SeaClouds Project

D2.2 Initial architecture and design of the SeaClouds platform

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	WP2. Requirements Analysis, overall Architecture and Standardization
Deliverable code	D2.2
Deliverable Title	Initial architecture and design of the SeaClouds platform
Nature	Report
Dissemination Level	Public
Due Date:	M9
Submission Date:	7th July 2014
Version:	1.0
Status	Final
Author(s):	Javier Cubo, Ernesto Pimentel, Jose Carrasco and Francisco Duran (UMA), Antonio Brogi, PengWei Wang and Michela Fazzolari (UPI), Elisabetta Di Nitto and Raffaela Mirandola (POLIMI), Christian Tismer (NURO), Roman Sosa and Francesco D'Andria (ATOS), Alex Heneveld and Andrea Turli (Cloudsoft)
Reviewer(s)	Francesco D'Andria (ATOS), Andrea Turli (CloudSoft)

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)	

Table of Contents

List of Figures

List of Tables

1. Executive summary

This deliverable, D2.2, aims to study and analyze the early architecture and design of the SeaClouds platform. The initial requirements for the SeaClouds platform identified in deliverable D2.1 [16] and also the case study described in deliverable D6.1 [17] are considered to obtain the initial version of the architecture and design for the SeaClouds platform, which will be required to get the final conceptual SeaClouds reference platform.

This report (to be delivered in month M9) will be continued and extended with a new and final version of the SeaClouds architecture in the deliverable D2.4 Final SeaClouds architecture (to be delivered in month M16).

The structure of this document is the following:

- Section 2 presents the scope of this document, lists the reference documents to generate the deliverable, and describes the methodology used to obtain the Architecture.
- Section 3 discusses the main objectives and challenges of SeaClouds, as well as how the positioning and progress proposed as regards initiatives and standards presented in Annex A.
- Section 4 provides a direct relationship between the requirements specified in the deliverable D2.1 and the SeaClouds platform functionalities.
- Section 5 introduces an early architecture of the SeaClouds platform, as well as the components that implement each functionality specified in the platform, with the corresponding details of description.
- Sections 6 and 7 present the SeaClouds case studies: SoftCare application use case and Cloud Gaming use case, and relate each one with the SeaClouds architecture.
- Section 8 concludes the document.
- Annex A presents the current initiatives and standards for Cloud Computing management and interoperability.

2. Introduction

This section introduces the deliverable D2.2 and presents its scope in the project. In addition, the methodology used to obtain the SeaClouds architecture is described, as well as an overview of the document.

2.1 Scope and outcome of the Deliverable

SeaClouds project works towards giving organizations the capability of "Agility After Deployment" for cloud-based applications by taking care of different aspects of cloud development life-cycle such as developing an open, generic and interoperable foundation that enables orchestrating parts of applications, and a layer to monitor, manage and migrate the underlying providers (both public and private PaaS/IaaS) based on informed SLA compliance decisions to guarantee performance and QoS on multi-cloud environments. Then, in order to achieve the main goal of the SeaClouds project, a first version of the SeaClouds architecture need to be delivered.

This document is focused on the specification of the initial architecture and the design of the SeaClouds platform. To obtain this early architecture, both the requirement specification document presented in D2.1 and the extended description of the case studies described in D6.1, have been analyzed in detail. This document is crucial for the rest of the project, since the main technical work packages (WP3, WP4 and WP5) will develop the corresponding components required for the SeaClouds platform according to the initial architecture and design (and also considering the modifications in the final version) of the SeaClouds platform. In Figure 1 are depicted the interactions among Task 2.2 (in which this deliverable D2.2 is generated) and tasks corresponding to other WPs.

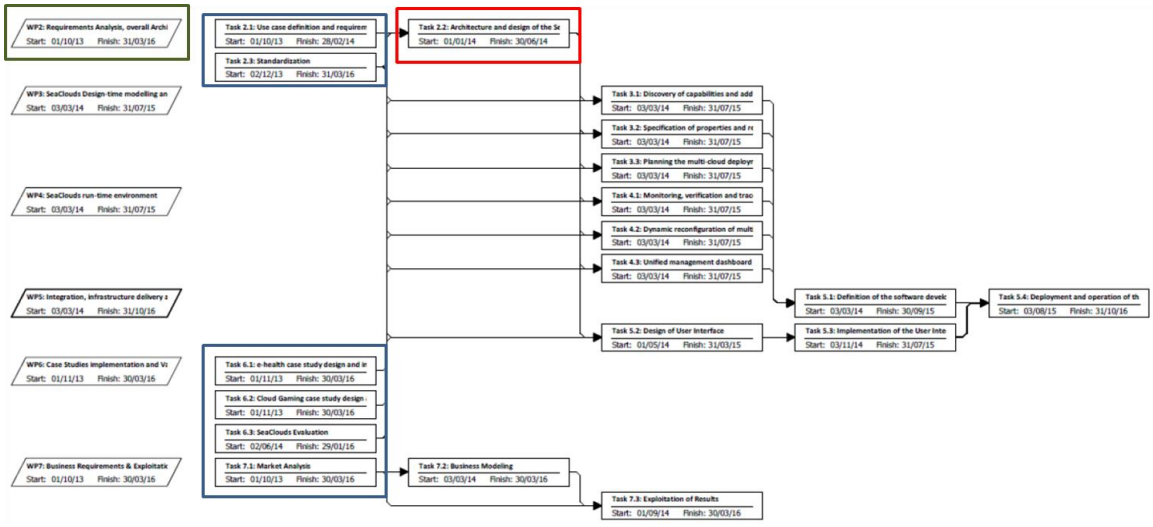

Figure 1. Interaction among WPs and relation of D2.2 with other deliverables

In this document, we intend to detail the architecture of the SeaClouds platform, detailing each component composing the platform and their functionalities. Thus, each component will be presented and detailed with its role, the corresponding inputs, outputs, interdependencies with other components, the success criteria, and the delivery date.

This deliverable will also provide the correspondences between the current architecture of the case studies to validate the project and the SeaClouds architecture, with the purpose of applying the SeaClouds platform over this use cases.

The development of the SeaClouds Reference Architecture constitutes the first step towards the creation of the system that will fulfil the SeaClouds vision.

SeaClouds provides the foundation for allowing **"Agility After Deployment"** providing necessary tools and a **framework** for **Modelling**, **Planning** and **Controlling Cloud Applications**.

SeaClouds answers questions such as: How can a **complex cloud application** be deployed, managed and monitored over **multiple and heterogeneous infrastructures Clouds**? How can the underlying cloud providers be **monitored** to check for quality of service compliance? How can applications be **reconfigured** if any problem or deviation from normal execution patterns is detected in any component at run time?

2.2 Methodology

The methodology used to generate the design of the SeaClouds Reference Architecture follows a set of steps as outlined below:

1. **Review state-of-the-art.** In Annex A are detailed the different works, initiatives and standards to be used as baselines, and over which SeaClouds will advance and contribute.
2. **Description of case studies.** In deliverable D6.1 are described the case studies, which is required to obtain the requirements and functionalities to be offered by the SeaClouds platform.
3. **Analysis of requirements**. In deliverable D2.1 a list of functional and non-functional requirements, as well as the business goals and the constraints are analyzed.
4. **Definition of use cases.** In D2.1 a list of use cases to be used is presented, which is needed to obtain the stakeholders and final functionalities for the SeaClouds platform.

5. **Mapping of the requirements to architectural components.** In this deliverable, it will be created descriptions (functionality offered and interoperation) for the SeaClouds components, including the interactions and inputs/outputs, and considering the use cases.
6. **Design of the initial SeaClouds Reference Architecture.** Once the previous steps have been performed, then the SeaClouds Reference Architecture will be detailed, whose design should satisfy both functional and non-functional requirements and the interoperation issues.
7. **Implementation of the SeaClouds components.** In WP3 and WP4 the different components will be implemented obtaining the proof-of-concepts. This is part of the methodology since we will need the individual components to be later integrated in the initial version of the platform, which will be analyzed to check if some update is required in the final version of the architecture as regards the initial one delivered in this document.
8. **Integration of the SeaClouds components**. In WP5 the components will be integrated by using a software developing environment, and obtaining the platform integration of the prototypes of the components. As in the previous step, this integration is needed to check the architecture is appropriate to the solution expected by the SeaClouds platform.
9. **Evaluation and validation of the architecture.** The initial (and the next and final version) architecture will be will be evaluated based on evaluation scenarios in WP6.

2.3 Overview of the Deliverable

The rest of this deliverable is organized as follows.

In Section 3, we discuss the main objectives and challenges of SeaClouds, as well as how the positioning and progress proposed as regards initiatives and standards presented in Annex A.

Section 4 provides a direct relationship between the requirements specified in the deliverable D2.1 and the SeaClouds platform functionalities.

In Section 5, we introduce an early architecture of the SeaClouds platform, as well as the components that implement each functionality specified in the platform, with the corresponding details of description.

Sections 6 and 7 presents the SeaClouds case studies: SoftCare application use case and Cloud Gaming use case, and relate each one with the SeaClouds architecture.

Section 8 concludes the document.

Finally, Annex A presents the current initiatives and standards for Cloud Computing management and interoperability.

2.4 Glossary of Acronyms

Acronym	Definition
SaaS	Software-as-a-Service
PaaS	Platform-as-a-Service
IaaS	Infrastructure-as-a-Service
QoS	Quality of Service
SLA	Service Level Agreement
TOSCA	Topology and Orchestration Specification for Cloud Applications
CAMP	Cloud Application Management for Platforms
GUI	Graphical User Interface
API	Application Programming Interface
APP	Application
DB	Database
HDD	Hard Disk Drive
RAM	Random Access Memory

Table 1. Acronyms

3. Challenges and Positioning of SeaClouds

In this section, we discuss about the main objectives of the SeaClouds project and how SeaClouds will address challenges and specific objectives. Also we present the positioning of SeaClouds as regards existing initiatives and standards in Cloud Computing management and interoperability.

3.1 Challenges and Objectives of SeaClouds

The main objective focuses on the development of a novel platform that performs a seamless adaptive multi-cloud management of service-based applications. And the specific objectives of SeaClouds are the following (they can be also found in the DoW):

- O1) Orchestration and adaptation of services distributed over different cloud providers.
- O2) Monitoring and run-time reconfiguration of services distributed over multiple heterogeneous cloud providers.
- O3) Providing unified application management of services distributed over different cloud providers.
- O4) Compliance with major standards for cloud interoperability.

For every specific objective, SeaClouds consortium plans to achieve certain challenges, which are described in this section.

Challenges in orchestration and adaptation for the cloud.

Objective O1 will be addressed by developing adaptive orchestrators of cloud-based application modules. Orchestrators are widely used in the service-oriented computing paradigm [1-11], mainly focusing on behavioral and context-aware adaptation of services, by coordinating the interactions between different services. In this context, services could be dealt as cloud resources of modules of a complex cloud application.

Several approaches exist that target formal adaptation of orchestrated services (detailed in Annex A), but, to the best of our knowledge, none of these approaches has been extended to the cloud environment. Challenges such as heterogeneity of cloud platforms and migration to different cloud providers have still to be addressed.

SeaClouds will address the following challenges in order to extend service-oriented approaches to the cloud: (i) adaptation could be needed to take into account cloud provider characteristics and Service Level Agreements (SLA), (ii) violations of Quality of Service (QoS) properties need to be monitored across different cloud platforms (this

challenge will be also tackled in the next objective), and (iii) dynamic architecture reconfiguration might involve migrating some components of the application to other cloud providers at runtime.

Challenges in monitoring of services on multiple clouds.

In order to address objective O2, SeaClouds' monitoring will use and enhance existing monitoring functionalities for the PaaS and IaaS levels: (i) With respect to the IaaS level, SeaClouds will reuse what is available (e.g., Brooklyn[1] or the EU project MODAClouds; both described in Annex A), and (ii) With respect to the PaaS level, SeaClouds aims at augmenting the set of metrics currently available from Cloud4SOA (described in Annex A), such as response time and up-time; and also to analyse the functionality of MODAClouds which could be used in SeaClouds.

For both the IaaS as the PaaS level, SeaClouds aims at coordinating, monitoring and aggregating monitoring information at the single service level to fulfill the purposes of whole orchestration. Thus, SeaClouds aims at: (i) being able to monitor each application component, and (ii) combining and aggregating the above mentioned data to highlight performance problems and their impact.

Challenges in unified application management of services distributed over different cloud providers.

SeaClouds will use existing management functionalities to address objective O3. Specifically, (i) SeaClouds' discovery functionality may use and extend existing matchmaking functionalities to match application requirements with PaaS offerings (in principle, SeaClouds is thinking on a basic matchmaking service), and (ii) SeaClouds management will use a REST harmonized API for the deployment, management and monitoring of complex cloud-based applications across different and heterogeneous cloud PaaS offerings.

SeaClouds intends to use Brooklyn's policy-driven functionality to integrate support for IaaS providers. Moreover, Brooklyn's approach to policy modeling and enforcing can provide guidance for SeaClouds' orchestration/adaptation and management functionality.

On the other hand, Brooklyn only targets the IaaS level and has no support for orchestration. Beyond what Brooklyn provides, SeaClouds will therefore extend policy-

[1] Brooklyn (http://www.brooklyn.io) is an open source, policy-driven control plane for distributed applications delivered by CloudSoft (a member of the SeaClouds consortium).

driven functionality to the PaaS level and also add support for adaptation and orchestration.

Challenges in standards for cloud interoperability.

SeaClouds intends to actively contribute to standards to achieve the objective O4.It plans to contribute to the standardization effort of CAMP [12] (see Annex A) both by exploiting CAMP-compliant interfaces provided by PaaS providers, and by contributing review proposals that will possibly emerge while specifying properties of SeaClouds orchestrations, adaptations and monitoring. CAMP is an OASIS initiative, and Brooklyn is an implementation following the CAMP specification.

SeaClouds will exploit the TOSCA [14] (see Annex A) specification to drive the design of the model for specifying cloud service orchestrations. In doing so, SeaClouds might actively contribute to the standardization effort of TOSCA, by contributing review proposals that will emerge while trying to devise TOSCA-compliant instances of the SeaClouds service orchestration model. TOSCA is an OASIS initiative, and OpenTOSCA[2] is an container implementation following the TOSCA specification.

On the other hand, SeaClouds will also focus on developing functionalities that are deliberately out of the scope of TOSCA to solve issues about policies for the dynamic management of service orchestrations, which currently we are analysing. Although the currently available implementations of TOSCA and CAMP do not yet support the management of complex applications over multiple clouds, SeaClouds will work towards building such management on top of them.

Due to a common partner in both initiatives, CAMP and TOSCA, (CloudSoft), Brooklyn can also benefit from integrating SeaClouds' functionalities, especially regarding the integration of adaptation techniques in supported policies.

3.2 Positioning SeaClouds

Figure 2 illustrates how SeaClouds intends to relate to the initiatives and standards presented in Annex A with the purpose of achieving the goals and challenges previously described.

In the figure, we can observe in the top layer the main components will be generated in SeaClouds, and in the other layers we observe the relationships between every

[2] OpenTOSCA (http://www.iaas.uni-stuttgart.de/OpenTOSCA/indexE.php) is an open source browser-based TOSCA runtime environment for running TOSCA-based applications, delivered by the Institute of Architecture of Application Systems (IAAS), University of Stuttgart.

component of SeaClouds and the existing efforts, in order to perform the specific objective of our platform.

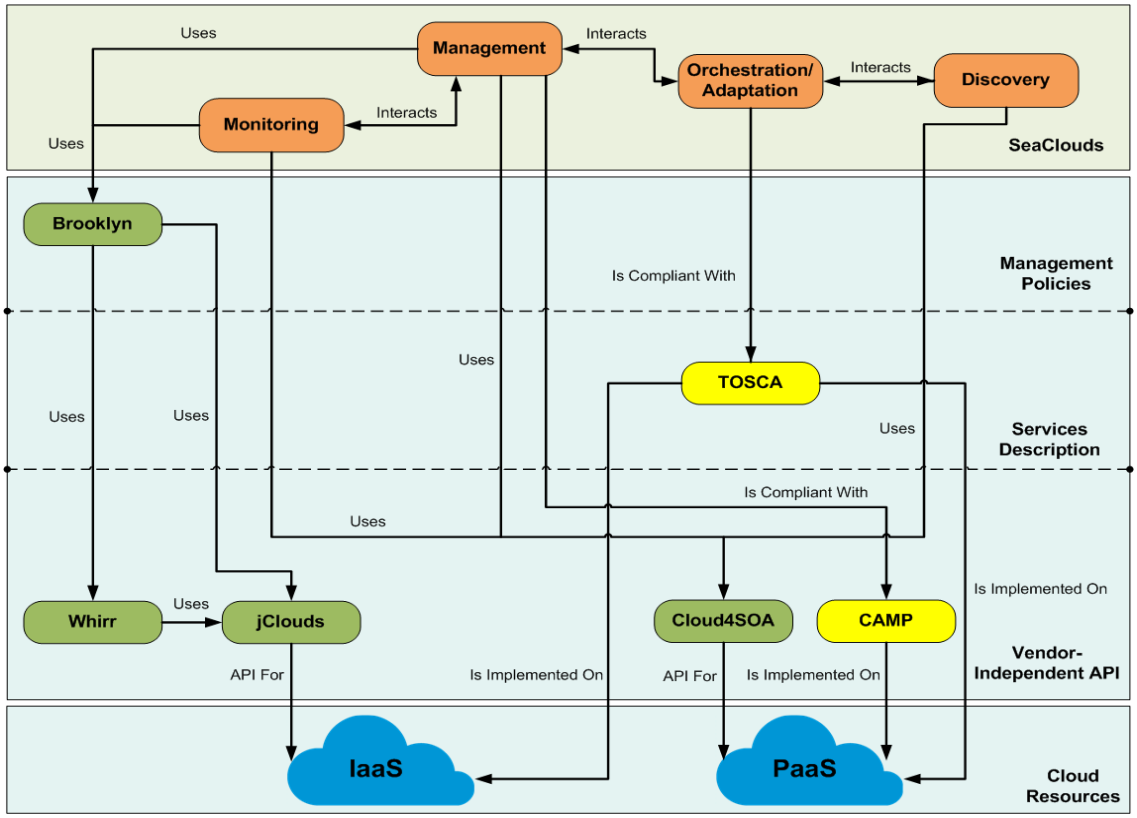

Figure 2. Position of SeaClouds with respect to related initiatives

4. SeaClouds Platform Stakeholders and Functionalities from the Requirements

Analyzing the requirements described in the Deliverable D2.1 [16], we can devise the following stakeholders and functionalities for the SeaClouds platform.

Stakeholders of the SeaClouds Platform:

- The ***Application Administrator*** oversees the correct execution of the application.
- The ***Application Designer*** designs the Application as an orchestration of services and interacts with the SeaClouds Platform through the comprehensive GUI to obtain a deployment plan.
- ***Cloud Providers*** provide the Cloud Resources (which offer some Cloud Capabilities). They do not necessarily interact directly with the SeaClouds platform, but the services offered are exploited by the platform to run service compositions.

SeaClouds requirements and associated use cases:

The table below provides a summary of the requirements defined in D2.1 and the associated use cases. The table contains eight requirements of which two are general ones that do not have a corresponding use case but encompass all uses of the SeaClouds platform. These are the need for a *Comprehensive Graphical User Interface* (Requirement 5) and the fact that *The SeaClouds platform must rely on standard APIs and languages* (Requirement 6).

The other requirements are fulfilled by the functionalities that will fulfill the eight use cases listed in the tables. More details can be found in D2.1.

Requirement ID	Short name	UseCase ID	Short name
Requirement5	Comprehensive Graphical User Interface		
Requirement6	The SeaClouds platform must rely on standard APIs and languages		
Requirement8 Requirement1	Orchestration Specification reuse to build different plans QoS dependent resource definition	UseCase1	Create a Deployment Plan
Requirement2	Service Level	UseCase2	Define Service Level

	Agreement Definition		Agreement
Requirement3	SLA Assessing and Violation Management	UseCase3	Manage Service Level Agreement
Requirement4 Requirement3	Metric-Driven Policy-Based Management SLA Assessing and Violation Management	UseCase4	Monitor periodically
Requirement4 Requirement3	Metric-Driven Policy-Based Management SLA Assessing and Violation Management	UseCase5	Monitor on events
Requirement8	Orchestration Specification reuse to build different plans	UseCase6	Initialize Application Deployment
Requirement9	Application updates	UseCase7	Update Deployed Application
Requirement7	Application Migration	UseCase8	Application Administrator reconfigures the application deployed on multiple clouds.

Table 2. Summary of the requirements defined in D2.1 and the associated use cases

SeaClouds Platform functionalities:

From an analysis of the requirements and the case studies summarized above and described in detail in D2.1 the following functionalities emerge:

- *SeaClouds Planner:* functionality in charge of implementing planning policy to orchestrate the multi-cloud deployment of the application modules (this fulfills Use Cases 1 and 2).
- *SeaClouds Controller*: functionality in charge of implementing the multi-cloud deployment of the application modules and SeaClouds monitoring policy. It is composed of the SeaClouds Monitor (/Analyzer) and the SeaClouds Deployer (this fulfills Use Cases 3, 4, 5, 6, 7 and 8).
- *SeaClouds Deployer*: functionality in charge of taking as input the orchestration specification generated by the Planner, and deploying (by exploiting the Multi-Cloud Deployment API) the application modules on the specified clouds (this fulfills Use Cases 6, 7, and 8).

- *SeaClouds Monitor*: functionality in charge of monitoring (by exploiting the Monitoring API) that the QoS properties of the application modules are not violated by the clouds in which they were deployed, and that the whole application satisfies the QoS properties specified for the whole application (this fulfills Use Cases 4 and 5). In D2.1 it was considered a component, the *SeaClouds Analyzer*, separated from the Monitor, although some discussions have concluded wiht the inclusion of this component inside the Monitor. Then, also the Monitor, with the functionality of analyzing the violations, is in charge of generating the reconfiguration suggestions (if needed) to be passed as inputs to the Planner Component to trigger the generation of a new adaptive orchestration plan (this fulfills Use Cases 3 and 8).
- SeaClouds Discoverer: this component was not considered as a independent component in D2.1. However, it have been also discussed to be included in the platform as a separate component with an own entity, which is in charge of discovering available capabilities offered by cloud providers.

In the following we shortly describe how the SeaClouds stakeholders exploit the Seaclouds platform with its functionalities.

The SeaClouds Platform interacts with the Application Administrator and the Application Designer providing information about the status of the system as well as tools to orchestrate the deployment of Application Modules into the available Cloud Resources. SeaClouds Platform exposes SeaClouds API (Designer API, Discovery API, Monitoring API, and Deployment API) to support the Application Designer in the analysis of Cloud Capabilities offered by the available Cloud Resources, by using the SeaClouds Discoverer, and in the creation of an effective Orchestration Specification. The Orchestration Specification is expressed in TOSCA or CAMP and is generated by the SeaClouds Planner starting from the specifications provided by the Application Designer. The Orchestration Specification is exploited by the SeaClouds Controller (composed by the SeaClouds Monitor(/Analyzer) and the SeaClouds Deployer) to orchestrate the deployment of application modules to the available cloud resources.

The SeaClouds Platform monitors and analyzes the status of the Application to check the violation of QoS constraints, and support the process of migrating Application Modules distributed in heterogeneous Cloud Platforms. The SeaClouds Platform is able to manage Cloud Resources depending on QoS Requirements and other limits.

5. Initial SeaClouds Reference Architecture

This section describes the initial Seaclouds Reference Architecture and Design for the SeaClouds platform.

5.1 SeaClouds Reference Framework Requirements

Figure 3 shows the initial architecture of the SeaClouds platform, with the basic functionalities (in light blue), and relationships between the different components. Later, each component will be described and its functionalities will be detailed.

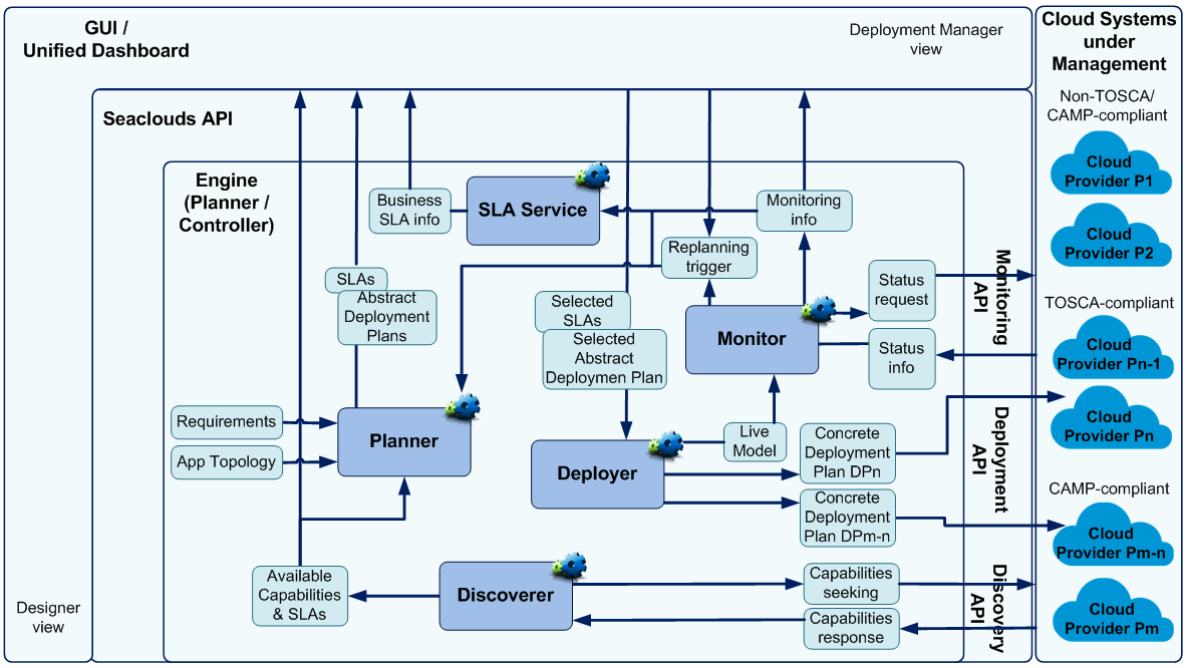

Figure 3. Initial Architecture of the SeaClouds Platform

5.2 SeaClouds Components implementing Functionalities

This section describes the components and services, their interactions, and the inputs/outputs of the platform, presented in Figure 3.

5.2.1. Discoverer Component

The following table describes the Discoverer component that allows discovering capabilities and add-ons featured by available clouds. The inputs/outputs of this component are described, as well as its interaction with the other SeaClouds components.

Component Name	***Discoverer***

Description/ Functionality	This sub-system is in charge of identifying the available capabilities offered by cloud providers that will be used by the Planner sub-system to perform a matchmaking process. Available capabilities can include both - Technology aspects, such as programming tools (programming languages, available frameworks, runtime environments, available database, database dimension, number of instances, etc.), possible add-ons, extensibility/scalability options and so on. - SLAs including QoS properties (bandwidth, monthly uptime percentage, etc) and the cost associated to each provided service. The trigger for this component is the time, in fact with a certain frequency it gathers information from the cloud providers. This information can be sent directly to the Planner and to the dashboard and/or can be used to create a shared repository that collects providers' characteristics.
Responsibilities	the data gathered by this component should be kept up-to-date, as the Planner sub-system relies on this information to match the requirements provided by a user with the available capabilities offered by cloud providers.
Constraints	-
Inputs	Cloud provider capabilities and desired SLAs.
Outputs	Cloud provider capabilities and SLAs to be sent to the planner and the dashboard.
Interactions and Interfaces	This component interacts with the Planner and the Dashboard. The Planner will consume the result of the discoverer to perform a matchmaking process and to decide where to deploy each application module according to its QoS and technology requirements. The Dashboard will present the result of the discoverer to the end- user. Optionally, this component could also receive automatic updates from cloud providers.
Implementation	Currently we are analysing the best way to implement this component, which in principle it will be implementing a basic matchmaking mechanism. Some considerations for the implementation are the following: this component relies on a data repository that will include a *profile* for each cloud

	provider. Each profile describes the QoS properties guaranteed by the cloud provider, the technology capabilities and the offered standard SLA. This repository can be accessed by the dashboard or it can be accessed by the Planner. This repository will be populated using the following strategies: ● Automatically by cloud providers: in an ideal situation each cloud vendor keeps its profile updated in an automatic way. The repository should be public and shared. ● By sending polling requests to cloud providers: in this case, the discoverer sub-system is responsible of asking the cloud providers for their capabilities and add-on to generate cloud profiles. ● By hand: if the previous ones are not feasible due to technological restrictions, the repository could be compiled by hand.
Platform dependency	none It mostly depends on the discovery API offered by each cloud providers.
Success Criteria	The cloud-agnostic discovery layer, which discovers a set of distinctive and overlapping properties, enables comparison and matching of different providers' offerings. The user does not need to worry about the choice of a specific provider which support the programming tools he/she uses, but he/she can focus on development issues.
Delivery date	● M12, discovery functionalities: first specification and early prototype. ● M18, discovery functionalities: complete specification, formal definition and final documentation. ● M22, SeaClouds prototype: final integrated prototype.

Table 3. Discoverer Component description

5.2.2. Planner Component

The following table describes the Planner component that is in charge of generating a distribution of application modules onto available clouds. The inputs/outputs of this component are described as well as its interaction with the other SeaClouds components.

Component Name	***Planner***

Description/ Functionality	The SeaClouds Planner is in charge of determining a distribution of application modules onto multiple available clouds so that the QoS properties and other technology requirements needed for individual application modules are not violated. The planner's input are the QoS and technology, the application topology, formulated by an application designer, the adaptation rules, written from an application administrator, and the discovered capabilities and SLAs coming from the Discoverer component. Using this complex inputs set, the planner will generate a set of abstract deployment plans, each of which describes a feasible distribution of application modules onto available clouds, and satisfies all the QoS properties and technology requirements required by the application designer. These abstract plans and related SLAs are returned to the application administrator, to support his decision. Once the application administrator selects an abstract deployment plan, it is then passed to the Deployer and Monitor components simultaneously, and to the Planner itself. With this, the Deployer will instantiate a concrete plan to actual deploy the application modules. During runtime, the Monitor will collect information about the execution of the application. Any QoS violation is detected and notified to the application administrator, so that she can decide whether to accept to replan. If so, the replanning trigger will be passed to the planner. Then, the planner tries to replan, generates a new set of abstract deployment plans and passes them to the deployment manager again. On the other hand, whenever new cloud capabilities are discovered by the Discoverer, the application administrator also can initiate a replanning trigger. With these new cloud capabilities, the planner then generates new abstract deployment plans.
Responsibilities	It can input the requirements and application topology submitted by the application designer. Moreover, it can generate feasible abstract deployment plans that distribute application modules onto multiple available clouds, and ensure QoS and technology requirements are satisfied. Finally, it allows the application administrator to initiate a replanning

	process.
Constraints	The application topology is specified by exploiting a standard specification (e.g., mainly we will use OASIS TOSCA and we will continue studying CAMP for this end). The application designer must provide the application modules that compose the application, together with the set of inter-module relationships (e.g., Module A *communicates with* Module B, Module A *is hosted on* Module B). For each (group of) application modules the application designer should provide a set of technology and QoS requirements.
Inputs	QoS properties and technology requirements, application topology, adaptation rules, available capabilities and SLAs, and replanning trigger.
Outputs	Abstract plans and related SLAs.
Interactions and Interfaces	The Planner receives the requirements and application topology from the application designer. It interacts with the Discoverer component to acquire the available capabilities and SLAs. It generates a set of abstract deployment plans and returns them to deployment manager. It receives replanning trigger from the Monitor component, and also from deployment manager.
Implementation	According to the requirements and application topology, the Planner tries to distribute the application modules onto multiple available clouds, in a convenient way, and ensures that the QoS properties and technology requirements are satisfied. This problem can be reduced to a matchmaking and optimization problem, which have been widely studied in the service-oriented computing paradigm, especially in the topics of service discovery and QoS-aware service selection and composition. Therefore, we can learn from these proven methodologies, combined with the actual situation and characteristics of cloud environment, and then propose some simple and effective methods for such distribution.
Platform dependency	It will depend on the outputs of Discoverer component
Success Criteria	A tool to plan the distribution of application modules in multiple available clouds.

Delivery date	• M12, discovery, design and orchestration functionalities: first specification and prototype. • M18, discovery, design and orchestration functionalities: complete specification, formal definition and final documentation. • M22, SeaClouds discovery and adaptation components prototype.

Table 4. Planner Component description

5.2.3. Deployer Component

The following table describes the Deployer component which interacts with platforms where the application will be deployed. This component generates a concrete plan and has the mechanisms to establish communication with the cloud providers in order to use and orchestrate the several services exposed by the platforms to instantiate the infrastructure components, call the different services and deploy the application components.

Component Name	***Deployer***
Description/ Functionality	The deployer is in charge of following the instructions coming as a deployment plan (CAMP-compliant) from the Planner. It is able to deploy the desired plan abstracting away the cloud-specific functionalities of the different cloud providers that is able to leverage, generating a concrete plan. The Deployer will generate a live model of the managed applications, storing the details about the deployed applications. It will have some healing capabilities to repair a managed application that is violating one or more constraints. Therefore, it has to monitor not only the deployment activities but also the running applications and report the failures.
Responsibilities	Since the plan is submitted to the Deployer sub-system, this has to execute the plan to deploy or reconfigure the application components in each target providers.
Constraints	This sub-system needs to consume the cloud provider API. It should be able to understand CAMP plans and to produce an application deployment on a multi-cloud environment.
Inputs	The plan describes the needed steps to deploy or reconfigure the application. This plan has to be approved by the

	deployment manager.
Outputs	A live model of the managed applications. A live model contains the components and services used by an application, the location for each of the application's component and the relationships among components and services.
Interactions and Interfaces	The deployer receives the plan from the planner. The sub-system interacts with the target platforms using the needed services.
Implementation	The Deployer component accepts a CAMP-compliant deployment plan. It parses the plan and orchestrate the deployment of each of the components/services described in the plan on the target cloud provider(s), re-using as much as possible a common abstract layer to consume IaaS and PaaS API. A storage system could be necessary to keep the credentials needed to authenticate to the different cloud providers. A parser has to be implemented, in order to analyze the deployment plan. A distinguishing aspect of the SeaClouds architecture is that it builds on top of OASIS standards initiatives and the deployer will initially use CAMP and we are planning to use the reference implementation of the standard, an Apache incubator project called Brooklyn, http://www.brooklyn.io. It is worth notice that the Deployer does not require cloud providers to be TOSCA or CAMP compliant, and it actually generates concrete deployment plans for non TOSCA/CAMP compliant providers as needed.
Platform dependency	This sub-system depends on the Planner. The Deployer component needs to generate a correct and concrete plan to instantiate and deploy the application modules.
Success Criteria	The expected result involves a correct plan execution to obtain an instantiation/reconfiguration of the application modules on the target environment.
Delivery date	• M12, definition of the monitoring strategies (D4.1) – to be included the multi-deployment strategies in this document. • M12, Cloud Application Programming Interface (D4.2). • M18, Unified dashboard and revision of Cloud API (D4.5). • M22, prototype and detailed documentation of the

	Seaclouds run-time environment components.

Table 5. Deployer Component description

5.2.4. Monitor Component

The Monitor component is in charge of timely collecting information on the whereabouts of the execution of an application. Once an deployment plan has been chosen, together with corresponding SLAs and adaptation rules, the monitor component is in charge of attaching to the modules to be deployed an appropriate infrastructure to collect the required information. These "attached observers" will provide continuous information which will be used to detect situations in which the application deployment requires a change. The information to be collected will depend on the adaptation rules to be considered. The information collected may be either directly provided to the GUI (either to directly show it or storing it in a DB for off-line analysis) or stored in a DB by the Monitor so that both the user and the Monitor could access it (in this way, more sophisticated adaptation rules might be considered). The following table describes the Monitor component.

Component Name	***Monitor***
Description/ Functionality	Given a set of SLAs (and probably adaptation rules) on an plan, the component is responsible of collecting the information read by the data collectors deployed with the components of the application. The monitor module is in charge of collecting and processing this information, forwarding it to the GUI after its processing for visualization and analysis by the administrator, and detecting situations in which replanning is necessary.
Responsibilities	This component assures the gathering of the monitoring information, and the support for its analysis for the appropriate implementation of the adaptation rules given (in case they are considered), taking into account the provided SLAs.
Constraints	Maximize performance and amount of collected data and minimize cost and performance degradation.
Inputs	Available capabilities and a set of the services in the selected deployment plan SLAs (and adaptation rules). Also, it has knowledge about the live model storing information about how the distribution of the application modules is done and

	deployed in cloud providers.
Outputs	Monitoring information and replanning triggers.
Interactions and Interfaces	The administrator provides, through the GUI, the deployment plan, the SLAs (and adaptation rules) to be used (either directly or choosing from a set of alternatives). The Monitor provides the collected monitoring information (or the results from its analysis) to the GUI for the administrator consultation, and triggers a replanning if problematic situations that require it are detected.
Implementation	In a first version, it is being considered to extend the Brooklyn monitoring functionalities. Also, a analysis of the functionalities of MODAClouds that could be included are being done. In this sense, a range of data collectors, for different platforms and contexts, and providing different types of information will be provided. All these data collectors will match a particular format, so that they adjust to the common monitoring infrastructure. This will allow us to later add further data collectors for new platforms/needs, giving support for potentially more powerful adaptation rules. The proposal in the MODAClouds project using RDF streams and CSPARQL is an interesting starting point.
Platform dependency	Concrete data collectors will be very dependent on the different cloud platforms. Since some providers supply tools are not available for other providers, the same data collectors cannot be used for all the platforms.
Success Criteria	The data collectors required for the main QoS properties will be provided, together with the support for the considered adaptation rules. Also, the analysis required by them will be implemented.
Delivery date	• M12, definition of the monitoring strategies (D4.1). • M16, design of the run-time reconfiguration process (D4.3). • M18, dynamic QoS verification (D4.4). • M22, prototype and detailed documentation of the Seaclouds run-time environment components.

Table 6. Monitor Component description

5.2.5. SLA Service

The SLA service is in charge of mapping the low level information gathered from the Monitor into business level information about the fulfillment of the SLA defined for a SeaClouds application. The following table describes the SLA service.

Component Name	***SLA Service***
Description/ Functionality	As in SeaClouds the service composition is very dynamic, we have designed this SLA Service with an SLA management framework to provide a generic end-to-end solution for SLA definition and operational management embracing multi-clouds services at IaaS and PaaS level. It provides an operational management with SLA composition and decomposition across functional and organizational Cloud domains; and covers the complete SLA and service lifecycle with consistent interlinking of planning and runtime management aspects; and can be applied to a large variety of industrial domains and use cases.
Responsibilities	The SLA Service is responsible for establishing, reviewing and cancelling of complex end-to-end- Service Level Agreements (SLAs) between Application Providers and Cloud Suppliers. It provides an operational management with SLA composition and decomposition across functional and organizational Cloud domains. It covers the complete SLA and service lifecycle with consistent interlinking of planning and runtime management aspects by implementing procedures and methods to evaluate and report Business Level Objectives.
Constraints	The SLA management strategy to be implemented should consider two well-differentiated phases: • The negotiation / preparation of the contract and • The monitoring of its fulfillment in real-time In the SLA context, Service Models refers to the resources associated with the service execution and relationship of these resources to each other, as well as the Cloud business/service level objectives and Key Performance indicator (KPI) and Key Quality Indicator (KQI) calculation used in the SLA.
Inputs	Monitoring info.
Outputs	Business SLA info.
Interactions and	Interactions with the rest of components. While the Planner

Interfaces	component will provide the inputs to create the SLA Agreements, the Deployer component will configure and set up the SLA Service at runtime. Finally, the Monitoring component will generate Business Metrics to evaluate agreements.
Implementation	Implementing procedures and methods to evaluate SLA agreements by relying on monitoring to report, respond and resolve SLA infringements.
Platform dependency	Service Level Agreements corresponding to cloud providers.
Success Criteria	Investigating new methods and tools for the SLA design and SLA Template Specification. Defining standards expression and meaning for SLA metrics and business concepts. Implementing a generic protocol to define the agreements, setting up targets and thresholds in the SLA (based, e.g., on its capabilities and feedback from its business).
Delivery date	• M18, dynamic QoS verification and SLA management approach (D4.4) – to be incorporated to this deliverable • M22, prototype and detailed documentation of the Seaclouds run-time environment components

Table 7. SLA Service description

Also, we consider in the SeaClouds architecture, the **GUI**, which features the graphical user interface (SeaClouds GUI) for two user roles (Designers and Deployment Managers). Application Designers use the GUI to provide a description of the topology of the application to be deployed, together with a set of requirements. These requirements can include QoS properties and technology requirements for the application modules. Deployment Managers instead, exploit the GUI through a unified dashboard that allows them to supervise the deployment and the monitoring of the application.

Finally, we have to design and generate the **SeaClouds API**, in order to define suitable application programming interfaces to allow the communication among the different SeaClouds components. Therefore, the above listed components are part of the **SeaClouds Engine**, and they interact with the SeaClouds GUI and with external systems through the SeaClouds APIs. SeaClouds is able to deploy, manage and monitor applications on clouds that are compatible with TOSCA and CAMP. Moreover, it will be compatible with a selected number of clouds (which will be studied in another deliverable).

6. SoftCare Application case study

In this section, we briefly present the adoption of the SeaClouds architecture as regards the SoftCare Application (this will be analyzed in deep in other tasks and documents).

ATOS aims at providing an e-health and social care case study by developing a full-featured business intelligence solution for assessing disease affecting elderly people.

The objective is to implement a Cloud-based social support network application that provides the following features:

- Supporting maintaining health and functional capability, through the risk assessment and the early detection of deterioration symptoms of the patients and distress signs of their carers;
- Providing the means for the self-care and the self-management of chronic conditions, through the development social networking as well as educational tools;
- Enhancing the home-as-care environment through the provision of user-friendly ICT tools for frequent, unobtrusive monitoring;
- Facilities for high-quality interaction between doctor and patients;
- Added-value features to create and maintain an easy-to-use web-based social network for individual elderly persons, to stimulate elderly person and their careers.

The SoftCare application is currently is a monolithic application (code written in one large program, and not modular) that has been installed/tested in the ATOS data centre.

As part of the work in SeaClouds, the application will be refactored as a Cloud-Enabled platform (the monolithic application will be decomposed into a set of web-based services ready to be cloudified), comprised of three main subsystems:

- web-based environment: social networking utilities, communication with carer, communication between the carers and medical personnel, educations tools.
- Monitoring system: development of smart devices for the conduction of remote psychometric tests; video-conferencing utilities.
- Risk assessment and analysis tools: data mining capabilities, retrieving information from psychometric tests, electronic health records, personal evaluations by medical experts, etc.

A draft of the desired architecture is depicted in Figure 4:

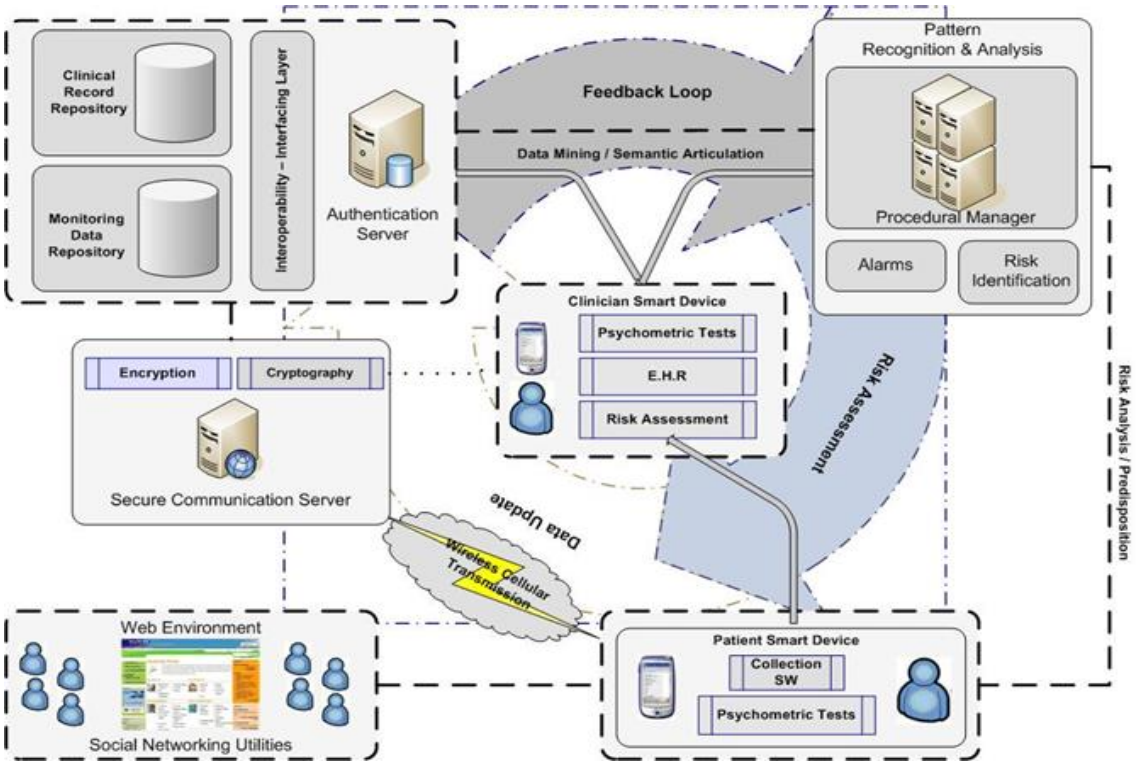

Figure 4. Architecture of the SoftCare Application using SeaClouds

Once the application is modularized, SeaClouds tools will help in the design, deployment, monitoring and governance of the SoftCare solution.

7. Cloud Gaming case study

In this section, we briefly present the adoption of the SeaClouds architecture as regards the Cloud Gaming case study (this will be analyzed in deep in other tasks and documents).

Nurogames has many games in the market and in deployment state, some of them based on Game Server Engines developed by Nurogames. The Game Server is responsible for data consistency and cheating protection.

NURO's cloud game case study is based on a Nurogames Engine (e.g. WebRpg or IsoGame) in the following called SeaCloudsGame. These engines are used for games that are online or will be launched soon.

NURO will modify its monolithic server approach to a cloud ready version. Also a testing system for different scenarios will be developed.

The Designer uses the SeaClouds System to describe all modules, needed to deploy the SeaCloudsGame.

This description consist of:

- type of module
- needed resources
- connection to other modules
- configuration, initialisation
- QoS and budget limits and how to handle violation

The early SeaClouds adoption for the Cloud Gaming application will consist on following modules:

Name	**Type**	**Resource**	**Connection to**	**Config, Init**	**QoS**
APP	Webservice with PHP	RAM: ca. 200 MB each worker	DB, CRON	Certificates, DNS, IP, DB creditals SVN:php_code.tgz	<=2sek per requests <=50€ per month
DB	MySQL compatible	HDD: 100GB growing	APP	Certificates, DNS, IP, SVN:dbdump.gz	<=50€ per month

CRON	crontab	wget	APP	Certificates, DNS, IP, crontab.txt SVN:skripts.tgz	<=???€

Table 8. Modules of the Cloud Gaming application

Figure 5 presents the desired adoption of SeaClouds as regards the Cloud Gaming Application.

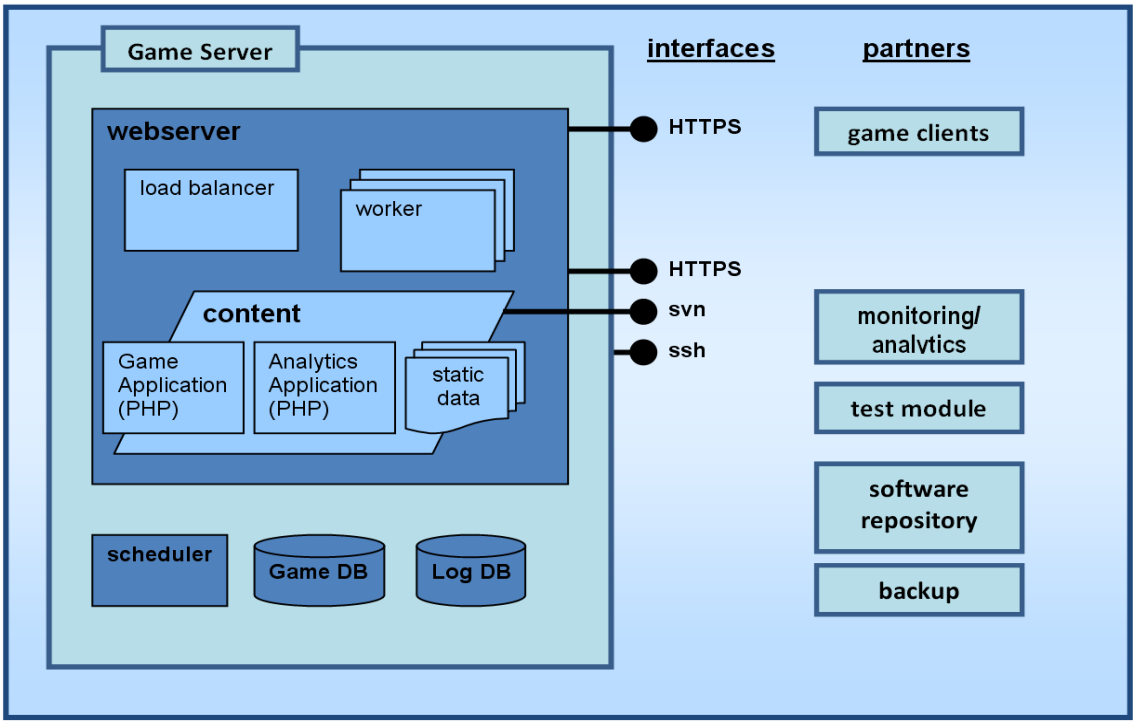

Figure 5. Adoption of SeaClouds in the Cloud Gaming Application

8. Conclusions

This deliverable presents the SeaClouds Reference Architecture and its main goal is to perform a seamless adaptive multi-cloud management of service-based applications. In order to achieve this main objective, SeaClouds addresses four challenges related to orchestration in the cloud, monitoring of services on multiple clouds, unified application management of services over different clouds, and standards for cloud interoperability.

We have detailed the different stakeholders and functionalities related to the SeaClouds framework by using the requirements analyzed in deliverable D2.1.

Next, we have defined the architecture framework as a reference to establish the components, their interactions, functionalities and inputs/outputs. The level of detail of this deliverable does not cover the thorough implementation design of the architecture components, since this information will be further provided within the context of work packages WP3, WP4, WP5, and WP6, which will use the architecture reference.

Finally, the desired adoption of the SeaClouds architecture as regards the two case studies (SoftCare Application and Cloud Gaming) have been provided. Anyway, this study will be analyzed more in deep in future tasks and documents during the lifecycle of the SeaClouds project.

Annexes

A. Baselines in Cloud Computing Management and Interoperability

This annex presents the current works, initiatives and standards for Service Composition and Cloud Computing management and interoperability.

A.1 Orchestration and adaptation in the cloud

Orchestrators are widely used in the services computing paradigm [1, 2, 3, 4, 5, 6, 7, 9, 10, 11], mainly focusing on behavioural and context-aware adaptation of services, by coordinating the interactions between different services. Several approaches exist that target formal verification and adaptation of orchestrated services, but, to the best of our knowledge, none of these approaches has been extended to the Cloud environment. This implies that extending such approaches to the Cloud is a progress beyond the state of the art. A cloud-compliant orchestration is not a trivial problem: challenges such as heterogeneity of Cloud platforms and migration to different Cloud providers have to be addressed, as well as the different standards emerging from distinct vendors. Therefore, existing approaches should be (substantially) extended to operate on heterogeneous Cloud providers. In the following we first present the state of the art in adaptation via orchestration, and subsequently the challenges that SeaClouds will address to extend this state of the art to the Cloud.

Restrictive approaches

A first class of existing works can be referred to as restrictive approaches [1, 3, 10, 11]. These approaches try to solve interoperability issues [5] by pruning the behaviours that may lead to mismatch, thus restricting the functionality of the services involved.

General limitations of restrictive approaches are: (i) lack of support for automatic adaptation at signature level, i.e. when operations present mismatches in their names or arguments, and (ii) weak support to enforce any properties beyond deadlock freedom.

Generative approaches

A second class of solutions, which can be referred to as generative approaches [2, 5, 6, 7, 8, 9], avoid the arbitrary restriction of service behaviour, and support the specification of advanced adaptation scenarios. Generative approaches build adaptors automatically from an abstract specification of how the different mismatch situations can be solved. These specifications are often referred to as the adaptation contract. Although generative approaches result in a more general and satisfactory solution while composing and

adapting services, writing the adaptation contract is a difficult and error-prone task. Adaptation contracts, which match the operations required by the services, may contain incorrect correspondences between operations in service interfaces or syntactic mistakes, which are particularly common in cases where the contract has to be specified using cumbersome textual notations.

In [6] is presented a toolbox that fully supports the adaptation process, including: (i) different methods to construct adaptation contracts involving several services; (ii) simulation and verification techniques which help to identify and correct erroneous behaviours or deadlocking executions; and (iii) techniques for the generation of centralized or distributed adaptor protocols based on the aforementioned contracts. The techniques proposed to support the adaptation contract construction drastically reduce the time spent to build the contract and the number of errors made during this process.

A.2 Monitoring of multi-cloud services

The EU FP7 Cloud4SOA project (www.cloud4soa.eu) provides an open source interoperable framework for application developers and PaaS providers. Cloud4SOA aims at supporting developers in deploying and monitoring their application with the ultimate objective of reducing the risk of vendor lock-in. The monitoring is based on unified metrics, but Cloud4SOA monitors each application separately and it is not able to aggregate monitoring results of multi-component applications.

The monitoring platform developed in MODAClouds overcomes the limitation of the one offered by Cloud4SOA by gathering data of various kinds from components, containers and cloud resources distributed and replicated on multiple clouds. The main characteristics of this monitoring platform are the following:

1. It supports distributed monitoring through the installation of proper data collectors that can probe data from various sources.
2. It enables monitoring at different abstraction levels (e.g., the hypervisor, the virtual machine, the PaaS-level container, the application code).
3. It allows application designers and operators to define monitoring rules that state the object of monitoring, the frequency of monitoring, the metric to compute (it could be the average value, the maximum value, etc), the set of values that are assumed correct, the action to be taken in case the monitoring platform identifies some incorrect value.
4. It offers proper APIs to develop new data collectors thus extending the kinds of information that can be acquired by the monitoring platform.

The open source software and additional information on the approach can be found here http://www.modaclouds.eu/software/monitoring/.

Several commercial and open source initiatives target monitoring of cloud applications. Often these initiatives address only particular platforms, for example Appsecute (http://www.appsecute.com) monitors only (open-source) CloudFoundry-based platforms. More platform-independent technologies are available for the IaaS level, since the latter has undergone a stronger harmonization effort. Deltacloud (http://deltacloud.apache.org) encapsulates the native API cloud provider to enable management of resources in different IaaS platforms, such as Amazon EC2. Rightscale (http://www.rightscale.com) supports monitoring several public (e.g., Amazon Web Services, Rackspace) and private IaaS clouds (e.g., CloudStack, Eucalyptus, OpenStack). Truly platform-independent monitoring solutions exist, the most known being NewRelic (http://www.newrelic.com). NewRelic achieves platform-independency by requiring each provider to implement a monitoring component and to integrate it in the offered cloud platform. On the one hand, this approach yields the best results from a monitoring point of view. On the other hand, it forces providers to invest quite some resources in order to implement the monitoring.

A.3 Unified management of multi-cloud applications

Brooklyn (http://www.brooklyn.io) is an open source, policy-driven control plane for distributed applications delivered by CloudSoft (a member of the SeaClouds consortium). It enables single-click deployment of applications across machines, locations and clouds. Then it continuously optimizes running applications to ensure ongoing compliance with policies. Brooklyn uses two open source tools to operate on cloud resources, Apache Whirr (http://whirr.apache.org) and Jclouds (http://www.jclouds.org), which support several IaaS providers. The already mentioned Cloud4SOA project also offers deployment and lifecycle management functionality using a harmonized API layer to encapsulate the providers' APIs.

A.4 Standards for cloud interoperability

CAMP (Cloud Application Management for Platforms) [12, 13] aims at defining harmonized APIs, models, mechanisms and protocols for the self-service management (provisioning, monitoring and control) of applications in a PaaS, independently of the cloud provider. However, CAMP is only a protocol specification, so it needs to be implemented by parties adopting the protocol.

The OASIS TOSCA (Topology and Orchestration Specification for Cloud Applications) [14, 15] aims at enabling the inter-operable description of application and infrastructure cloud services, the relationships between parts of the service, and the operational behaviour of these services, independently from the cloud provider.

By increasing service and application portability in a vendor neutral ecosystem, TOSCA aims at enabling portable deployment to any compliant cloud, smoother migration of existing applications to the cloud, as well as dynamic, multi-cloud provider applications.

A.5 Related Cloud initiatives

It is important to stress the fact that SeaClouds approach uses adaptation via orchestration and therefore it does not require code modifications to existing services. There are several initiatives and standards that target services deployed on the cloud, and aim at guaranteeing properties such as Quality of Service of those services. These initiatives use different approaches, with the consequence that software developers need to either use special APIs or programming models to code their applications, or to model them using project-specific domain languages. Some of these projects are mentioned in following.

The Broker@Cloud project (http://www.broker-cloud.eu/) aims at helping enterprises to transition to the cloud while enforcing quality control on the developed services. Capabilities for cloud service governance and quality control such as lifecycle management, dependency tracking, policy compliance, SLA monitoring, and certification testing are included in the project. Nonetheless, Broker@Cloud targets a brokering architecture, where the above mentioned services are available, and therefore cannot change the orchestration of the deployed services to adapt to changing conditions.

The MODAClouds project (http://www.modaclouds.eu/) also aims at providing quality assurance during the application life-cycle, support migration from cloud to cloud when needed, and techniques for data mapping and synchronization among multiple clouds. In order to do so, MODAClouds requires software developers to adopt a Model-Driven Development approach. This approach has therefore, differently from SeaClouds, an impact on the code that needs to be deployed on the cloud.

Also the PaaSage project (http://www.paasage.eu/) has Quality of Service as one of its goals. PaaSage also intends to match application requirements against platform characteristics and make deployment recommendations and dynamic mapping of components to the platform(s) selected for the application instantiation. Analogously to MODAClouds, it also requires the developers to adopt a modeling language in order to specify the model of the application.

The mOSAIC project (http://www.mosaic-cloud.eu/) aims at providing developers with vendor agnostic APIs, so that the resulting applications can be deployed on different IaaS using a sort of mOSAIC virtual machine. mOSAIC plans also to support SLA negotiation (with monitoring to detect SLA violations) and application life-cycle, but requires developers to adopt the project's API.

References

1. M. Autili, P. Inverardi, A. Navarra, and M. Tivoli. SYNTHESIS: A Tool for Automatically Assembling Correct and Distributed Component-based Systems. In Proceedings of the 29th International Conference on Software Engineering (ICSE ’07), pages 784–787. IEEE Computer Societyn (2007)

2. A. Bracciali, A. Brogi, and C. Canal. A Formal Approach to Component Adaptation. The Journal of Systems and Software, 74:45–54, 2005. Special Issue on Automated Component-Based Software Engineering (2005)

3. A. Brogi and R. Popescu. Automated Generation of BPEL Adapters. In Proc. of ICSOC ’06, volume 4294 of LNCS, pages 27–39. Springer (2006)

4. A. Brogi, J. Camara, C. Canal, J. Cubo and E. Pimentel: Dynamic contextual adaptation. In: In Proc. of Fifth International Workshop on the Foundations of Coordination Languages and Software Architectures 2006 (FOCLASA'06), vol. 175, no. 2, 81-95, Elviser (2007)

5. C. Canal, Poizat P., Salaun, G.: Model-Based Adaptation of Behavioural Mismatching Components. IEEE Transactions on Software Engineering 34, 546-563 (2008)

6. J. Camara, J.A. Martn, G. Sala □un, J. Cubo, M. Ouederni, C. Canal and E. Pimentel: Itaca: An integrated toolbox for the automatic composition and adaptation of web services. In: Proc. of International Conference on Software Engineering 2009 (ICSE'09),627-630, IEEE Computer Society Press (2009)

7. J. Cubo and E. Pimentel: Damasco: A framework for the automatic composition of component-based and service-oriented architectures. In: In I. Crnkovic, V. Gruhn, M. Book (editors), European Conference on Software Architecture 2011 (ECSA'11), Lecture Notes in Computer Science 6903, 388-404, Springer-Verlag (2011)

8. J.A. Martín, F. Martinelli, E. Pimentel. Synthesis of Secure Adaptors. JLAP, 81(2):99 – 126 (2012)

9. R. Mateescu, P. Poizat and G. Salaün. Adaptation of Service Protocols using Process Algebra and On-the-Fly Reduction Techniques. IEEE Transactions on Software Engineering, 2012, IEEE Computer Society Press, pp. 755-777 (2012)

10. H. Nezhad, B. Benatallah, A. Martens, F. Curbera, and F. Casati. Semi-Automated Adaptation of Service Interactions. In Proceedings of the 16th International Conference on the World Wide Web (WWW ’07), pages 993–1002. ACM (2007)

11. H. Nezhad, G.Y. Xu and B. Benatallah: Protocol-aware matching of web service interfaces for adapter development. In: Proc. of 19th International Conference on World Wide Web 2010 (WWW'10), 731-740, ACM (2010)

12. OASIS: CAMP 1.0 (Cloud Application Management for Platforms), Version 1.0. http://docs.oasis-open.org/camp/camp-spec/v1.1/camp-spec-v1.1.html/ (2012)

13. OASIS: Cloud Application Management for Platforms (CAMP) Technical Committee Charter. https://www.oasis-open.org/committees/camp/charter.php (2013)

14. OASIS: TOSCA 1.0 (Topology and Orchestration Specification for Cloud Applications), Version 1.0. http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.pdf (2012)

15. OASIS: OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) Technical Committee Charter. https://www.oasis-open.org/committees/tosca/charter.php (2013)

16. SeaClouds Project: Requirements for the SeaClouds Platform (SeaClouds Consortium). http://www.seaclouds-project.eu/deliverables/SeaClouds-D2.1-Requirements_for_the_SeaClouds_Platform.pdf (2014)

17. SeaClouds Project: Case Study extended description (SeaClouds Consortium). http://www.seaclouds-project.eu/deliverables/SeaClouds-D6.1-Case_study_extended_description.pdf (2014)