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**

SLAManagement User Guide

Activity 2: *Service Construction*

WP 2.2: *Cloud QoS contracting and service configuration*

Due Date:		M34
Submission Date:		31/03/2013
Start Date of Project:		01/06/2010
Duration of Project:		36 months
Organisation Responsible for the Deliverable:		SCAI
Version:		1.0
Status		Final
Author(s):	Hassan Rasheed Wolfgang Ziegler	Fraunhofer SCAI
Reviewer(s)	Sotiris Stamokostas (NTUA) Johan Tordsson (UMU)	

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	10.04.2012	Initial version	Angela Rumpl
0.2	08.10.2012	Updated Version	Hassan Rasheed
0.3	31.03.2013	Updated version	Hassan Rasheed

Table of Contents

1 Introduction

This document includes the user guide for the software component SLAManagement. It is intended for software developers who are going to implement a client for the SLAManagement component.

1.1 Glossary of Acronyms

Acronym	Definition
D	Deliverable
WP	Work Package
QoS	Quality of Service
AC	Admission Control
CO	Cloud Optimizer
WSAG4J	WS-Agreement for Java

2 SLAManagement User Guide

2.1 Release information

Component Name	Release Number	Release Date
SLAManagement	0.0.2-SNAPSHOT	15.03.2013

2.2 Introduction

The SLAManagement component can be used to create Service Level Agreements between an OPTIMIS SP and an OPTIMIS IP. An agreement includes the Service Manifest as its main part of the contract. The Service Manifest describes the requirements that the Service Provider requires for the deployment of the service. It also describes the cost, trust, eco efficiency and risk factors that the IP will comply to. This guide will not go into the details of describing the service manifest. This information can be found in the OPTIMIS detailed design deliverable [4].

The SLAManagement component was implemented by using the WSAG4J framework [1]. The WS-Agreement for Java framework is a tool to create and manage service level agreements (SLAs) in distributed systems. It is an implementation of the OGF WS-Agreement standard [2]. WSAG4J helps in designing and implementing SLAs for a service and automates typical SLA management tasks like SLA offer validation, service level monitoring, persistence, accounting, and more.

2.3 Functionalities

- Retrieving Templates
- Negotiation
- Create Agreement
- Terminate Agreement
- Monitoring
- Notifications

2.3.1 Retrieving Templates

The OPTIMIS infrastructure providers will expose default templates to describe infrastructure services they are providing. These templates can then be used by the Service Provider (in this context the user) to choose the most fitting template and simply adjust some values, e.g. increase the minimum memory required as needed by its application.

To retrieve the agreement template provided by an IP the SP simply needs to create an agreement factory client and call the getTemplate() method. Furthermore, the SP needs to know the Template Name and ID of the template that he wants to retrieve.

```
http://pandora.atosorigin.es/svn/optimis/branches/OptimisY3/SLAManagement/trunk/sla-management-service/src/test/java/eu/optimis/sla/VMProvisioningIT.java
```

2.3.2 Negotiation

SP selects the template for OPTIMIS service deployment and creates a negotiation offer based on the agreement template. The negotiable parameters are service components and

associated virtual machines for number of instances, speed, memory, disk and network etc., the resource and TREC requirements and price constraints. The negotiation offer is sent to the IP SLA management, which in turn queries Admission Control whether or not to admit the requested service. Based on the AC decision a counter offer is created. The counter offer is returned to the initiator. This process is repeated until the maximum negotiation round are exceeded (failure) or an acceptable offer was negotiated (success). Complete example on instantiating negotiation instance and steering the negotiation process can be found in below test case:

```
http://pandora.atosorigin.es/svn/optimis/branches/OptimisY3/SLAManagement/trunk/sla-management-service/src/test/java/eu/optimis/sla/ServiceNegotiationIT.java
```

2.3.3 Create Agreement

As soon as the SP has decided the template fits his requirements, it can create an agreement which will perform the necessary actions on the IP side:

1. run admission control test to check if the service can be deployed on the IP side
2. As soon as admission control permits the deployment of the service, the manifest will be forwarded to the Cloud Optimizer, which will perform the deployment of the service, including startup of VMs and performing other required actions as needed.

To create an agreement you have to create an offer by using the retrieved template. Before creating an offer, it is possible to change values in the Service Manifest that is included in the agreement template.

```
XmlObject[] objects = serviceDescriptionTerm
                .selectChildren(XmlBeanServiceManifestDocument
                .type.getDocumentElementName());
XmlBeanManifestType manifestDoc = (XmlBeanManifestType) objects[0];
```

Now you can manipulate the values in the manifest xml document.

As soon as all values of the offer are set to fit the SPs requirements, an agreement can be created based on this offer:

```
AgreementOffer offer = new AgreementOfferType( template );
AgreementClient agreement = factory.createAgreement( offer );
```

2.3.4 Terminate Agreement

The termination of an agreement will undeploy the service from the OPTIMIS IP.

```
agreement.terminate();
```

2.3.5 Agreement Monitoring

To be sure the service is up and running, the SP can monitor the agreement service term states.

Complete example can be found in the following integration test:

```
http://pandora.atosorigin.es/svn/optimis/branches/OptimisY2/SLAManagement/tags/sla-management-0.0.1/trunk/sla-management-service/src/test/java/eu/optimis/sla/VMProvisioningIT.java
```

```
ServiceTermStateType[] stStates = agreement.getServiceTermStates();
int maxTrys = 6;
while ( stStates[ 0 ].getState() == ServiceTermStateDefinition.NOT_READY )
        {
            System.out.println( "Waiting to change state to ready..." );
            Thread.sleep( 5000 );
            stStates = agreement.getServiceTermStates();
```

```
            maxTrys--;
            if ( maxTrys == 0 )
            {
                fail( "State change of agreement failed. Monitoring not invoked." );
            }
        }
```

2.3.6 Notifications

To retrieve notifications about monitoring events, the SP first has to create a notification URI, which is used to subscribe to the subscription service of the SLAManagement component. The parameter NOTIFICATION_URL is simply the URL of the SLAManagement component + "/sla/notifications". The method createNotificationEndpoint() creates a new notification endpoint that stores monitoring events that are published by the SLA management layer. The notification endpoint can be queried by a component for the complete history of published events.

```
NotificationEndpointFactoryService notificationService =
   ClientFactory.create( NOTIFICATION_URL, NotificationEndpointFactoryService.class );

URI notificationUri = notificationService.createNotificationEndpoint();
```

Now this URI can be used to subscribe to the subscription service of the SLAManagement component. The subscribe() method creates a new subscription for an agreement. The provided notification endpoint acts as receiver of SLA monitoring events that are created by the SLA management layer. If the subscription is created successfully the URI of the created subscription is returned. The subscription resource URI is used to query subscription details and to terminate a subscription.

```
SubscriptionService subscriptionService =
          ClientFactory.create( SUBSCRIPTION_URL, SubscriptionService.class );
URI subscriptionUri = subscriptionService.subscribe( agreement.getAgreementId(), notificationUri
);
```

All notification events for this agreement will be published and the notification client can retrieve the events.

```
NotificationEndpoint notificationClient =
   ClientFactory.create( notificationUri, NotificationEndpoint.class );
notificationClient.getNotificationEventHistory()
```

Detailed information on the notification interface can be found in the OPTIMIS detailed design document [4].

Complete example can be found in the following integration test:

```
http://pandora.atosorigin.es/svn/optimis/branches/OptimisY2/SLAManagement/tags/sla-management-
0.0.1/trunk/sla-management-service/src/test/java/eu/optimis/sla/SLANotificationsIT.java
```

2.4 Getting Started

The SLAManagement component can be accessed via the standard WSAG4J interfaces. Therefore it is necessary to implement the WSAG4J client.

How to set up a client project in your Eclipse IDE can be found in the WSAG4J online documentation [3].

2.4.1 Testing the Software

To test your client implementation, install the SLAManagement web application on a tomcat server. Please refer to the Installation Guide for a description.

2.4.2 Configuration

Since WSAG4J uses WS-Security to digitally sign messages exchanged between client and server, a client must provide a set of security credentials. The easiest way to provide these credentials is to provide a valid LoginContext as shown below:

```
KeystoreProperties properties = new KeystoreProperties();
properties.setKeyStoreAlias("wsag4j-user");
properties.setPrivateKeyPassword("user@wsag4j");

properties.setKeyStoreType("JKS");
properties.setKeystoreFilename("/wsag4j-client-keystore.jks");
properties.setKeystorePassword("user@wsag4j");

properties.setTruststoreType("JKS");
properties.setTruststoreFilename("/wsag4j-client-keystore.jks");
properties.setTruststorePassword("user@wsag4j");

LoginContext loginContext = new KeystoreLoginContext(properties);
loginContext.login();
```

2.5 Other information

2.5.1 Source Code Information

The SLAManagement web application code can be checked out from the OPTIMIS subversion repository:

```
svn co http://pandora.atosorigin.es/svn/optimis/branches/OptimisY3/SLAManagement/trunk
```

2.5.2 Directory Structure

This directory contains the following subdirectories:

sla-management-server: the implementation of all server actions to create agreements etc.

sla-management-service: the configuration files and integration tests. This module is used to generate the war file by overlaying the present configuration files with the WSAG4J server war. The pom file is configured to start a tomcat server, deploy the generated war and run all integration tests.

sla-management-types: the xml schema files for service monitoring types and subscription types.

3 References

[1] WSAG4J framework: http://packcs-e0.scai.fraunhofer.de/wsag4j/index.html

[2] WS-Agreement specification: OGF, http://www.ogf.org/documents/GFD.192.pdf

[3] WSAG4J client configuration, http://packcs-e0.scai.fraunhofer.de/WSAG4J/client/client-setup.html

[4] OPTIMIS Detailed Design, Deliverable D1.2.2.2 of OPTIMIS project.