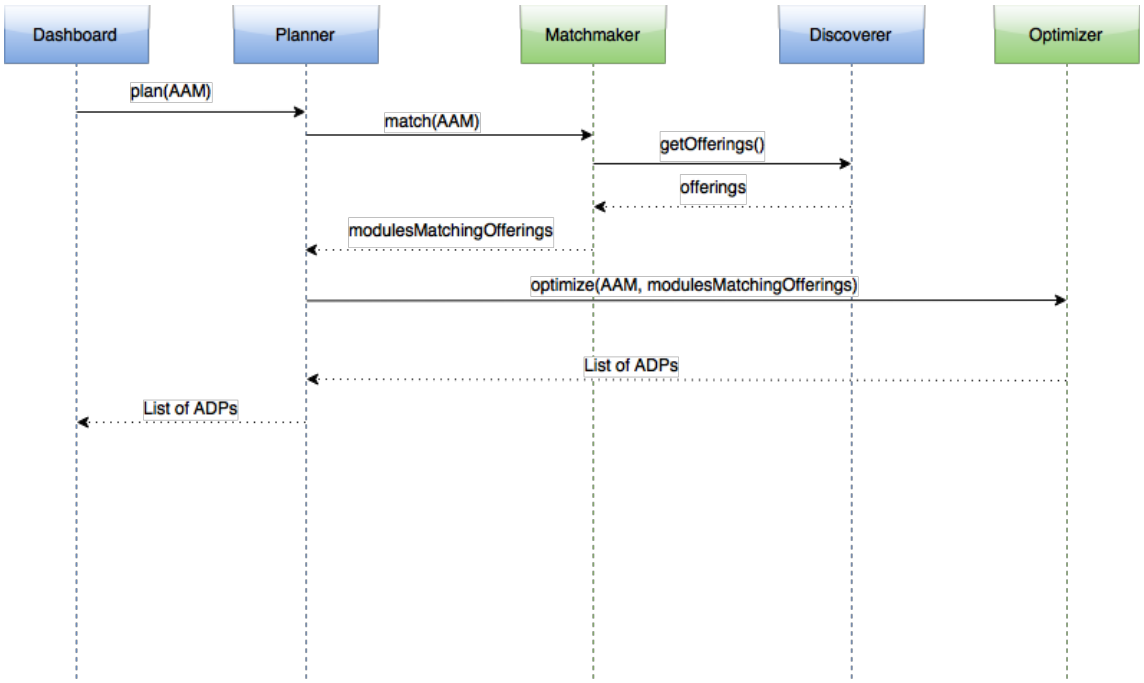

Figure 6. Sequence diagram of planning to generate a list of ADPs

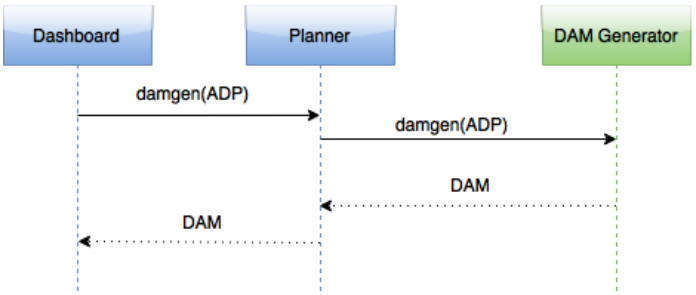

Figure 7. Sequence diagram of planning to generate the DAM

After the first successful deployment of an application with SeaClouds, it can happen that for different reasons (e.g. monitoring or SLA violations) a replanning is required. The process of replanning an application is similar to planning but, having already deployed the application, the Planner can leverage the Live Model information to better optimize the deployment proposals. In particular the Live model contains information about the actual deployed application and the causes that triggered the replanning process. Figure 8 and Figure 9 shows the sequence diagrams for replanning.

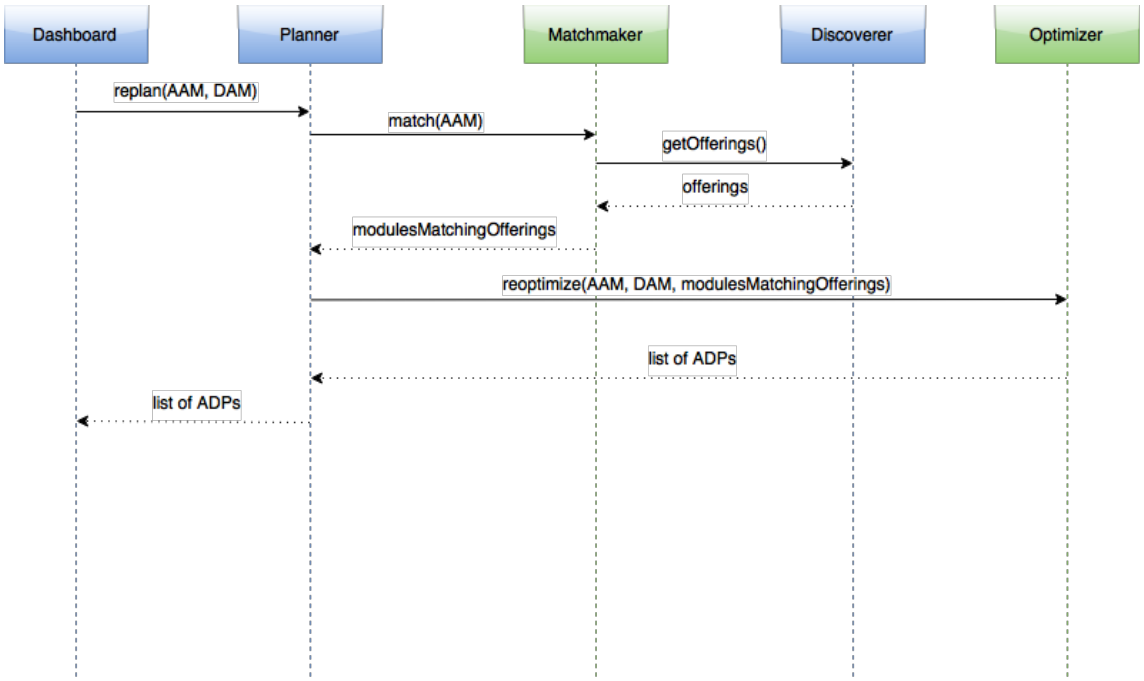

Figure 8. Sequence diagram of replanning to generate the list of ADPs

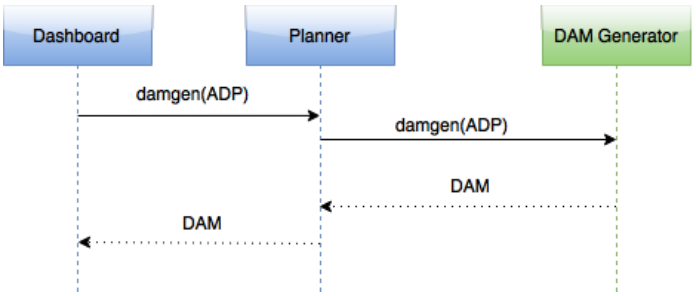

Figure 9. Sequence diagram of planning to generate the DAM

4.2. Implementation

In this section we describe the available methods that are offered from the Planner API and its subcomponents. The code of the Planner is available at https://github.com/SeaCloudsEU/SeaCloudsPlatform/tree/master/planner.

3.3.2. Web service layer

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	damgen
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 **(DAM) 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

3.3.3. 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.

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.

3.3.4. Optimizer

SeaClouds planning activity provides an optimization step where it is searched the solution - among the ones that are able to satisfy all the requirements given by the user- that is expected to furnish the best trade-off between its performance, availability and cost properties. In case of replanning, the also optimization process also takes into account an

additional parameter for the trade-off study: the "number of migrations" required to change the application deployment from its current deployment to the one represented in the candidate solution.

The current prototype includes the following search-based optimization methods: *hill-climbing,* which finds local optimum; *simulated annealing,* which performs a partial exploration of the search space aiming to find the global optimum; and *blind-search*, which works as a random search of different candidate solutions and keeps the best one. *Blind-search* has been implemented as initial proof-of-concept of the Optimizer module, and currently it acts as a baseline to evaluate the goodness of the rest of methods implemented.

Interface

The interface of Optimizer methods was defined in D4.5 [6]. The following tables retrieve them in order to complete some details that lacked in the previous API description in D4.5.

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 serialized 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 serialized in TOSCA YAML 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 serialized 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 in TOSCA YAML 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 serialized in TOSCA YAML, 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.

4. How to get and install the SeaClouds Integrated Platform

SeaClouds project has a Continuous Integration chain in place. This allows to have all the binaries produced by each software component of SeaClouds to be always available from https://oss.sonatype.org/content/groups/public/eu/seaclouds-project/.

The consortium has identified Apache Brooklyn as the tool to easily deploy SeaClouds. We currently support deployments against [Bring Your Own Nodes (BYON)] and to all the IaaS provider supported by Apache jclouds[1].

In the following subsections we show how it is possible to deploy the SeaClouds platform both on a local computer and on the cloud.

4.4 Local Deployment

The deployment of SeaClouds on a local computer is supported to allow users experimenting with the platform.

To simplify the creation of the nodes needed to deploy SeaClouds, a convenient Vagrantfile has been created for the end-users. Make sure you have Vagrant[2] and Apache Brooklyn[3] installed, then:

```
cd $HOME
git clone git@github.com:SeaCloudsEU/seaclouds-distribution.git
cd seaclouds-distribution
./setup
```

Please make sure you have configured BROKLYN_HOME at least in the current terminal.

```
vagrant up
```

This spins up a virtual environment, made up of 2 VMs, which are accessible at `192.168.100.10` and `192.168.100.11`.

Start Apache Brooklyn

```
nohup $BROOKLYN_HOME/bin/brooklyn launch &
```

This starts up your instance of Apache Brooklyn on your workstation, accesible at http://localhost:8081.
Please double-check in nohup.out the correct url.

[1] http://jclouds.org
[2] https://www.vagrantup.com/
[3] https://brooklyn.incubator.apache.org/

Finally, copy and paste SeaClouds blueprint[4] to deploy the SeaClouds platform on the 2 VMs created by Vagrant previously.

4.5 Launching in the clouds

The previous deployment option has to be considered non-production ready: it is a great way to start with SeaClouds with no effort and get familiar with the main concepts. Of course, deploy SeaClouds on the cloud is more interesting if an organization wants to support it in production. By simply editing the location pre-specified on the seaclouds blueprint, it'd be possible to deploy SeaClouds against any IaaS provider supported by Apache Jclouds

For example, instead of:

```
location:
  byon:
    user: vagrant
    privateKeyFile: ~/git/seaclouds/seaclouds-distribution/seaclouds_id_rsa
    hosts:
    - 192.168.100.10
    - 192.168.100.11
```

one could instead use:

```
location: jclouds:softlayer:ams01
```

To provision the 2 hosts on demand on the IBM SoftLayer cloud provider in the datacenter in Amsterdam.

5. Conclusions

In this deliverable we have described the implementation of the discovery, design and orchestration functionalities. Such implementation will be continuously updated from now until the end of the project, following the continuous integration approach we have adopted

[4] https://github.com/SeaCloudsEU/seaclouds-distribution/blob/master/seaclouds.yaml

6. References

[1] SeaClouds Project Team, Public Project Deliverable. "D5.1.2. – Integrated Platform", available at http://www.seaclouds-project.eu/deliverables/SEACLOUDS-D5.1.2-IntegratedPlatform.pdf, April 2015.

[2] SeaClouds Project Team, Public Project Deliverable. "D3.2. - Discovery, design and orchestration functionalities", available at http://www.seaclouds-project.eu/deliverables/SEACLOUDS-D3.2%20Discovery_design_and_orchestration_functionalities.pdf, March 2015.

[3] SeaClouds Project Team, Public Project Deliverable. "D4.6. Prototype and detailed documentation of the SeaClouds run-time environment components", to be published, 2015.

[4] CloudHarmony web page https://cloudharmony.com/ (last retrieved July 2015)

[5] Paasify Comparative and Supported Providers. http://www.paasify.it/vendors (last retrieved July 2015)

[6] SeaClouds Project Team, Public Project Deliverable. "D4.5. - Unified dashboard and revision of Cloud API"