Project Acronym:	**OPTIMIS**
Project Title:	**Optimized Infrastructure Services**
Project Number:	**257115**
Instrument:	**Integrated Project**
Thematic Priority:	**ICT-2009.1.2 – Internet of Services, Software and Virtualisation**

Admission Control User Guide

Activity 3:	*Service Deployment*
WP 3.5:	*Admission Control*

Due Date:		M36
Submission Date:		07/06/2013
Start Date of Project:		01/06/2010
Duration of Project:		36 months
Organisation Responsible for the Deliverable:		ICCS/NTUA
Version:		1.0
Status		Final
Author(s):	Kleopatra Konstanteli Sotiris Stamokostas	ICCS/NTUA ICCS/NTUA
Reviewer(s)	George Kousiouris	ICCS/NTUA

Project co-funded by the European Commission within the Seventh Framework Programme		
Dissemination Level		
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	5/4/2013	Initial input compiled into the document	Sotiris Stamokostas (ICCS/NTUA)
0.2	15/4/2013	Updated all existing sections with new information	Sotiris Stamokostas (ICCS/NTUA)
0.3	30/4/2013	Completed version without ONE extensions	Sotiris Stamokostas (ICCS/NTUA)
0.4	30/05/2013	Included new sections about AC ONE module (interoperability component)	Kleopatra Konstanteli (ICCS/NTUA)
0.5	31/05/2013	Ready for submission	Kleopatra Konstanteli (ICCS/NTUA)
1	07/06/2013	Version for release	Malena Donato (ATOS)

Table of Contents

Index of Figures

Index of Tables

1 Introduction

This document serves as the user guide for the software component Admission Control final prototype of the OPTIMIS project. In more detail the document at hand focuses on:

- **Design and use cases:** A general overview of the Admission Control design, including high level objectives, and a description of the Admission Control module service-oriented architecture along with detailed sequence diagrams and use cases.
- **Functionality and interfaces:** A description of the functionality and the interfaces of each Admission Control component, the way they are incorporated into the overall OPTIMIS framework and their known limitations, including the interoperability extensions for the OpenNebula environment.

It should be noted that detailed instructions on how to install and use each AC component, including software dependencies and supported platforms as well as configuration instructions can be found in the Admission Control Installation Guide [3] and README.txt [4] and are not replicated here for the sake of brevity. Furthermore, in the Admission Control Scientific Report [2] one can find a detailed description of the optimization algorithm used behind the Admission Control Component.

1.1 Glossary of Acronyms

Acronym	Definition
AC	Admission Control
CO	Cloud Optimizer
CQoS	Cloud Quality of Service Agreement Factory
BARON	Branch and Reduce Optimization Navigator
CPU	Central Processing Unit
D	Deliverable
DRS	Document Review Sheet
EC	European Commission
GAMS	General Algebraic Modeling System
IP	Infrastructure Provider
MINLP	Mixed integer Non-Linear Programming
QoS	Quality of Service
SDO	Service Deployment Optimizer
SLA	Service Level Agreement
SP	Service Provider
TREC	Trust, Risk, Eco and Cost
VM	Virtual Machine
VMU	Virtual Machine Unit
WAR	Web ARchive
WP	Work Package
WS	Web Service

2 Admission Control User Guide

The Admission Control (AC) component is a set of four service-oriented components (RESTful services) with distinct tasks, that work together to provide the overall admission control functionality in OPTIMIS. Their tasks cover workload analysis, Trust-Risk-Eco-Cost (TREC) information retrieval, optimization modelling and admission control, and web service (WS) access to the aforementioned functionality. The services that encapsulate the capabilities mentioned above are the *Workload Analyzer,* the *TREC Analyzer*, the *Admission Controller* and the *AC Gateway* respectively.

It should be noted that in the final version of the Admission Control system, the Workload Analyzer subcomponent has been replaced by *AC_PhHostssInfo_aaS* component, which is a lighter and better performing version of the *Workload Analyzer* component. However, in order not to create confusion and for consistency reasons, the name *Work Analyzer* is still used to refer to this subcomponent of the AC in the figures and sequence diagrams that follow.

Moreover, differently to the previous version of the AC system, the AC Gateway is now retrieving the Trust, Risk, Eco and Cost required information by communicating with the *AC_TRECcommon_aaS* component, which serves as a common interface to the TREC tools.

During admission control, the Admission Controller formulates and solves a mathematical optimization problem [1] for finding the optimum allocation of the services to be deployed inside the IP's resources by communicating either with the Heuristic Solver developed in Python or the General Algebraic Modeling System (GAMS) [4]. A high-level view of the AC architecture and the way the AC components (shown in blue color) work together to provide the overall AC functionality is shown in Figure 1.

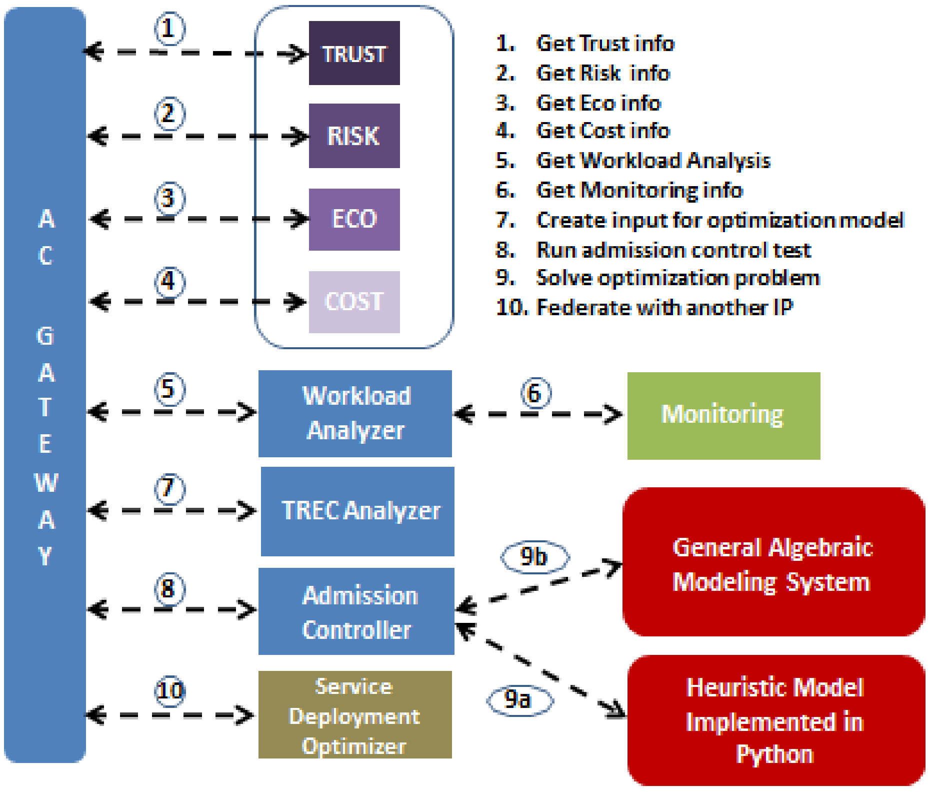
AC GATEWAY
1
2
3
4
5
6
7
8
9a
9b
10
TRUST
RISK
ECO
COST
Workload Analyzer
Monitoring
TREC Analyzer
Admission Controller
General Algebraic Modeling System
Heuristic Model Implemented in Python
Service Deployment Optimizer
1. Get Trust info
2. Get Risk info
3. Get Eco info
4. Get Cost info
5. Get Workload Analysis
6. Get Monitoring info
7. Create input for optimization model
8. Run admission control test
9. Solve optimization problem
10. Federate with another IP

Figure 1: High-level view of the Admission Control architecture.

2.1 Release information

Component Name	Release Number	Release Date
AC Gateway	1.0-SNAPHOT	05/2013
Admission Controller	1.0-SNAPHOT	05/2013
TREC Analyzer	1.0-SNAPHOT	05/2013
AC_PhHostsInfo_aaS	1.0-SNAPHOT	05/2013
AC_TRECcommon_aaS	1.0-SNAPHOT	05/2013
AC ONE	1.0-SNAPHOT	05/2013

Table 1: AC components release information.

2.2 Introduction

As already explained, the AC is a set of RESTful components:

AC Gateway: the AC Gateway component is the Admission Control's entry point. Apart from serving as a single point of access to the overall AC functionality, the AC Gateway also performs calls to external to the AC components, as well as orchestrating the internal interactions among the rest AC components.

AC_PhHostsInfo_aaS: the AC_PhHostsInfo_aaS component is responsible assisting the process of admission control by providing information about the physical hosts of the underlying Cloud infrastructure. To this direction the AC_PhHostsInfo_aaS establishes communication with the Monitoring component for obtaining critical information regarding the current available physical resources in the Cloud, either in use or idling. The output of this analysis serves as input to the Admission Controller component.

AC_TRECcommon_aaS: the AC_TRECcommon_aaS component is responsible assisting the process of admission control by providing information about the TREC factors of a service and of a physical host of the underlying Cloud infrastructure. To this direction the AC_PhHostsInfo_aaS establishes communication with the TRECCommonAPIIP component andt the TRUST, RISK, COST and ECO components for obtaining their information regarding their current values. The output of this analysis serves as input to the Admission Controller component.

AC ONE: the AC ONE is a component that was developed in order for the Admission Control to become functional in an OpenNebula environment, and it supports the retrieval the required monitoring information and the deployment of the VMs according to the generated allocation pattern in an OpenNebula environment. However, note this component is only required when the AC is operating in standalone mode, i.e. without the Cloud Optimizer (CO) component in place. Similarly to the usual operation of the AC in an OPTIMIS Emotive environment, when operating in an OpenNebula OPTIMIS enhanced environment, the monitoring information is retrieved from the CO (and not directly from the OpenNebula infrastructure) and the actual deployment of the VMs is handled by the DRP4ONE VMM plugin. Therefore it is required only when the CO component is not used.

TREC Analyzer: the TREC Analyzer is a supportive component to the Admission Controller, whose main responsibility is to transform the results of the Workload Analyzer and those of the TREC Assessment tools, as well as the requirements coming from the Service Manifest into a format understandable by the Admission Controller, which is the actual component that performs the admission control test.

Admission controller: The Admission Controller, the "brain" of the AC subsystem, is the AC subcomponent that is responsible for performing the admission control test. In case of acceptance, it provides the optimum allocation pattern (including the cost and related risk), namely the allocation offer, for each of the services in the manifest. This is the allocation offer and it can be calculated either by Heuristic Solver implanted in Python or by GAMS model using GAMS.

As already described, the AC interacts with the following OPTIMIS components: Trust, Risk, Eco, Cost, Monitoring system, and Service Deployment Optimizer. Therefore, in order for the AC to be fully functional, the aforementioned components should be also deployed and up and running. Furthermore, Python needs to be installed in the same physical machine that hosts the Admission Controller or alternatively the General Algebraic Modeling System in case someone wants to use it.

2.3 Use Cases

When it comes to the AC component, two use cases have been identified:

1. **Single mode operation:** This use case corresponds to the Private and Multi-Cloud architecture ("All Optimis" and "Some Optimis"). In this use case, the AC component is invoked by the Cloud QoS Agreement Factory (CQoS) that is in turn invoked by the SP)through passing a complete service manifest to be considered for admission in the underlying IP's Cloud that owns the AC component. In case the manifest cannot be admitted, the AC returns a negative answer.
2. **Federated mode operation:** this corresponds to the "Federated Cloud" and "Hybrid Cloud" architectures. Differently from the previous use case in which the result of inability to admit a new service resulted in denial of service, in this case the AC delegates the un-admitted part of the service to the Service Deployment Optimizer (SDO). The latter is responsible for finding an external IP (IP B) and returning the offer to the AC (of IP A) in order to merge them and return them to the SP level through the CQoS.

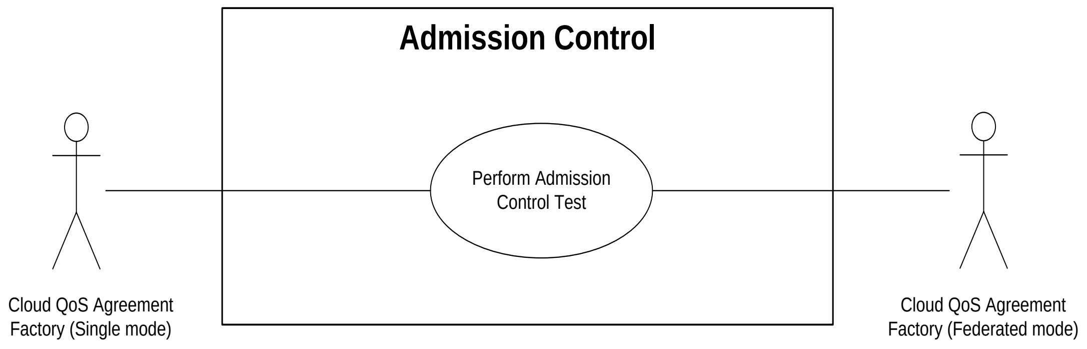
Admission Control
Perform Admission Control Test
Cloud QoS Agreement Factory (Single mode)
Cloud QoS Agreement Factory (Federated mode)

Figure 2: Admission Control use cases

2.4 Sequences

In accordance to the identified use cases, we can distinguish between two different operations of the Admission Control in OPTIMIS: (1) Single-mode operation, and (2) Federated-mode operation.

2.4.1 Single-mode operation

This operation corresponds to both "Private" and "Multi-Cloud" mode, for which Admission Control will only consider the placement of all VMs inside the underlying IP Cloud. The response may as well be negative, i.e. the VMs cannot be admitted inside the given infrastructure, if Admission Control fails to accommodate at least one of them. Under this mode and in case of rejection, the AC will not consider delegation of the un-admitted part of the service manifest to other IPs. Figure 3 shows the detailed sequence diagram for this mode.

In more detail the sequence involves the following steps:

- **Step 1:** CQoS triggers AC Gateway passing the service manifest of the service to be admitted, which contains the following information that is of importance to the admission control process:
    - Availability constraint
    - Performance requirements for all service components that comprise the service, both basic (minimum number of VMs) and elastic ones (maximum number of VMs)
- **Steps 2-5:** At next step, the AC Gateway contacts the Workload Analyzer sub-component passing it the service manifest in order to retrieve information about the current workload of the underlying infrastructure. The Workload Analyzer contacts the Monitoring component in order to retrieve monitoring data. The monitoring information is then passed onto the Workload Analyzer and the result of the workload analysis is passed back to the AC Gateway.
- **Steps 6-7:** Having obtained the workload analysis, the AC Gateway contacts the TREC tools in order to retrieve information about trust, risk, eco and cost, namely the TREC factors. The AC Gateway contacts the aforementioned components through four (4) different interfaces, one for each one of them (see [2] for more details).
- **Steps 8-9:** Afterwards, the AC Gateway triggers the TREC Analyzer passing the manifest, the values of the TREC factors and the result of the workload analysis. The TREC Analyzer transforms and stores this information into several files that serve as input to the optimization model behind the Admission Controller (see [1] for more details).
- **Steps 10-11**: The paths to the input of the optimization model along with the service manifest is then passed onto the Admission Controller, which runs the admission control test and solves the optimization problem by communicating with Heuristic Model implemented in Python or by communicating with the General Algebraic Modeling System (GAMS).

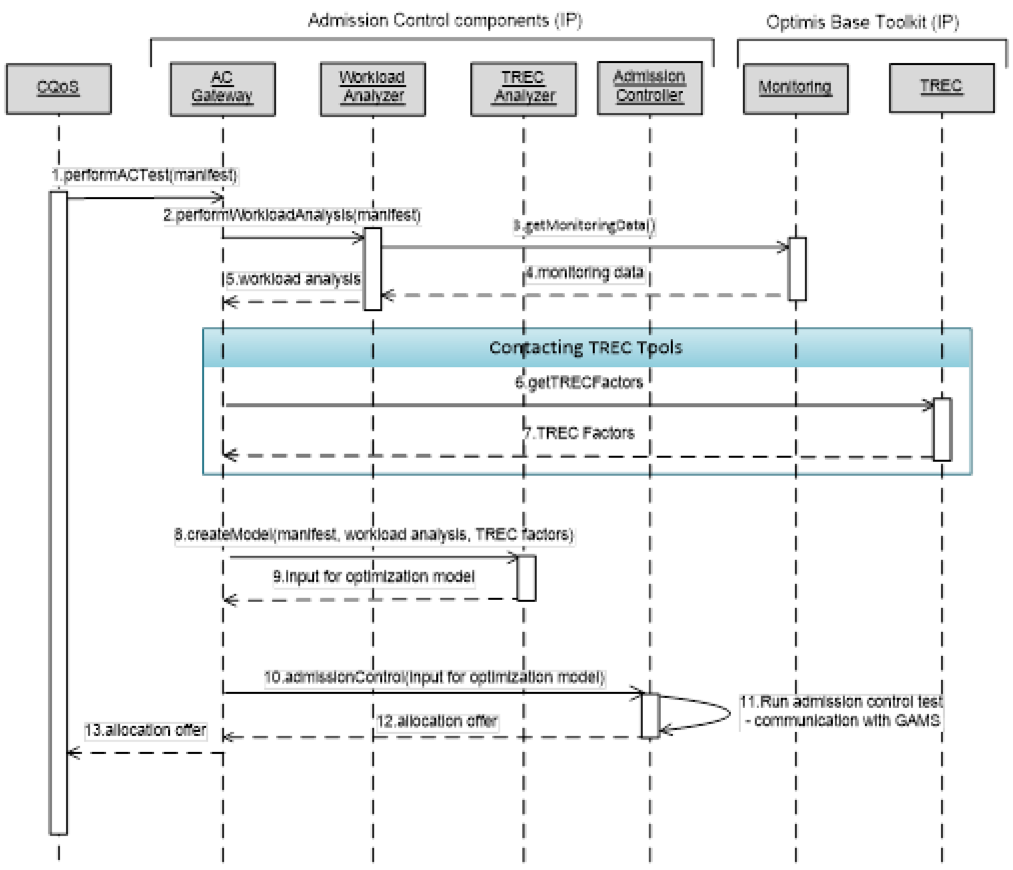
Admission Control components (IP)
Optimis Base Toolkit (IP)
CQoS
AC Gateway
Workload Analyzer
TREC Analyzer
Admission Controller
Monitoring
TREC
1.performACTest(manifest)
2.performWorkloadAnalysis(manifest)
3.getMonitoringData()
4.monitoring data
5.workload analysis
Contacting TREC Tools
6.getTRECFactors
7.TREC Factors
8.createModel(manifest, workload analysis, TREC factors)
9.input for optimization model
10.admissionControl(input for optimization model)
11.Run admission control test - communication with GAMS
12.allocation offer
13.allocation offer

Figure 3: Sequence diagram for single-mode operation

- **Step 12:** The output of the AC in case of acceptance is an allocation offer that includes the following information:
    - Acceptance or rejection of the service, and in case of acceptance
    - Allocation offer, including the hosts that are to be used for the placement of the admitted VMs of the service.
    - The cost of the allocation, as obtained from the Cost tool.
- **Steps 13:** The allocation offer is passed to the CQoS for further assessment.

2.4.2 Federation-mode operation

Under this mode, and in the case of partial admission of the service workflow under examination, the AC may delegate the un-admitted part of the manifest to other trusted IPs, by communicating with the SDO. Upon finding an IP for the un-admitted services, the AC is responsible for merging the two allocation offers into one and passing in it back to the Cloud QoS Agreement Factory. The way AC operates under deployment federations is shown in Figure 4 below.

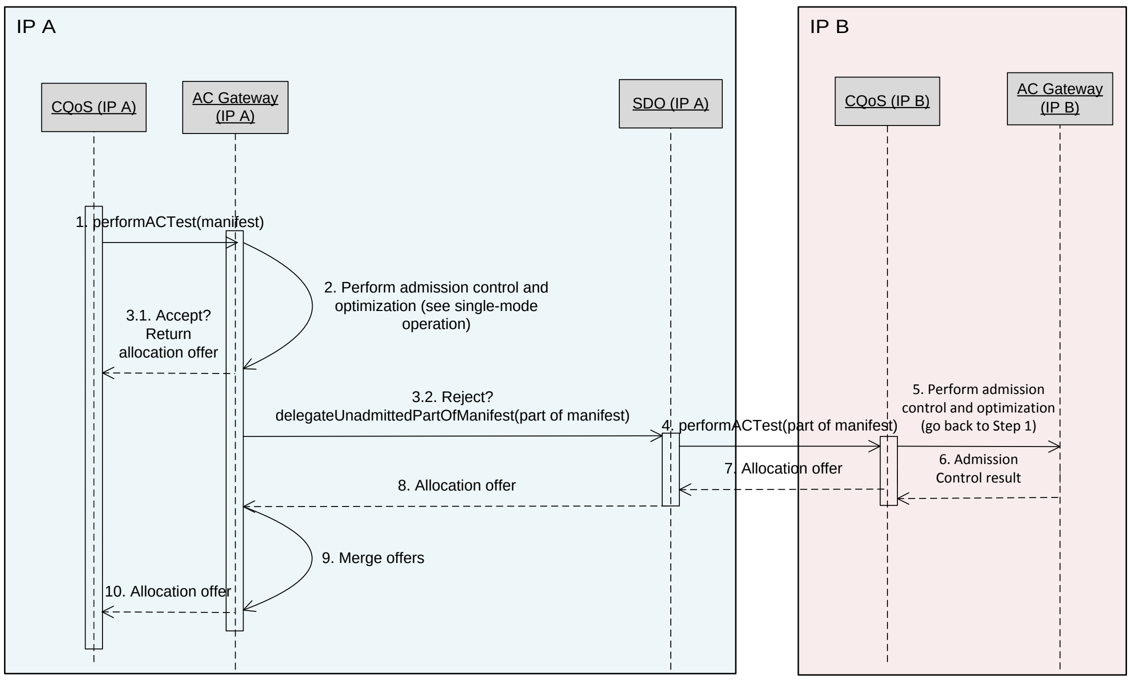
IP A
IP B
CQoS (IP A)
AC Gateway (IP A)
SDO (IP A)
CQoS (IP B)
AC Gateway (IP B)
1. performACTest(manifest)
2. Perform admission control and optimization (see single-mode operation)
3.1. Accept? Return allocation offer
3.2. Reject? delegateUnadmittedPartOfManifest(part of manifest)
4. performACTest(part of manifest)
5. Perform admission control and optimization (go back to Step 1)
6. Admission Control result
7. Allocation offer
8. Allocation offer
9. Merge offers
10. Allocation offer

Figure 4: Sequence diagram for federated-mode operation

In more detail the sequence involves the following steps:

- **Step 1:** CQoS triggers the AC Gateway passing the service manifest to be admitted.
- **Step 2:** The AC performs all steps that are involved in the single-mode admission control operation (see sequence diagram in Figure 3).
- **Step 3.1:** In case of full acceptance, the AC Gateway returns the allocation offer to the CQoS for assessment.
- **Step 3.2:** In case the service can only be partly admitted into the infrastructure, the AC Gateway delegates the un-admitted part of the manifest to the SDO component that resides at the IP A domain.
- **Step 4-5:** The SDO processes the information and launches a new admission control test for the un-admitted part of the manifest by contacting CQoS (IP B). The later one forwards the test to the AC at another IP domain (IP B).
- **Step 6-8:** The output of the AC (IP B) is passed back to the CQoS (IP B), which passes it back to the SDO (IP A) and the later back to the AC at IP A.
- **Step 9-10:** The AC Gateway (IP A) merges the two offers and returns the result to the CQoS at IP A.

2.5 Functionalities

2.5.1 Gatewaying and orchestration

The AC Gateway implements a set of RESTful interfaces that allow other components to make use of the overall AC functionality. The responsible for triggering the aforementioned set of calls is the Cloud QoS Agreement Factory (CQoS) by passing the service manifest in order for the AC to analyze it internally. The response send back to the CQoS component is an updated version of the service manifest that includes the decision of whether the service will be admitted or not, and in case of acceptance, it also includes the optimal allocation pattern of the service components inside the Cloud physical resources (see method *performACTest* of for more details).

It should also be noted that in case of partial acceptance of a service, and given that federation is allowed by the SP and that there are no restrictive affinity rules in place when it comes to the deployment of the service components and the service as a whole, the AC Gateway contacts the Service Deployment Optimizer (SDO) for establishing possible federation with another IP.

2.5.2 Retrieval of monitoring information about the available resources

The AC_PhHostsInfo_aaS main feature is the retrieval of monitoring data of the underlying Cloud infrastructure from the Monitoring component. Upon request from the AC Gateway, the AC_PhHostsInfo_aaS communicates with the Cloud Optimizer to get the available physical hosts names and then queries the Monitoring component for the load of the available infrastructure by using the interfaces provided by the Monitoring component. It first retrieves a list of all physical hosts, and the virtual hosts running on a particular physical host. It then retrieves monitoring data sets for all physical and virtual hosts. Finally, an xml document is created for all processed metrics, which is then returned to the AC Gateway and then retrieved by the TREC Analyzer component for generating the necessary by the Admission Controller input to run the admission control test (see method *getPhysicalHostsInfo* of Table 2 for more details).

2.5.3 Retrieval of TREC values used in the optimization process

The AC_TRECcommon_aaS main feature is the retrieval of TREC data in the underlying Cloud infrastructure from the TRECCommonAPIIP component. Upon request from the AC Gateway, the AC_TRECCcommon_aaS communicates with each one of the TREC (Trust, Risk, Eco and Cost) to get the available info for each service and also communicates with Eco and Risk to get available info for each physical host by using the interfaces provided by the TRECCommonAPIIP. Finally, all info returned to the AC Gateway and then retrieved by the TREC Analyzer component for generating the necessary by the Admission Controller input to run the admission control test (see method *getTRECofAserviceInfo and getTRECofAhostInfo* of Table 2 for more details).

2.5.4 AC ONE

The AC ONE supports the direct retrieval of the required monitoring information from an OpenNebula cloud environment as well as the actual deployment of the VMs according to the optimal allocation pattern, which is generated by the Admission Controller.

2.5.5 Generating input for the optimization model

AC Gateway triggers TREC Analyzer, once the former receives the results from both the Workload Analyzer and the TREC tools. These results are fed to the TREC Analyzer, whose main functionality is to create several input files in CVS format needed by GAMS optimization model behind the Admission Controller (see method *createModel* of Table 2 for more details).

2.5.6 Perform admission control test

In accordance to the implementation of the rest AC and OPTIMIS subcomponents, the Admission Controller’s functionality is offered as a web service through a RESTful interface. The latter exposes an operation for launching the admission control test, which is triggered by the AC Gateway according to the sequence in

Figure 3. In more detail, the AC Gateway contacts Admission Controller through a RESTful call, passing the path where the input files for the GAMS model are located. The Admission Controller then communicates with GAMS (which gets its input values from the CVS files created by the TREC Analyzer). The Admission Controller then parses the model output (Heuristic or GAMS) and generates a response which includes the optimum allocation scheme (in case of acceptance), which is passed back to the AC Gateway (see method *getAdmissionControl* of Table 2 for more details).

2.5.7 Setting IP's TREC-based acceptance policy

The AC through its Gateway exposes an interface for dynamically setting the weights of the TREC factors in the overall objective that is optimized. For each of the four TREC weights, a CVS file is created, which are later used inside the optimization model that runs in GAMS. This method is offered as part of the holistic management that drives decisions in OPTIMIS (see method *setPolicy* of Table 2 for more details).

2.5.8 Setting IP's TREC-based constraints policy

The AC through its Gateway exposes an interface for dynamically setting the constraints of the TREC factors in the overall objective that is optimized. For each of the four TREC constraints, a CVS file is created, which are later used inside the optimization model either Heuristic Model or Gams. This method is offered as part of the holistic management that drives decisions in OPTIMIS (see method *setPerServiceConstraints* of Table 2 for more details).

NOTE: For a step-by-step explanation and examples about how to create and use a client for this method please see Admission_Control_Installation_Guide.doc.

Method Name	Description	Input	Output	Owner	Actor
performACTest	Perform admission control test	List of Service manifests	List of (updated) Service manifests	AC Gateway	CQoS
getPhysicalHostsInfo	Receive physical hosts info	IPaddress - Port	XML document	AC_PhHostsInfo_ aaS, AC ONE	AC Gateway
createModel	Generate input for optimization model	Service manifest, Workload analysis document, TREC factors	Input to optimization model (CVS files)	TREC Analyzer	AC Gateway
getAdmissionControl	Perform optimum allocation	Input to optimization model	Allocation offer	Admission Controller	AC Gateway
setPolicy	Set TREC policy	TREC policy levels (predetermined classes)	Change internal AC TREC policy	AC Gateway	Cloud Optimizer
setPerServiceConstraints	Set constaints policy	Input to optimization model	Change internal AC constraints policy	AC Gateway	Cloud Optimizer
getTRECofAserviceInfo	Receive TREC info	IPaddress – Port- Service Manifest	String	AC_TRECCommon _aaS	AC Gateway
getTRECofAhostInfo	Receive TREC info	IPaddress - Port	String	AC_TRECCommon _aaS	AC Gateway
deployVMs	Deploy VMs in OpenNebula	Allocation offer	String	AC ONE	AC Gateway

Table 2: AC interfaces

2.6 Known limitations

These limitations are known to exist in this release of the Admission Control:

2.6.1 GAMS license required for running the GAMS implementation of the model

GAMS is not open-source. Without a license file GAMS can only operate in free demonstration mode with the following limitations:

1. Model limits:
   - Number of constraints and variables: 300.
   - Number of nonzero elements: 2000 (of which 1000 nonlinear).
   - Number of discrete variables: 50.
2. Additional limit for global solvers: Number of constraints and variables: 10.

The AC optimization model that is used to perform Admission Control exceeds the aforementioned limits, and therefore in order to solve it using the GAMS implementation a GAMS base license must be acquired. This base license includes a set of MINLP (Mix Integer Non Linear Programming) solvers that could be used for solving the problem. If there is no license in place, GAMS will abort its execution and report an error message. For achieving the best performance, an extra GAMS license for the Branch and Reduce Optimization Navigator (BARON)[5] may be optionally acquired.
However note that obtaining a GAMS license is not required since a heuristic solver, ad-hoc to the specific optimization was developed during Y3, which is open source. The heuristic solver (see Admission Control Scientific report for details [1]) can obtain solutions of very good quality compared to the ones that BARON solver reports and can do that at a fraction of the cost that is required by the BARON solver.

2.6.2 Performance limitations of the AC heuristic solver

The evaluation of the implemented heuristic solver for the AC optimization problem proved that the obtained solution quality is worse than the actual optimal one only in 0,6% of the examined cases, reaching a maximum of 7% only in one profile, but can converge to this sub-optimal solution much faster (two to three orders of magnitude faster). The results indicated that acceptable running times (< 30 secs, which is the default time out when it comes to web services) can be obtained for up to 500 hosts even for demanding cases whereby 5 services with 5 components each are examined simultaneously. When there is no requirement for simultaneous examination of multiple services, the number of hosts may reach a maximum of 5000. For larger sized problems, it is suggested that there is a different deployment of the AC every 500 or 5000 hosts, for examination of multiple services are examined in parallel or not .

2.6.3 Statistical knowledge required to enable AC probabilistic mode

Given that the required by the AC statistical knowledge is not available, the AC model can be configured to run under deterministic mode (i.e. always guarantee 100% availability of the resources) or run under the assumption that the probability distributions of the components capacity requirements are independent and uniformly distributed within the range defined by the basic and elastic requirements of the components.

2.7 Getting started

For a step-by-step guidance for installing, configuring and using each AC component please see Admission_Control_Installation_Guide.doc.

Source code and information about the structure of the source directories can be found in README.txt files for each AC component.

2.7.1 Using the Software

AC Gateway:

In order to use the AC Gateway, one must generate WS clients for invoking the operations described below for performing an admission control test on a given service manifest and for defining the policy with regard to the TREC factors.

Method Name	**Description**	**Input**	**Output**	**Owner**	**Actor**
performACTest	Perform admission control test	List of Service manifest	List of Service manifest	AC Gateway	CQoS
setPolicy	Set TREC policy	4 float	4 csv files	AC	Cloud

		values		Gateway	Optimizer
perServiceConstraints	Set TREC Constraints for each Service	4 float values	4 csv files	AC Gateway	Cloud Optimizer

Table 3: AC Gateway interfaces

The implementation of method "*performACTest*" is located in the Java class called "ACModelApi" under version "1.0-SNAPSHOT".

The unit tests "ACGatewayRemoteTest" and "ACGatewayTest" under the ACGateway Maven module provide example clients for invoking the method "*performACTest*".

The implementation of methods "*setPolicy*", "perServiceConstraints" can be found in the Java class "ACDataAPi" under version "1.0-SNAPSHOT".

The unit test "SetPolicyTest" under the ACGateway Maven module provides an example client for invoking the method *"setPolicy"*.

The unit test "PerServiceConstraints Test" under the ACGateway Maven module provides an example client for invoking the method *"*perServiceConstraints*"*.

Important notice:

Please note that in order for method *"performACTest"* to properly work, the method "*setPolicy*" must be invoked first.

AC_TRECcommon_aaS:

In order to use the AC_TRECcommon_aaS, one must generate a client that implements the following interface in order to use it in standalone mode.

Method Name	**Description**	**Input**	**Output**	**Owner**	**Actor**
AC_TRECcommon	Returns TREC factors	Service Manifest, Physical Hosts Names	TREC values	AC_TRECcommon_aaS	AC Gateway

Table 4: AC_TRECcommon_aaS interfaces

There is a Maven module called "AC_TRECcommon_aaS" with version "1.0-SNAPSHOT" that provides an easy implementation to use the *"*AC_TRECcommon*"* method.

AC_PhHostssInfo_aaS:

In order to use the AC_PhHostssInfo_aaS, one must generate a client that implements the following interface in order to use it in standalone mode.

Method Name	**Description**	**Input**	**Output**	**Owner**	**Actor**
getXML	Generate Physical Hosts Info	IP address	Physical Hosts Info	AC_PhHostssInfo_aaS	AC Gateway

Table 5: AC_PhHostssInfo_aaS interfaces

There is a Maven module called "AC_PhHostssInfo_aaS" with version "1.0-SNAPSHOT" that provides an easy implementation to use the *"getXML"* method.

AC_ONE:

In order to use the AC ONE, one must generate a client that implements the following interface in order to use it in standalone mode.

Method Name	Description	Input	Output	Owner	Actor
getXML	Retrieve resources Info from ONE	IP address (ONE XML-RPC endpoint)	Physical Hosts Info	AC ONE	AC Gateway
deployVMs	Deploy VM in ONE	Allocation offer	Success/Failure	AC ONE	AC Gateway

Table 6: AC_ONE interfaces

There is a Maven module called "AC_ONE" with version "1.0-SNAPSHOT" that provides an easy implementation to use the *"getXML"* method.

TREC Analyzer:

In order to use the TREC Analyzer, one must generate a client that implements the following interface in order to use it in standalone mode.

Method Name	Description	Input	Output	Owner	Actor
createModel	Generate input for optimization model	Physical Hosts Info, TREC factors, Service Manifest	Input to optimization model (CVS files)	TREC Analyzer	AC Gateway

Table 7: TREC Analyzer interfaces

There is a Maven module called "TRECAnalyzer" with version "1.0-SNAPSHOT" that provides an easy implementation to use the *"createModel"* method.

Admission Controller:

In order to use the Admission Controller in standalone mode, one should generate a WS client that implements the following interface:

Method Name	Description	Input	Output	Owner	Actor
getAdmissionControl	Perform admission control test	GamsDirectory, AllocationInfo Path, IP_Id, Solver Selection	Allocation offers, Allocation Details	Admission Controller	AC Gateway

Table 8: Admission Controller interfaces

The implementation of the "*getAdmissionControl*" method can be found in the Java class "AdmissionController" under version "1.0-SNAPSHOT".

2.7.2 Testing the Software

1. In order to test the method *"performACTest"* of the AC Gateway one needs to run the "ACGatewayRemoteTest". The latter has client for examining multiple service manifests at the same time and one for passing only one service manifest.

When running the aforementioned test, the paths to the service manifest(s) need to be given as input. Make sure that the correct paths and IPs are included in the file config.properties (located at /ACGateway/src/test/resources/).

The output of the “ACGatewayRemoteTest” is a list of updated service manifests the number of which is equal to the number of the service manifest that was given as input to the test. The exact location of each updated service manifest file is printed in the terminal window.

The command for invoking the “ACGatewayRemoteTest”:

```
mvn –Dtest=ACGatewayRemoteTest test
```

which is run at the location of the pom.xml.

The “ACGatewayTest” can be run in a similar way. The only difference is that has client for examining only one service Manifest at the same time.

All the test can be executing with the command:

```
mvn test
```

Important notice:

Please note that in order for method “*performACTest*” to properly work, the method “*setPolicy*” must be invoked first.

2. In order to test the “setPolicy”, you need to run the “SetPolicyTest”.

   The command for invoking the “SetPolicyTest” (under the pom.xml location) is the following:

   ```
   mvn –Dtest=SetPolicyTest test
   ```

3. In order to test the PerServiceConstaints, you need to run “PerServiceConstraintsTest”.

   The command for invoking the “PerServiceConstraintsTest” (under the pom.xml location) is the following:

   ```
   mvn –Dtest= PerServiceConstraintsTest test
   ```

4. In order to test the AC_PhHostssInfo_aaS, you need to run the command :

   mvn test (under the pom.xml location)

   Before running the test, make sure that the correct paths and IPs are included in the file config.properties (located at / AC_PhHostssInfo_aaS /src/test/resources/).

5. In order to test the AC_TRECcommon_aaS, you need to run the command :

   mvn test (under the pom.xml location)

   Before running the test, make sure that the correct paths and IPs are included in the file config.properties (located at / AC_TRECcommon_aaS /src/test/resources/).

2.7.3 Configuration

The Admission Control configuration can be done with two ways:

- Inside war configuration (Default Values)
- Outside war configuration (Optional)

  The Outside war configuration is optional and **overrides** the Inside war configuration. In case of absence of outside war configuration or in case of absence of certain property from outside war configuration file the Admission Control is auto-configured with the default values from inside war configuration.

Inside war configuration (Default Values):

- **Admission Controller:**

  1. In case of use GAMS Solver for solving the optimization problem the file AdmissionController/src/main/resources/gams.properties must contain the correct path to GAMS_DIR:

  executable = GAMS_DIR/gams

  The default path is:

  executable = /usr/gams2395/gams

  2. The Admission Controller IP address must be noted in order to be included in ACGatewayConfig.properties.

- **ACGateway:**

  The file src/main/resources/ACGatewayConfig.properties must be changed to include the following:

  - The paths to input files for the Admission Controller:

    gams.path, path.gams, path.AllocationInfo

    Three relative paths for getting

    GamsDirectory and AllocationInfoDirectory depending on where Apache Tomcat is installed (Apache_DIR).

    GamsDirectory = Apache_DIR/gams.path/path.gams

    AllocationInfoDirectory = Apache_DIR/gams.path/path.AllocationInfo

    Default values:

    path.gams = gams

    path.AllocationInfo = AllocationInfo

    gams.path = webapps/AdmissionController/WEB-INF/classes/

  - The IP and the port where the Admission Controller is listening. Note that the TREC Analyzer must be deployed in the same physical host as the Admission Controller.

    Default values:

    admission.controller.ip = localhost

    admission.controller.port = 8080

  - The IP and the port where the AC_TRECcommon_aaS is listening.

    Default values:

    AC_TREC.ip = localhost

    AC_TREC.port = 8080

  - The IP and the port where the Risk is listening.

    Default values:

risk.ip = localhost

risk.port = 8080

- The IP and the port where the Trust is listening.

  Default values:

  trust.ip = localhost

  trust.port = 8080

- The IP and the port where the Eco is listening.

  Default values:

  eco.ip = localhost

  eco.port = 8080

- The IP and the port where the Cost is listening.

  Default values:

  cost.ip = localhost

  cost.port = 8080

- The IP and the port where the AC_PhHostssInfo_aaS is listening.

  Default values:

  physicalHostsInfo_aaS.ip = localhost

  physicalHostsInfo_aaS.port = 8080

- The solver which will solve the optimization problem. You can choose between Heuristic Solver, and GAMS Solver

  Default Values:

  admission.controller.solver = use_Heuristic

<u>Outside war configuration (Optional):</u>

– **<u>Admission Controller:</u>**

  In the file:

  /opt/optimis/etc/AdmissionControl/gams.properties

  The property executable must set to contain the correct path to GAMS_DIR:

  i.e

  executable = /usr/gams2395/gams

– **<u>ACGateway:</u>**

  - In the file:

    /opt/optimis/etc/AdmissionControl/trec.properties

    could be set the IP and the port of each one of the TREC components:

    eco.ip, eco.port, trust.ip, trust.port, risk.ip, risk.port, cost.ip, cost.port

    i.e

eco.ip = localhost

eco.port = 8080

- In the file:

  /opt/optimis/etc/AdmissionControl/AdmissionControl.properties

  The properties gams.path and AllocationInfo.path are absolute paths.

  The property admission.controller.solver set the Solver to solve the optimization problem. It can also set the IP and the port where the Admission Controller is listening and the IP and the port where the AC_PhHostssInfo_aaS is listening. It can also set the IP_ID.

  i.e

  gams.path = /opt/optimis/GamsDirectory/

  AllocationInfo.path = /opt/optimis/AllocationInfoDirectory/

  admission.controller.ip = localhost

  admission.controller.port = 8080

  physicalHostsInfo_aaS.ip = localhost

  physicalHostsInfo_aaS.port = 8080

  admission.controller.solver = use_GAMS

  IP_ID = ATOS

2.8 FAQ

N/A

2.9 Other information

2.9.1 Source Code Information

The Admission Control source code is divided in five directories:

- ACGateway
- AC_PhHostsInfo_aaS
- AC_TRECCommon_aaS
- AC ONE
- TREC Analyzer
- Admission Controller

2.9.2 Directory Structure

The Admission Control uses two directories:

- *Configuration directory (Optional)*: it contains the configuration files of the Admission Control. It is located at opt/optimis/etc/AdmissionControl.
- *Output logs directory*: it contains the log files generated by the Admission Control. It is placed either at opt/optimis/var/log/AdmissionControl:
  - ACModel.log: contains logs produced by the ACGateway.

- AC_PhHostsInfo_aaS.log: contains logs produced by the AC_PhHostsInfo_aaS.
- AC_TRECCommon_aaS.log:contains logs produced by the AC_TRECCommon_aaS.
- TREC Analyzer.log: contains logs produced by the TREC Analyzer.
- Admission Controller.log: contains logs produced by the Admission Controller.
- Admission.lgo: contains logs by produced by the Heuristic Solver implemented in Python.

2.9.3 Contributors

The people that contributed to the development of the AC components are listed below:

AC Gateway: Sotiris Stamokostas (ICCS/NTUA), Juan Luis Prierto Martinez (ATOS)

AC_TRECCommon_aaS: Sotiris Stamokostas (ICCS/NTUA)

AC_PhHostsInfo_aaS: Sotiris Stamokostas (ICCS/NTUA)

AC ONE: Sotiris Stamokostas (ICCS/NTUA)

Workload Analyzer: Hassan Rasheed (SCAI Fraunhofer)

TREC Analyzer: Sotiris Stamokostas (ICCS/NTUA), George Katsis (ICCS/NTUA)

Admission Controller: Sotiris Stamokostas (ICCS/NTUA), Konstantinos Psychas (ICCS/NTUA), George Katsis (ICCS/NTUA), Kleopatra Konstanteli (ICCS/NTUA)

3 References

[1] Admission Control Scientific Report, ICCS/NTUA and other partners, May 2013.

[2] Admission Control Installation Guide, ICCS/NTUA and other partners, May 2013.

[3] AdmissionControl_README.txt, ICCS/NTUA and other partners, May 2013.

[4] General Algebraic Modeling System, GAMS, http://www.gams.com/.

[5] Nikolaos V. Sahinidis. BARON Branch and Reduce Optimization Navigator User's Manual v4.0. University of Illinois at Urbana-Champaing, Department of Chemical Engineering, June 2000. Available at http://archimedes.cheme.cmu.edu/baron/manuse.pdf.

[6] Konstanteli, K.; Cucinotta, T.; Psychas, K.; Varvarigou, T., "Admission Control for Elastic Cloud Services," *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, vol., no., pp.41,48, 24-29 June 2012.

[7] Konstanteli, K.; Varvarigou, T.; Cucinotta, T., "Probabilistic admission control for elastic cloud computing," *Service-Oriented Computing and Applications (SOCA), 2011 IEEE International Conference on* , vol., no., pp.1,4, 12-14 Dec. 2011.