# D4.3.3

# Report on the Implementation of the Infrastructure Services Layer

Version 1.0

WP 4.3 Grid Infrastructure Services Layer

Dissemination Level: Public

Lead Editor: Antonis Litke, ICCS/NTUA

15/02/2007

Status: Final

**SIXTH FRAMEWORK PROGRAMME**

**PRIORITY IST-2002-2.3.1.18**

Information Society

*Grid for complex problem solving*

*Proposal/Contract no.: 004293*

**Context**

| | |
|---|---|
| **Activity 4** | Detailed Architecture, Design & Implementation |
| **WP 4.3** | Grid Infrastructure Services Layer |
| **Dependencies** | Based on work from WP 3.1, WP 4.1, WP 4.2 and WP 4.4. |

**Contributors:** Annalisa Terracina (DATAMAT), Antonis Litke (ICCS/NTUA), Daniele Bocci (CRMPA), Francesco D'Andria (ATOS Origin), Giulio Negro (CRMPA), Giuseppe Laria (CRMPA), Hannes Eichner (BOC), Ivanov Momtchil (BOC), Josep Martrat (ATOS Origin), Kleopatra Konstanteli (ICCS/NTUA), Nadia Romano (CRMPA), Robert Woitsch (BOC), Sergio Romero (TID), Jesús Movilla (TID), Tom Kirkham (CCLRC), Vassiliki Andronikou (ICCS/NTUA)

**Reviewers:** Cristian Morariu (University of Zurich), Juan E. Burgos (TID)

**Approved by:** Victor Villagrá, Universidad Politécnica de Madrid, Spain, as Quality Assurance Manager

| Version | Date | Authors | Sections Affected |
|---|---|---|---|
| 0.1 | 14/11/2006 | ICCS/NTUA | Table of Contents |
| 0.2 | 19/12/2006 | All | Initial contribution compiled into the document. Sections affected: EMS, Data Management Services, SLA-Enforcement Service, Monitoring Service, Metering Service, Policy Manager Service, SSDS , Annex B |
| 0.3 | 15/01/2007 | DATAMAT, TID, ICCS/NTUA, BOC | Updated contribution compiled into the document. Sections affected: EMS, DMS, Monitoring Service, Metering Service, SSDS , Annex A/B |
| 0.4 | 26/01/2007 | DATAMAT, CRMPA, CCLRC | Updated contribution compiled into the document. Sections affected: Data Management Service, Policy Manager, Annex A |
| 0.5 | 5/02/2007 | ICCS/NTUA | Pre-final version sent to internal reviewers |
| 0.6 | 13/02/2007 | All | Updated contributions addressing reviewers' comments compiled into the document. |
| 1.0 | 15/02/2007 | ICCS/NTUA | Final version |

# Table of Contents

1. Introduction ........ 14
  1.1. Scope of the Akogrimo Grid Infrastructure Services Layer ........ 14
  1.2. Services Overview ........ 15
2. Implemented Akogrimo Grid Infrastructure Services ........ 17
  2.1. Execution Management Service (EMS) ........ 17
    2.1.1. EMS Design ........ 17
      2.1.1.1. Functionality ........ 21
      2.1.1.2. Use Cases ........ 29
      2.1.1.3. Interactions with other services ........ 34
    2.1.2. EMS Implementation ........ 35
      2.1.2.1. Involved Technologies ........ 35
      2.1.2.2. Configuration and Deployment ........ 36
    2.1.3. Security Considerations ........ 36
  2.2. Data Management Service (DMS) ........ 37
    2.2.1. DMS Design ........ 37
      2.2.1.1. Functionality ........ 37
      2.2.1.2. Use Cases ........ 40
      2.2.1.3. Interactions with other services ........ 43
    2.2.2. DMS Implementation ........ 44
      2.2.2.1. Involved Technologies ........ 44
      2.2.2.2. Configuration and Deployment ........ 44
    2.2.3. Security Considerations ........ 45
  2.3. Monitoring Service ........ 45
    2.3.1. Monitoring Service Design ........ 45
      2.3.1.1. Functionality ........ 46
      2.3.1.2. Use Cases ........ 48
      2.3.1.3. Interactions with other services ........ 52
    2.3.2. Monitoring Service Implementation ........ 53
      2.3.2.1. Involved Technologies ........ 54
      2.3.2.2. Configuration and Deployment ........ 54
    2.3.3. Security Considerations ........ 55
  2.4. Service Level Agreement Enforcement Services (SLA-Enforcement Services) ........ 55
    2.4.1. SLA-Enforcement Services Design ........ 55
      2.4.1.1. Functionality ........ 55
      2.4.1.2. Use Cases ........ 57

2.4.1.3. Interactions with other services ..... 60
2.4.2. SLA-Enforcement Services Implementation ..... 61
2.4.2.1. Involved Technologies ..... 61
2.4.2.2. Configuration and Deployment ..... 61
2.4.3. Security Considerations ..... 62
2.5. Metering Service ..... 62
2.5.1. Metering Service Design ..... 62
2.5.1.1. Functionality ..... 62
2.5.1.2. Use Cases ..... 65
2.5.1.3. Interactions with other services ..... 69
2.5.2. Metering Service Implementation ..... 69
2.5.2.1. Involved Technologies ..... 70
2.5.2.2. Configuration and Deployment ..... 70
2.5.3. Security Considerations ..... 70
2.6. Policy Manager Service ..... 70
2.6.1. Policy Manager Service Design ..... 70
2.6.1.1. Functionality ..... 71
2.6.1.2. Use Cases ..... 71
2.6.1.3. Interactions with other services ..... 73
2.6.2. Policy Manager Service Implementation ..... 73
2.6.2.1. Involved Technologies ..... 75
2.6.2.2. Configuration and Deployment ..... 75
2.6.3. Security Considerations ..... 75
2.7. Semantic Service Discovery Service (SSDS) ..... 76
2.7.1. SSDS Design ..... 76
2.7.1.1. Functionality ..... 77
2.7.1.2. Use Cases ..... 81
2.7.1.3. Interactions with other services ..... 89
2.7.2. SSDS Implementation ..... 89
2.7.2.1. Involved Technologies ..... 90
2.7.2.2. Configuration and Deployment ..... 90
2.7.3. Security Considerations ..... 90
3. Interoperability issues ..... 91
GT4 vs WSRF.NET ..... 91
4. Conclusions ..... 92
Annex A. Design Details ..... 95

EMS – SIP Broker interaction ... 95
Annex B. API of Services ... 97
B.1. EMS API ... 97
B.2. DMS API ... 98
B.3. Monitoring Service API ... 99
B.4. SLA Enforcement Service API ... 99
B.5. Metering Service API ... 100
B.6. Policy Manager Service API ... 100
B.7. SSDS API ... 101

# List of Figures

Figure 1: Layered diagram of Akogrimo ........ 14
Figure 2: WP4.3 services interactions overview ........ 16
Figure 3: EMS services and their internal interactions ........ 20
Figure 4: Example of low level QoS parameters ........ 21
Figure 5: Sequence diagram for the Advertisement phase ........ 22
Figure 6:  Sequence diagram for the Negotiation and Discovery phase ........ 23
Figure 7:  Sequence diagram for the Advance Reservation phase ........ 25
Figure 8:  Sequence diagram for the Execution phase ........ 28
Figure 9: EMS use case ........ 29
Figure 10: Storing Data sequence diagram ........ 38
Figure 11: Retrieving Data sequence diagram ........ 38
Figure 12: Transferring Data sequence diagram ........ 39
Figure 13: Querying Stored Data sequence diagram ........ 39
Figure 14: Access Stored Data sequence diagram ........ 40
Figure 15: DMS use cases ........ 40
Figure 16: Resource Creation sequence diagram ........ 46
Figure 17: Start Monitoring sequence diagram ........ 47
Figure 18: Stop Monitoring sequence diagram ........ 47
Figure 19: Notification Handling sequence diagram ........ 48
Figure 20:  SLA interfaces ........ 56
Figure 21:  Sequence diagram for the Activation phase ........ 56
Figure 22: Sequence diagram for the Service Use phase ........ 57
Figure 23: Metering component and A4C in Akogrimo ........ 63
Figure 24: Sequence diagram of the Metering Service execution ........ 64
Figure 25: Metering Service use cases ........ 65
Figure 26: Require Policy sequence diagram ........ 71
Figure 27:  Policy Context ........ 74
Figure 28: Policy Attachment ........ 74
Figure 29:  Semantic Service Discover Service: Interfaces and Components ........ 77
Figure 30: Sequence diagram Service Discovery ........ 80
Figure 31: SSDS use cases ........ 82
Figure 32: EMS-SIP Broker interactions ........ 96

# List of Tables

Table 1: Services involved in the Negotiation phase of EMS .......... 23
Table 2: Services involved in the Advance Reservation phase .......... 25
Table 3: Services involved in the Execution phase .......... 29
Table 4: Advertise service use case .......... 30
Table 5: Negotiate SLA contract use case .......... 31
Table 6: Reserve resources use case .......... 32
Table 7: Execute business service use case .......... 33
Table 8: EMS interactions with other Akogrimo services .......... 34
Table 9: Upload Data use case .......... 40
Table 10: Retrieve Data use case .......... 41
Table 11:  Transfer Data use case .......... 42
Table 12: DMS interfaces .......... 43
Table 13: Monitoring resource creation use case .......... 48
Table 14: EMS enabling of the monitoring process on the previously created resources Use Case .. 49
Table 15: EMS disabling of the monitoring process on a previously created resource Use Case ...... 50
Table 16: Notification sent from Metering to Monitoring Consumer resource Use Case .......... 51
Table 17: Monitoring interactions with other Akogrimo services .......... 52
Table 18: SLA creation and execution Use Case .......... 57
Table 19: Violation Detection and Best Policy Application Use Case .......... 59
Table 20: SLA-Enforcement interactions with other Akogrimo services .......... 60
Table 21: Services involved in the execution of the Metering service .......... 65
Table 22: Manage of lifecycle Use Case .......... 66
Table 23: Collect Metering Info Use Case .......... 67
Table 24: Collect Accounting Info Use Case .......... 68
Table 25: Metering Service interactions with other Akogrimo services .......... 69
Table 26: Require Policy Use Case .......... 71
Table 27: Policy Manager interactions with other services .......... 73
Table 28: Meta-Model for Service Discovery .......... 78
Table 29: Modelling Service Requirements Use Case .......... 82
Table 30: Modelling Service Semantic Use Case .......... 83
Table 31: Modelling Service Provision Use Case .......... 84
Table 32: Register Service Use Case .......... 85
Table 33: Upload Discovery plug-in Use Case .......... 85
Table 34: Browsing for Services Use Case .......... 86
Table 35: Searching Services Use Case .......... 87

Table 36: Discovering Services Use Case ........ 88
Table 37: SSDS interactions with other Akogrimo services ........ 89
Table 38: GT4 and WSRF.NET services ........ 91
Table 39: EMS Interface ........ 97
Table 40: DMS Interface ........ 98
Table 41: Monitoring Interface ........ 99
Table 42: SLA Enforcement Interface ........ 99
Table 43: Metering Interface ........ 100
Table 44: Policy Manager Interface ........ 100
Table 45: SSDS interface ........ 101

# Abbreviations

| | |
|---|---|
| **Akogrimo** | Access To Knowledge through the Grid in a Mobile World |
| **A4C** | Authentication, Authorization, Accounting, Auditing and Charging |
| **BPEL** | Business Process Execution Language |
| **CPU** | Central Processing Unit |
| **DAO** | Data Access Object |
| **DMS** | Data Manager Service |
| **EMS** | Execution Management Services |
| **EPR** | Endpoint Reference |
| **GRAM** | Grid Resource Allocation Manager |
| **GT4** | Globus Toolkit 4 |
| **GUI** | Graphical User Interface |
| **MDS** | Monitoring and Discovery System |
| **NR** | Negotiation Resource |
| **OASIS** | Organization for the Advancement of Structured Information Standards |
| **OGSA** | Open Grid Services Architecture |
| **OGSA-DAI** | Open Grid Services Architecture Data Access and Integration |
| **OpVOBroker** | Operative VO Broker |
| **OWL** | Web Ontology Language |
| **OWL-S** | Web Ontology Language-Services |
| **QoS** | Quality of Service |
| **RFT** | Reliable File Transfer |
| **RR** | Reservation Resource |
| **SDC** | Semantic Service Discovery Component |
| **SIP** | Session Initiation Protocol |
| **SLA** | Service Level Agreement |
| **SMC** | Semantic Service Modelling Component |

| | |
|---|---|
| **SOAP** | Simple Object Access Protocol |
| **SP** | Service Provider |
| **SQL** | Structured Query Language |
| **SSDS** | Semantic Service Discovery Service |
| **SSL** | Secure Sockets Layer |
| **SWRL** | Semantic Web Rule Language |
| **TLS** | Transport Level Security |
| **URI** | Uniform Resource Identifier |
| **URL** | Uniform Resource Locator |
| **VO** | Virtual Organization |
| **WS** | Web Services |
| **WSDL** | Web Service Definition Language |
| **WSI** | Web Services Inter-operability |
| **WSRF** | Web Services Resource Framework |
| **XML** | Extensible Mark-up Language |

# Executive Summary

This document describes the design and implementation of the services that belong to the WP4.3 Grid Infrastructure Services Layer of the Akogrimo project, the architecture of which was presented in D4.3.1 (Architecture of the Infrastructure Services Layer) [1]. The Grid Infrastructure Services Layer defines a service-oriented architecture which consists of a set of services capable of providing the core Grid functionality. These capabilities include issues such as service discovery, advance reservation, execution management, monitoring, SLA enforcement, policy management and others.

The services that encapsulate the capabilities mentioned above are the Execution Management Service, the Data Management service, the Monitoring service, the Service Level Agreement Enforcement services, the Policy Manager, the Metering service and the Semantic Service Discovery Service. For each of these services, this document focuses on the following issues:

- Design: The functionality of the services is being described through the use of sequence diagrams and use cases.
- Implementation: It focuses on the technologies that were used for the development of each service and describes the configuration needed by them.
- Security: Considerations regarding the security are included for each service. The convergence of these considerations will lead to the definition of a security framework that will be vertically applied to all services that reside in the Grid Infrastructure Services Layer.

The Annex presents details on the design of the services as well as their corresponding APIs, in order to clarify their functionality and interactions with other modules.

# 1. Introduction

This deliverable describes the functionality of the services that were developed within the Akogrimo WP 4.3 and it is a continuation of the work presented in D4.3.1 (Architecture of the Infrastructure Services Layer) [1] and D4.3.2 (Prototype Implementation of the Infrastructure Services Layer) [2], that were based on D3.1.1 (Overall Architecture Version 1) [3] and D3.1.3 (Overall Architecture Definition and Layer Definition) [4] respectively. This document provides details on the design, the implementation and the configuration/deployment of the services that reside in the Grid Infrastructure Services Layer.

## 1.1. Scope of the Akogrimo Grid Infrastructure Services Layer

The Akogrimo Grid Infrastructure Services Layer is placed in the middle of the Akogrimo architecture and is responsible for providing the core Grid functionalities. The following figure gives a conceptual overview of the layers defined in the Akogrimo infrastructure.

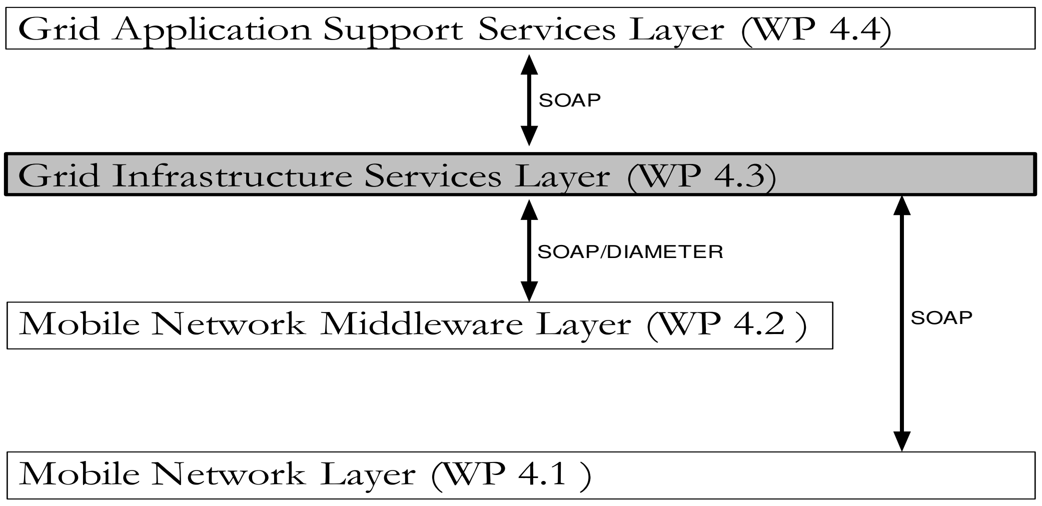
Grid Application Support Services Layer (WP 4.4)
SOAP
Grid Infrastructure Services Layer (WP 4.3)
SOAP/DIAMETER
Mobile Network Middleware Layer (WP 4.2 )
SOAP
Mobile Network Layer (WP 4.1 )

**Figure 1: Layered diagram of Akogrimo**

In particular, the layers that comprise the Akogrimo architecture are:

• **Mobile Network Layer (WP 4.1)**

This layer is responsible for the provisioning and management of the network. It deals with issues such as mobility, QoS provisioning and handover.

• **Mobile Network Middleware Layer (WP 4.2)**

This layer is in charge of context management (including SIP enriched presence provisioning and handling), local and grid semantic service discovery as well as authentication, authorization, accounting, auditing and charging.

• **Grid Application Support Services Layer (WP 4.4)**

This layer is responsible for the provisioning and management of the VO and workflow orchestration.

The Grid Infrastructure Services Layer is positioned on top of the Mobile Network Middleware Layer and below the Application Services Layer and is able to establish communication with all the layers, including the Mobile Network Layer. Its key aim is the provisioning and management of the Akogrimo Grid through Open Grid Services Architecture (OGSA) [5] compliant grid services. OGSA aims to define and standardize an open architecture for Grid-based systems.

Although up to now the OGSA specification is still in a draft form, it has already identified the most important services that should exist in Grid systems. Depending on the Akogrimo specific needs and key goals, a number of these services have been chosen and incorporated into the Grid Infrastructure Services Layer.

These OGSA compliant services are built upon the Web Services Resource Framework (WSRF) [6], which is a family of specifications developed by the OASIS [7] that specifies the way of developing and managing stateful Web Services that are required by the OGSA specification. The development of the chosen services was based on the Globus Toolkit 4 (GT4) [8] developed by The Globus Alliance [9] and WSRF.NET [10] platforms, both of them including a full implementation of the WSRF specification.

## 1.2. Services Overview

The basic role of the WP4.3 layer is to manage the execution of the services on request of the Grid Application Support Service Layer, aiming to fulfil the Grid requirements, addressing the performance issues in a transparent way to the user and conforming to the determined Service Level Agreement (SLA). The services one can encounter in the Grid Infrastructure Services Layer, addressing the specific requirements as identified by the consortium and the OGSA specification, are listed below:

- **Execution Management Service (EMS):** Apart from the execution of the services that are offered to the Akogrimo clients, this service deals with the advertisement, negotiation, discovery and reservation of the resources needed for the execution.
- **Data Management Service (DMS**): This service deals with the discovery, transfer and access of large data within the Grid.
- **SLA Enforcement services:** This group of services is responsible for the detection of SLA violations and enforcement of the SLA contractual terms.
- **Metering Service:** This service is supplementary to the monitoring and accounting services and deals especially with the measurement of resource usage.
- **Monitoring Service:** This service provides for the monitoring of the resource consumption during the execution and is the link between the Metering service and the SLA Enforcement group of services.
- **Policy Manager Service:** This service comprises the functionality concerned with the management of rules and the policies which apply in the execution of services within the Akogrimo framework.
- **Semantic Service Discovery Service (SSDS):** This service provides for the discovery of services in the Mobile Network Middleware Layer, acting as a link between the discovery process performed in the Grid Infrastructure Services and Network Middleware Layers.

In the following figure, a high level view of the WP4.3 services and their interactions is depicted.

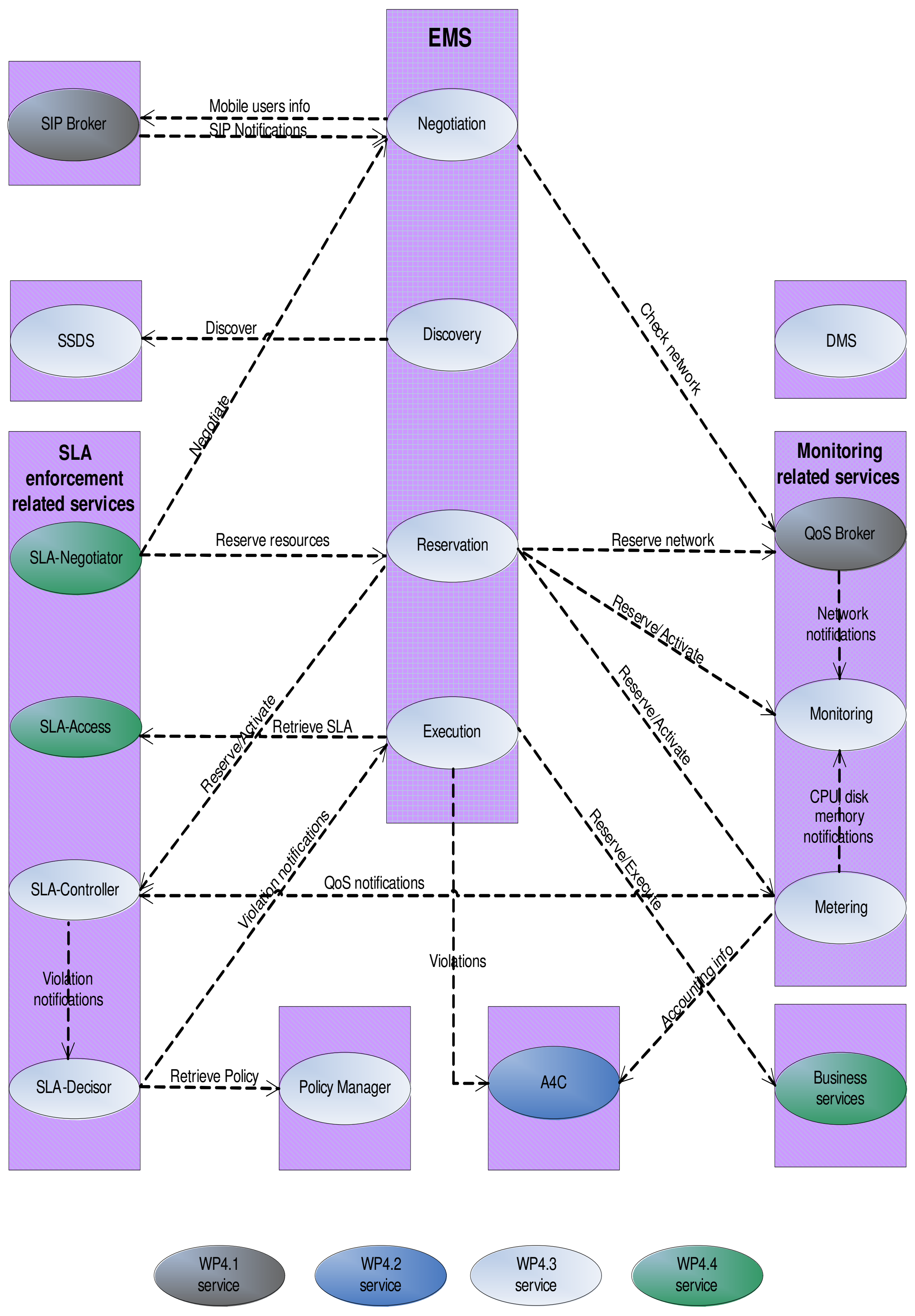

**Figure 2: WP4.3 services interactions overview**

# 2. Implemented Akogrimo Grid Infrastructure Services

## 2.1. Execution Management Service (EMS)

The Execution Management Service (EMS) comprises the central controller of the business service[1] execution in Akogrimo Framework and is developed on the GT4 platform. Built on the basis of the OGSA and WSRF specifications, the EMS is responsible for finding candidate services, selecting the most suitable execution location, making advance reservation on selected resources, initiating and managing/monitoring the execution of a business service. In order to address these tasks and at the same time ensure continuous conformation to the terms of the SLA contract as well as awareness of changes in the location of mobile users, the EMS works closely together and involves numerous interactions with many other Akogrimo services.

### 2.1.1. EMS Design

#### General Concepts

Although, the EMS is designed so as to address the specific needs of the Akogrimo project with respect to its overall architecture, it builds heavily on the OGSA Framework specifications. The challenge was to design an ***OGSA compliant*** EMS service that can guarantee ***QoS enforcement*** by exploiting the functionalities offered by the rest of the services in the Akogrimo Framework, while at the same time addressing the main goal of the Akogrimo project which is the development of a ***mobile aware Grid***. These requirements have been met by the Akogrimo EMS in the following ways:

**Mobile awareness:** By interfacing with the SIP Broker service the EMS is able to keep track of the current location of the "mobile" users. The SIP Broker acts as a gateway service in front of WP4.2 SIP Server responsible for receiving and translating requests made by the EMS. The SIP Broker interacts with the SIP Server in order to find the current service End Point Reference (EPR) and its availability status. When there is a change in the client's status, a notification message is generated by the SIP Broker. This notification message includes information about the availability/unavailability of the mobile client. In case of availability, the new location of the mobile client and the list of services that are hosted there are included in the notification message. These notifications are propagated to the EMS through the use of a WS-Notification mechanism, established between the EMS and the SIP Broker. Whenever such a notification is received, the EMS is responsible for updating the registry it uses to store info for mobile users and reallocate resources in case the shift takes place during the actual execution of a business service. For more information refer to Annex A.

[1] The term *business service* is used though out this document to refer to the application services that are offered to the clients through the Akogrimo framework

**OGSA compliancy:** The EMS addresses the basic set of requirements that must be met by an execution management service according to the OGSA specifications:

- ***Discovery of candidate services/Selection of winner service*:** The EMS is able to discover a candidate set of business services that meet the clients' needs and restrictions, as expressed in their SLAs. Available mobile services are discovered through the use of a mobile registry that is constantly updated with information by the SIP Broker, whereas non-mobile services are discovered through the use of an Index service that is included and deployed by default into the Java WS container of the GT4 as part of the WS MDS subsystem. The GT4's Index service is so versatile a service that can serve both as a local index as well as a VO-wide index. The Index service that is deployed in the same location as the EMS, acts as the central VO index to which all Index services deployed in different containers through out the underlying Grid are registered. The "advertisement" of the business service includes information related to low level performance parameters such as memory, CPU and disk storage. All available business services are registered to their local Index service. Local registrations propagate to the EMS's Index Service after a specific period of time. The EMS in this case acts as a gateway service in front of the central Index service, which is responsible for making requests to and interpreting the responses from the Index service. The EMS queries its Index service to find resources matching the requested by the client low level performance parameters and returns a candidate set of available resources.[2] Once the list of candidate locations is known, the EMS decides where the execution should really take place by following very simple rules, based on low level performance parameters. However, the selection mechanism could be enhanced so as to involve different selection algorithms that optimize different objective functions or attempt to enforce different policies or SLAs.
- ***Advance reservation of resources:*** Two different types of advance reservation are supported by the EMS: **(1) Service resource reservation** and **(2) Network resource reservation**. **Advance service resource reservation** is achieved by creating instance resources of all services involved in the execution (the business service, the SLA-Enforcement and Monitoring group of service). Those services are developed on the WS-Resource Factory pattern [12][13] that enables the management of multiple resources through the use of a factory service that creates instance resources of the service. Whenever the EMS wants to create a new business resource, it contacts the factory service that corresponds to the business service. The factory service returns to the EMS an EPR (Endpoint Reference)[3] to the newly created business resource. After obtaining the EPR to the resource, the EMS manages its lifecycle by setting the termination time of the resource equal to the time that the SLA-contract becomes invalid. Only the client for whom the business resource is created can access it. On the other hand, **advance network resource reservation** is achieved through the use of the QoS Broker service. The EMS contacts the QoS Broker service requesting a specific network bandwidth during the entire lifetime of the business service execution. Also, EMS will be able to achieve advance system resource reservation by reserving disk storage through the Data Manager Services once this functionality is fully supported by the latter. The resources are allocated at reservation start time and released when the SLA contact expires.

[2] Because of its advanced flexibility and functionality, it was decided that the EMS will use the GT4's MDS instead of the SSDS. The latter supports the GrSDS service (WP4.2) in the discovery process of Mobile Network Middleware Layer.

[3] The EPR is a pair of a Unique Resource Identifier (URI) and a key, which differentiates an instance of a resource from all other instances of the same resource.

- ***Initialization/Management/Monitoring of execution:*** Once the execution of the business service has started, it is managed and monitored in order to achieve continuous conformance to the contractual terms of SLAs. In case of execution failure or failure to meet the SLA criteria and depending on the explicit type of failure, EMS is able to reallocate the execution. In order to detect possible failures and maintain the state of the execution between possible reallocations, a fault-detection in conjunction with a recovery scheme is being used. The EMS establishes a WS-Notification mechanism with the business services in order to receive notification messages every time a property of the business resource changes its value. Resource properties provide to the EMS a view on the current state of the resource. Every time the EMS receives a notification message it stores the new value of the resource property. If a failure occurs during the execution of the business service and the creation of new business resource is needed (either on the same or different location), the EMS will set the resource properties to the last known ones before the failure occurs. This is a first approach to a recovery scheme used by the EMS that makes use of the WS-Resource specification in order to maintain the state of the business resources.

**QoS enforcement:** During the execution of the requested service the QoS requirements that are expressed in the SLA contract must be fulfilled through the management of the execution and the associated Grid resources. In order to achieve this, EMS exploits the functionalities offered by the Akogrimo Monitoring and SLA Enforcement groups of services to the fullest.

## Design Details

Instead of designing a complicated service responsible for all tasks involved in the execution management process, the EMS is a set of sub-services, with each of them addressing a specific task. Although from the client's perspective it appears to be a single Grid service, actually the EMS is a composition of the six Grid services listed below:

- *Core Gateway service*, which acts as a gateway service, offering a single-point of access to the EMS whilst "hiding" its details and complexity from the clients. It is therefore responsible for interpreting client's requests, delegating them to the appropriate EMS sub-services, orchestrating the interactions between the latter and returning the results to the clients.

- *Advertisement service,* which is used by the Service Providers (SPs) to advertise their services in the index service used by the EMS in the discovery process. The SPs run an Advertise client application on their local machines, providing all information necessary to form the advertisement of the service. The advertisement is registered with the index service which is running locally on the SP's container. The EMS index service, which acts as the SP-wide index service, automatically aggregates the advertisements from all index services in the SP domain.

- *Negotiation service,* which is responsible for interpreting requests for the negotiation of the terms of the client's SLA contracts. The Negotiation service contains a resource factory that creates Negotiation Resources (NRs). The NRs are persistent resources used to store information related to the negotiations. After a successful negotiation, the EPR to the NR that was created is returned to the Core Gateway service, which in turn passes it to the client.

- *Reservation service*, which is in charge of performing advance reservation on the service and network resources needed for the execution and monitoring of the business service. Upon successful reservation, a persistent Reservation Resource (RR) is created and all information related to the reservation is stored inside it. The EPR to the RR is returned to the Core service and through it to the client.

- *Execution service*, which is in charge of the coordination of the SLA Enforcement and Monitoring groups of services and the actual execution of the reserved business resources as well as being responsible to take corrective actions when needed. At the end of the execution of the business service, the result is returned to the Core Gateway service. The Core Gateway service filters the result and sends it to the client.

- *Discovery service*, which is responsible for finding available services, registered in the underlying Grid, that meet the QoS parameters defined in the SLA. It is designed so as to follow rules based on low level performance parameters related to the execution of the service, such as CPU, memory, network bandwidth and disk capacity as well as the availability of the service and the price that the client is willing to pay. It is built on top of the GT4's MDS Index service. The Discovery service is not only used by the Negotiation service but by the Reservation and Execution services when reallocation is needed.

A high level view of the EMS services and their internal interactions is depicted in Figure 3: EMS services and their internal interactions.

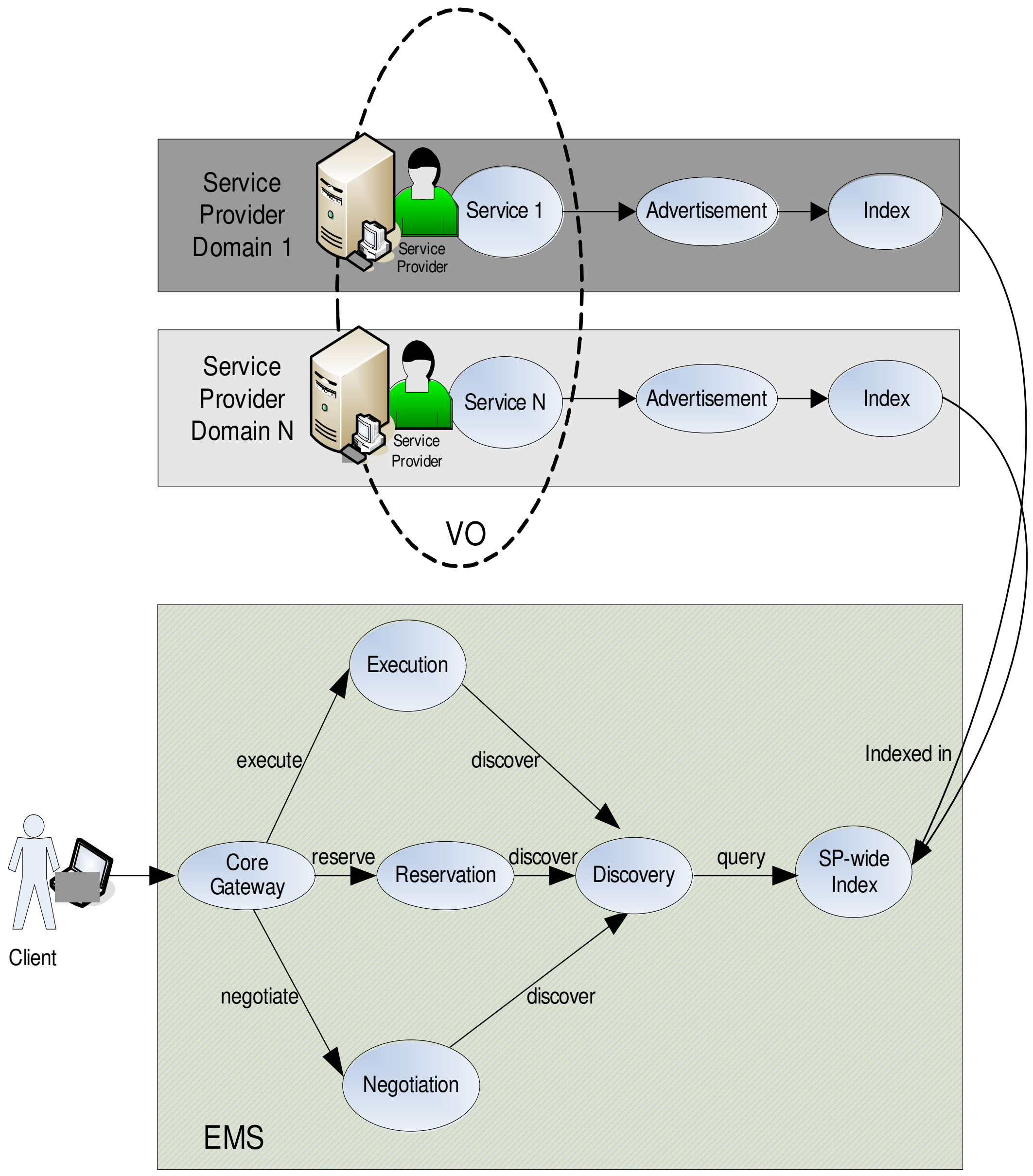

**Figure 3: EMS services and their internal interactions**

#### 2.1.1.1. Functionality

The functionality of the EMS is divided into four phases: 1) Advertisement of business services, 2) Negotiation and Discovery phase, 3) Advance Reservation phase and 4) Execution and Monitoring phase. The first phase is not connected to the rest and may take place at any time, whereas the remaining three are part of a sequence that leads to the actual execution of the business service.

***1) Advertisement of business services***

In this phase, the SP can advertise his services with QoS properties to the EMS. The SP must specify a unique within his domain name for the service (serviceID)[4], the URI of the service (serviceURI) as well as the values of low level parameters related to its performance. An example of the low level parameters that are used in action is shown in Figure 4: Example of low level QoS parameters. Figure 5: Sequence diagram for the Advertisement phase, shows the sequence diagram for this phase:

1. The SP runs an Advertise client application on his local machine, specifying a serviceID, a serviceURI and values for the low level performance parameters.
2. The Advertise client filters given values to check if they have accepted format and are within a logical range and then invokes the Advertisement service, which runs in the SP's container. In particular, at this point the Advertise client invokes the Register operation, requesting that the advertisement be included in the list of the advertised services by this SP.
3. The Register operation of the Advertisement service triggers the creation of a WS-Resource called Advertisement. Each service for sale in the VO is represented by an Advertisement resource in the SP's container.
4. Upon creation, the Advertisement resource registers with the Advertisement Index running locally on the SP's container, which gathers information on the advertisements in the SP's machine.
5. After some time, the local registrations will propagate to the EMS's Index service which will also cache the data stored in the SP's local Advertisement index service.

```
<QoSParams>

        <QoSItems>
                <QoSItem>
                        <param>cpuSpeed</param>
                        <paramValue>3.5</paramValue>
                        <paramType>MHz</paramType>
                </QoSItem>
                <QoSItem>
                        <param>diskSpace</param>
                        <paramValue>1</paramValue>
                        <paramType>GB</paramType>
                </QoSItem>
                <QoSItem>
                        <param>memory</param>
                        <paramValue>1</paramValue>
                        <paramType>GB</paramType>
                </QoSItem>
        </QoSItems>
</QoSParams>
```

**Figure 4: Example of low level QoS parameters**

[4] Parameter serviceID is a simple string. One serviceID that is actually used in action is "ECGDataAnalyzer". Although it can have any value, it is recommended to use a value that describes the functionality of the business service.

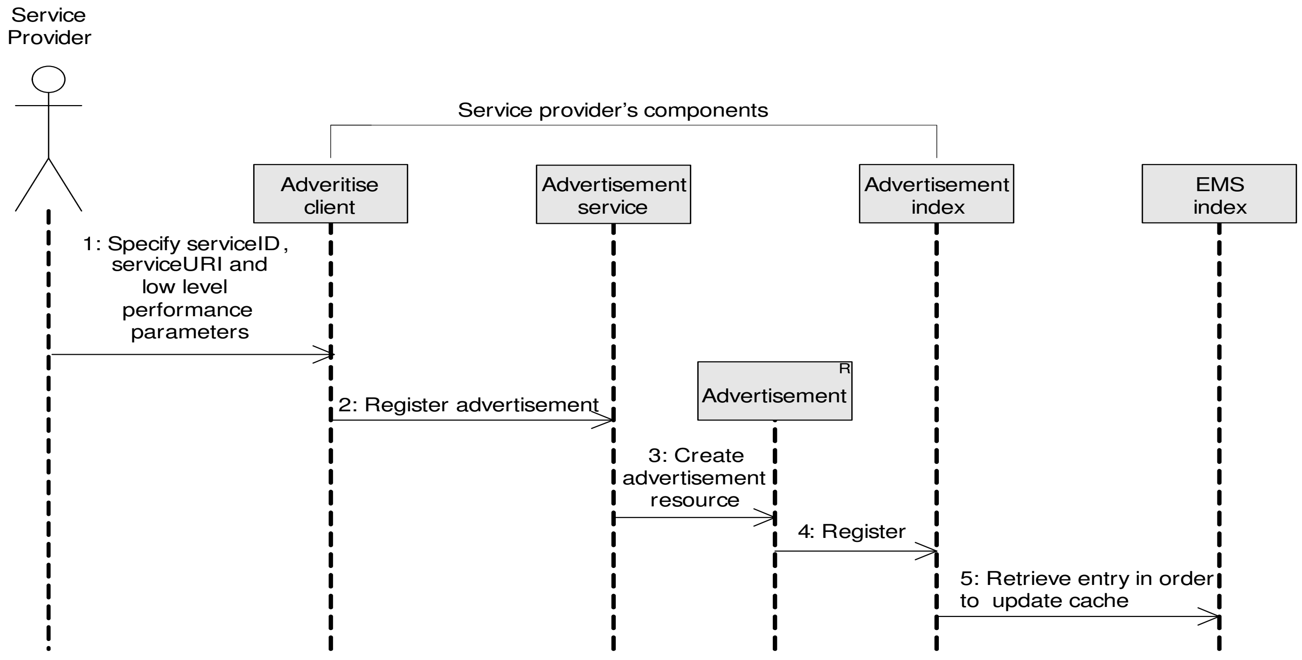
Service Provider
Service provider's components
Adveritise client
Advertisement service
Advertisement index
EMS index
1: Specify serviceID, serviceURI and low level performance parameters
2: Register advertisement
R
Advertisement
3: Create advertisement resource
4: Register
5: Retrieve entry in order to update cache

**Figure 5: Sequence diagram for the Advertisement phase**

### *2) Negotiation and Discovery Phase*

During this phase the EMS is in charge of discovering the resources needed for the execution of the business services based on low level QoS parameters passed by the SLA-Negotiator service. The phase begins with the SLA-Negotiator service asking the EMS to check for service and network availability by invoking the *checkResourcesAvailability* operation on it. Figure 6: Sequence diagram for the Negotiation and Discovery phase, shows the sequence diagram for this phase:

1. The SLA-Negotiator service invokes EMS Core service providing the serviceID, the clientID and desired values for the low level parameters.
2. The Core service checks the format of the input parameters and contacts the EMS Negotiation service if their format is valid. Otherwise, it returns a null value to the SLA - Negotiator.
3. The Negotiation service after processing the low level parameters invokes the EMS Discovery service.
4. If the requested service is not a mobile application, the Discovery service queries the EMS Index service trying to find service resources that meet the client's requirements. Otherwise, it checks its mobile registry, where information related to mobile application sent from the SIP Broker is gathered.
5. A list of candidate services is returned to the Negotiation service.
6. The first service in the list is selected.
7. The Negotiation service contacts the QoS Broker to check the network availability of this service.
8. If the client's network criteria are not met by this service, the service is removed from the list of candidates and steps 6 and 7 are repeated. In case the QoS Broker verifies the availability of the network, a Negotiation resource (NR) is created.
9. All information related to the negotiation request and the discovered resources is stored into the Negotiation resource.

10. The Negotiation service returns the EPR of the NR to the Core service.
11. Finally, the EPR to the NR is returned to the SLA-Negotiator service. In case no match is found the EPR returned has a null value.

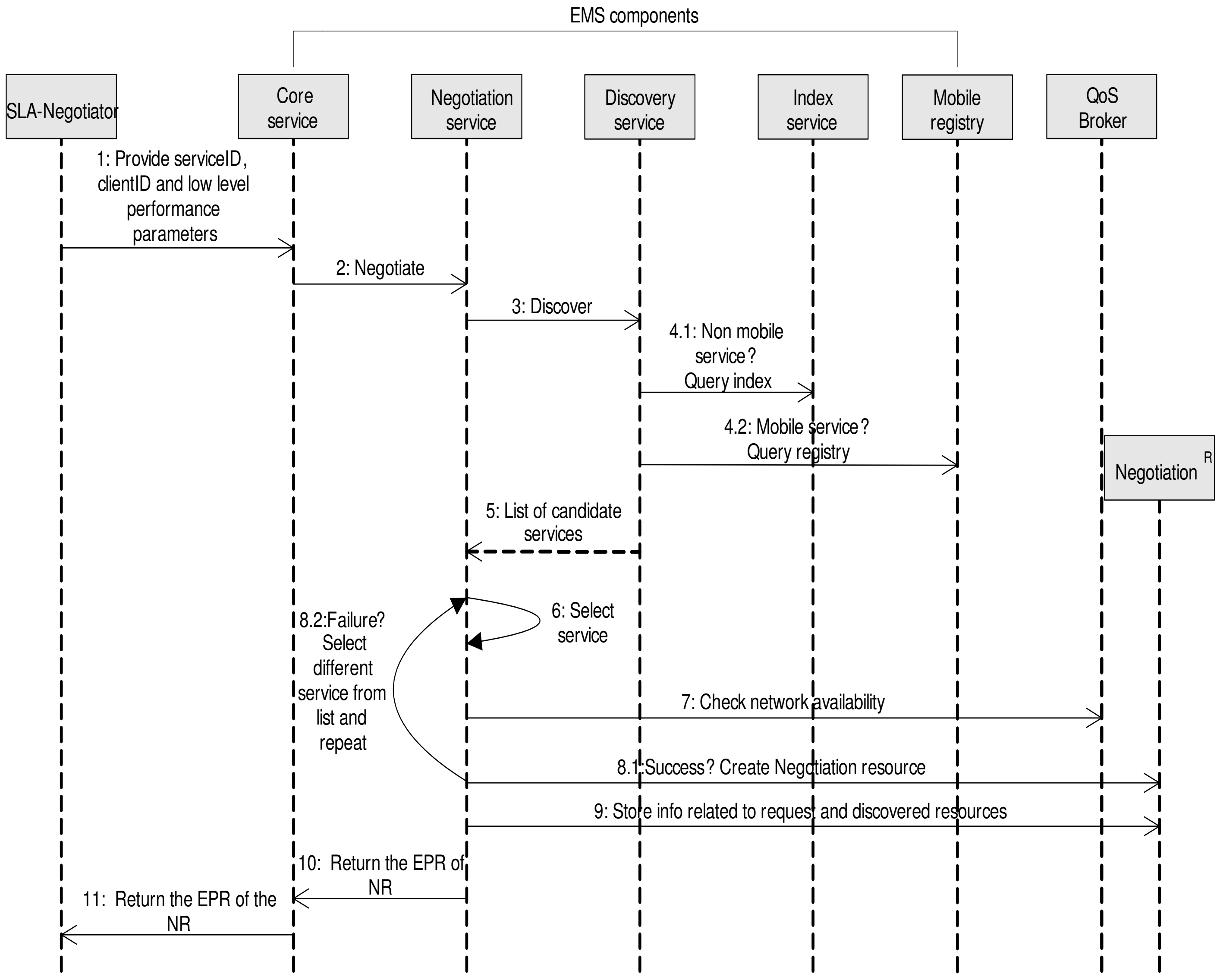
EMS components
SLA-Negotiator
Core service
Negotiation service
Discovery service
Index service
Mobile registry
QoS Broker
Negotiation R
1: Provide serviceID, clientID and low level performance parameters
2: Negotiate
3: Discover
4.1: Non mobile service? Query index
4.2: Mobile service? Query registry
5: List of candidate services
6: Select service
8.2:Failure? Select different service from list and repeat
7: Check network availability
8.1: Success? Create Negotiation resource
9: Store info related to request and discovered resources
10: Return the EPR of NR
11: Return the EPR of the NR

**Figure 6: Sequence diagram for the Negotiation and Discovery phase**

The high-level view of the interactions performed by the EMS as depicted in Figure 6: Sequence diagram for the Negotiation and Discovery phase in conjunction with Table 1: Services involved in the Negotiation phase of EMS provides a complete description of the design details and the behavior of the EMS during the Negotiation and Discovery phase (Table 1 doesn't contain the interactions between the subcomponents of the EMS).

**Table 1: Services involved in the Negotiation phase of EMS**

| Service | Interaction | Interface |
|---|---|---|
| QoS Broker | Step 7 | *boolean check_qos_availability(user, SLAid, qos_bundle)* |

***3) Advance Reservation Phase***

After a successful negotiation phase, the SLA-Negotiator service asks EMS to perform advance service and network reservation on the resources that were discovered during the negotiation phase by invoking the *performAdvanceReservation* operation on it. Figure 7: **Sequence diagram for the Advance Reservation phase**, shows the sequence diagram for this phase:

1. The sequence begins with the SLA-Negotiator service invoking the EMS Core service, providing the EPR to the NR that was obtained at the end of the Negotiation phase.
2. The Core service delegates the task to the EMS Reservation service.
3. The Reservation service retrieves the information stored inside the Negotiation resource during the Negotiation phase (QoS parameters, start and end time of the SLA, URI of the discovered business service etc).
4. The Reservation service creates a business resource and obtains its EPR.
5. By using the end time defined in the SLA, the Reservation service is able to manage the lifecycle of the newly created business resource.
6. At next step, the Reservation service creates an SLA-Controller resource.
7. The termination time of the SLA-Controller resource is set to the time the SLA expires.
8. The Reservation service creates a Metering resource.
9. Upon creation, the Reservation service sets the termination time of the Metering resource to the time the SLA expires.
10. The Reservation service creates a Monitoring resource.
11. The termination time of the Monitoring resource is set to the time the SLA expires.
12. Once the advance service reservation has been performed, the Reservation service proceeds with the reservation of the network resources by invoking the QoS Broker service
13. The EMS creates a Reservation resource.
14. The EPRs to the resources that were created in previous steps as well as info retrieved from the Negotiation resource is internally stored into the Reservation resource.
15. The Reservation service then returns the Reservation EPR to the Core service.
16. Finally, the Core service returns the Reservation EPR to the SLA-Negotiator service.

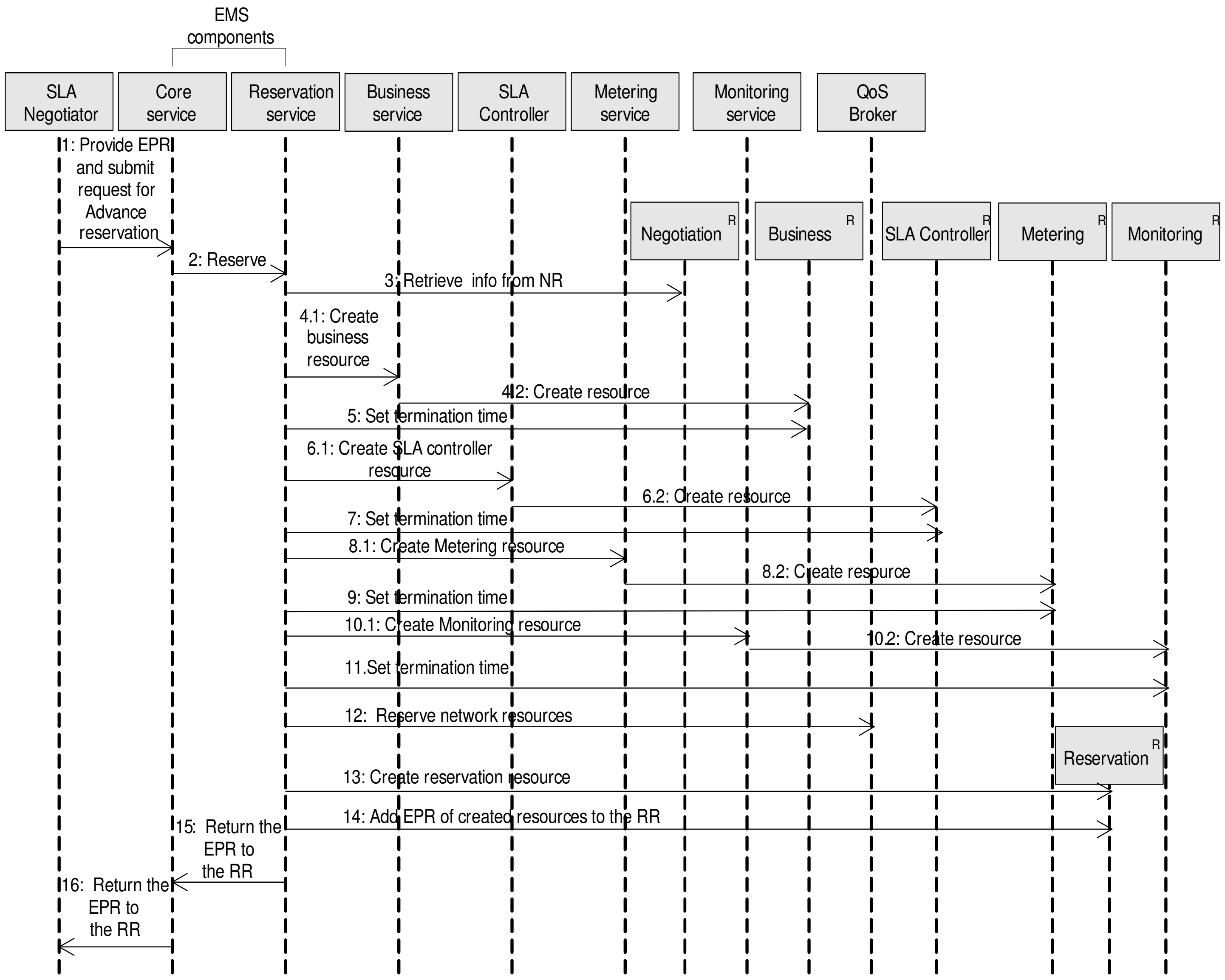

**Figure 7: Sequence diagram for the Advance Reservation phase**

The high-level view of the interactions performed by the EMS as depicted in Figure 7: **Sequence diagram for the Advance Reservation phase** in conjunction with Table 2: Services involved in the Advance Reservation phase provides a complete description of the design details and the behavior of the EMS during the Advance Reservation phase (Table 2 doesn't contain the interactions between the subcomponents of the EMS).

**Table 2: Services involved in the Advance Reservation phase**

| Service | Interaction | Interface |
|---|---|---|
| Business | Step 4 | *EndpointReferenceType createResource()* |

| | | |
|---|---|---|
| | Step 5 | *org.oasis.wsrf.lifetime.SetTerminationTimeResponse setTerminationTime(org.oasis.wsrf.lifetime.SetTerminationTime setTerminationTimeRequest)* |
| SLA - Controller | Step 6 | *EndpointReferenceType create(edu.virginia.cs.gcg.wsrf.ArrayOfXmlElement portTypeInitializers)* |
| | Step 7 | *org.oasis.wsrf.lifetime.SetTerminationTimeResponse setTerminationTime(org.oasis.wsrf.lifetime.SetTerminationTime setTerminationTimeRequest)* |
| Metering | Step 8 | *EndpointReferenceType createResource()* |
| | Step 9 | *org.oasis.wsrf.lifetime.SetTerminationTimeResponse setTerminationTime(org.oasis.wsrf.lifetime.SetTerminationTime setTerminationTimeRequest)* |
| Monitoring | Step 10 | *EndpointReferenceType createResource()* |
| | Step 11 | *org.oasis.wsrf.lifetime.SetTerminationTimeResponse setTerminationTime(org.oasis.wsrf.lifetime.SetTerminationTime setTerminationTimeRequest)* |
| QoS Broker | Step 12 | *boolean set_qos(user, SLAid, qos_bundle)* |

### *4) Execution and Monitoring Phase*

In this phase, the EMS is responsible for the initialization and management, to completion, of the business service execution. During the execution, the EMS receives notification messages from the SLA-Decisor regarding failures to meet its SLA and the applied policy. If the execution fails or if the SLA violation demands it, EMS will try to reallocate resources. Reallocation implies discovery of new location and creation on the fly of service resources. The status of the newly created resources is set to the previous one and execution is restarted. It is also possible that the execution resources don't exist anymore due to possible container restarts. EMS is able to understand when the failure derives from resource loss and will first try to recreate duplicate resources in the same locations.

The sequence begins with a Service Agent requesting the execution of a business service by invoking the *performExecution* operation on the EMS Core Gateway service. Figure 8: Sequence diagram for the Execution phase, shows the sequence diagram for this phase:

1. The Service Agent provides the RR EPR, the client ID, the service method that he wishes to invoke as well as the input needed by it.
2. The Core Gateway service filters the input and delegates the task to the EMS Execution service.
3. At first step, the Execution service retrieves the information needed for the execution from the Advance Reservation resource (SLA-Controller resource EPR, Business resource EPR, etc) while updating certain Advance Reservation resource properties used for auditing. The following 3 steps (Step 4, 5 and 6) are performed only if it is the first request for execution related to the specific SLA, otherwise the steps have already been performed in previous invocation and the control is handed over to Step 7.

4. The QoS parameters that are specified in the client's SLA are retrieved from the SLA-Access service.
5. Using the EPR of the SLA-Controller resource, the Execution service activates it and binds it to the specific SLA, business resource and metering resource.
6. On the next step, the Execution service activates the Monitoring resource.
7. Once the Monitoring resource is activated, the Execution service activates the Metering resource on the server hosting the business resource.
8. After setting up the services that perform the monitoring and SLA enforcement, the Execution service initiates the execution of the business resource.

9-13. During the execution, the Metering service sends notification messages to Monitoring service. These notifications are being forwarded to the SLA-Decisor through the SLA-Controller. When a violation is detected, the SLA-Decisor propagates the notification to the EMS along with the policy that should be applied. The EMS filters the message and propagates it the A4C system and reallocates resources if needed.

14. At the end of the execution, the Execution service obtains its output.
15. The Execution service stops the execution of the Metering resource[5].
18. The result of the service execution is reported to the Core Gateway service.
19. On last step, the result of the execution is returned to the Service Agent.

[5] To reduce SOAP message traffic, the Monitoring and SLA-Controller resources are deactivated only at final invocation. The EMS destroys all resources related to the execution of the service when the SLA expires. The Reservation resource is not destroyed as it used for auditing purposes.

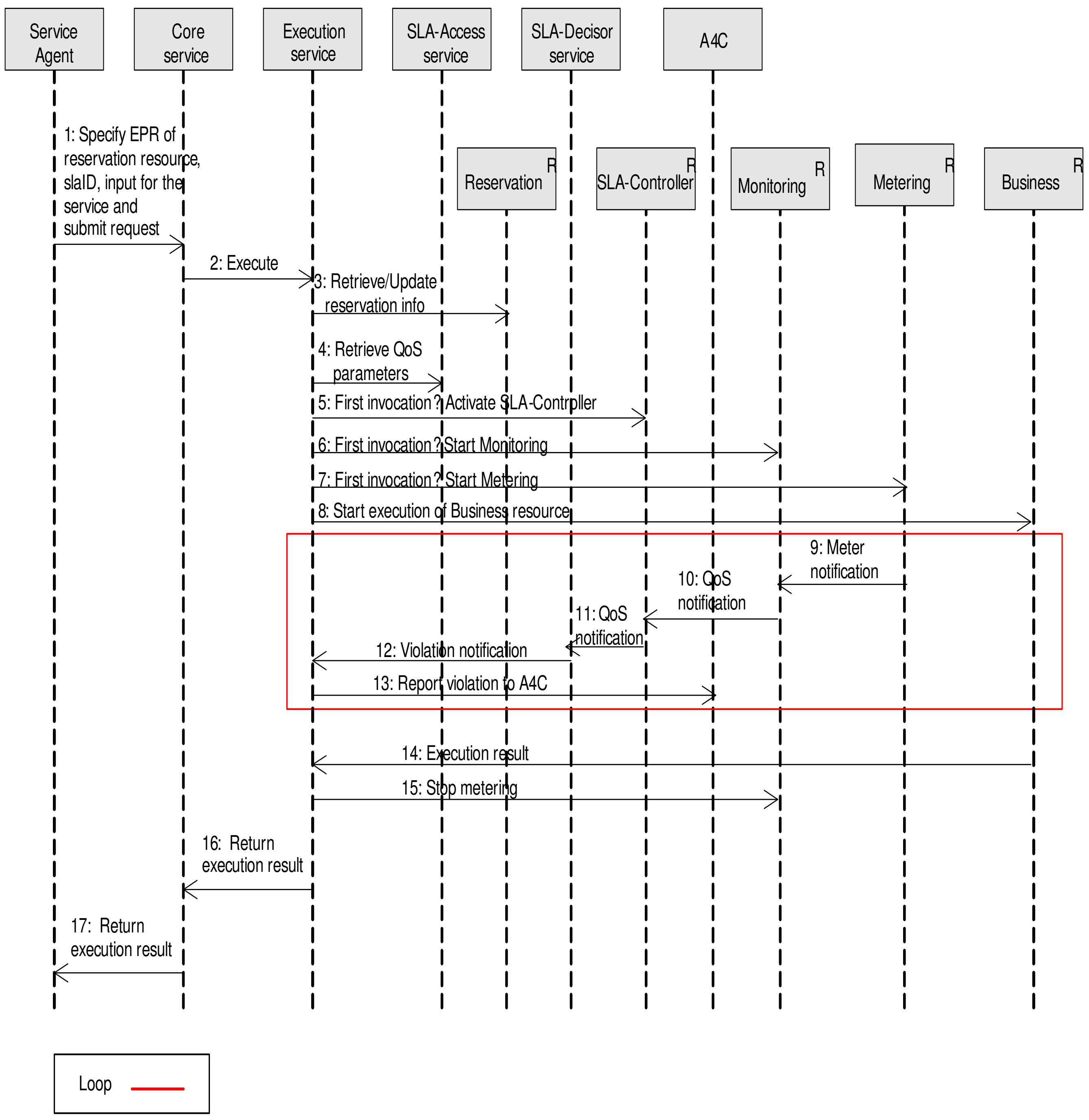
Service Agent
Core service
Execution service
SLA-Access service
SLA-Decisor service
A4C
Reservation
SLA-Controller
Monitoring
Metering
Business
R
1: Specify EPR of reservation resource, slaID, input for the service and submit request
2: Execute
3: Retrieve/Update reservation info
4: Retrieve QoS parameters
5: First invocation? Activate SLA-Controller
6: First invocation? Start Monitoring
7: First invocation? Start Metering
8: Start execution of Business resource
9: Meter notification
10: QoS notification
11: QoS notification
12: Violation notification
13: Report violation to A4C
14: Execution result
15: Stop metering
16: Return execution result
17: Return execution result
Loop

**Figure 8: Sequence diagram for the Execution phase**

The high-level view of the interactions performed by the EMS as depicted in Figure 8: Sequence diagram for the Execution phase, in conjunction with

Table 3: Services involved in the Execution phase provides a complete description of the design details and the behavior of the EMS during the Advance Reservation phase (Table 3 doesn't contain the interactions between the subcomponents of the EMS).

**Table 3: Services involved in the Execution phase**

| Service | Interaction | Interface |
|---|---|---|
| SLA-Controller | Step 5 | *void activeServiceController(String serviceId, objQoS objQoSData, String slaID)* |
| | Not shown in sequence – performed at final invocation | *void destroy()* |
| SLA-Access | Step 4 | *org.akogrimo.www.SLAManagement.SLAAccess.ObjQoS getQoSParameters(String slaID)* |
| Monitoring | Step 6 | *String startMonitoring(org.akogrimo.www.namespaces.notifications.Monitoring.StartMonitoringRequest parameters)* |
| | Not shown in sequence – performed at final invocation | *String stopMonitoring(int parameters)* |
| Metering | Step 7 | *Boolean startMetering(String serviceID, String clientID)* |
| | Step 15 | *Boolean stopMetering(String serviceID, String clientID)* |
| A4C | Step 13 | *To be defined* |
| Business Service | Step 8 | *Not standard (depends on the business service)* |

### 2.1.1.2. Use Cases

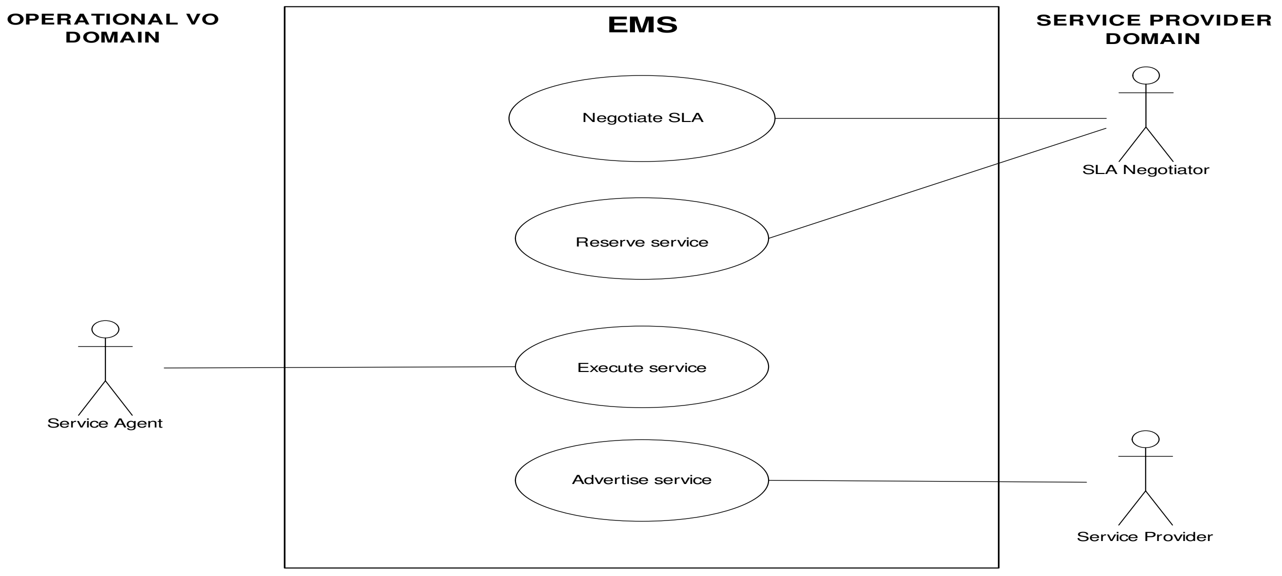

**Figure 9: EMS use case**

**Table 4: Advertise service use case**

| | |
|---|---|
| **Use Case ID** | UC_EMS_1 |
| **Use Case Name** | Advertise service |
| **Initiator** | Service Provider |
| **Primary Actor** | Advertise client |
| **Additional Actors** | EMS Advertisement service, MDS Index service |
| **Description** | The SP invokes EMS Advertise service in order to advertise the existence of a business service in one of the machines in his domain. |
| **Pre-condition** | A full GT4 installation and specific configuration is required in the machine that the business service is running (See section 4.1.2.2 for details). Also, the Advertise client must be installed in the SPs machine. |
| **Post-condition** | An "advertisement" for the service is created and registered to the SPs Advertisement Index service in the form of a persistent WS-Resource. All advertisements are propagated to the EMS index service after a specific period of time. |
| **Use Case Functionality** | |
| **Sequence** | See Section 2.1.1.1 for a full walk through of the sequence. |
| **Alternatives** | None |
| **Exceptions** | The Advertise client filters the values of the parameters that are passed by the SP. In case one or more parameters have wrong format or an irrational value, the Advertise client blocks the request and asks from the SP to re-enter the values of the parameters. |
| **Use Cases used** | None |
| **Further Information** | |
| **Particular Requirements** | None |
| **Open Issues** | None |
| **Information Requirements** | None |

**Table 5: Negotiate SLA contract use case**

| | |
|---|---|
| **Use Case ID** | UC_EMS_2 |
| **Use Case Name** | Negotiate SLA Contract |
| **Initiator** | SLA-Negotiator service |
| **Primary Actor** | EMS Core |
| **Additional Actors** | EMS Negotiation, EMS Discovery, QoS Broker |
| **Description** | During this phase the EMS is in charge of discovering the resources needed for the execution of the business services based on low level QoS parameters passed by the SLA-Negotiator service |
| **Pre-condition** | All services involved in the Negotiation and Discovery phase (See Figure 6: Sequence Diagram For The Negotiation And Discovery Phase) must be up and running. |
| **Post-condition** | All information related to the negotiation request and the discovered resources is stored into the EMS in the form of a persistent WS-Resource. The EPR to this resource is returned to the SLA-Negotiator service. |
| **Use Case Functionality** | |
| **Sequence** | See Section 2.1.1.1 for a full walk through of the sequence. |
| **Alternatives** | None |
| **Exceptions** | • EMS filters the values of the parameters that are passed by the SLA-Negotiator. In case one or more parameters have wrong format or an irrational value, EMS blocks the request and reports error to the SLA-Negotiator. <br> • In case a fatal exception occurs that could not be handled by the EMS, a RemoteException is thrown. |
| **Use Cases used** | None |
| **Further Information** | |
| **Particular Requirements** | None |
| **Open Issues** | None |

| | |
|---|---|
| **Information Requirements** | None |

**Table 6: Reserve resources use case**

| | |
|---|---|
| **Use Case ID** | UC_EMS_3 |
| **Use Case Name** | Reserve resources |
| **Initiator** | SLA-Negotiator service |
| **Primary Actor** | EMS Core |
| **Additional Actors** | EMS Reservation, QoS Broker, Metering, Monitoring, SLA-Controller, Business services |
| **Description** | The SLA-Negotiator invokes EMS in order to perform reservation on the resources discovered during the Negotiation and Discovery phase. |
| **Pre-condition** | The SLA-Negotiator must pass EMS a valid EPR to a negotiation resource, i.e. the negotiation of the SLA must always precede the reservation.<br>All services involved in the Advance Reservation phase (See Figure 7: **Sequence diagram for the Advance Reservation phase**) must be up and running. |
| **Post-condition** | All information related to the reservation request is stored into the EMS in the form of a persistent WS-Resource. The EPR to this resource is returned to the SLA-Negotiator service. |
| **Use Case Functionality** | |
| **Sequence** | See Section 2.1.1.1 for a full walk through of the sequence. |
| **Alternatives** | None |
| **Exceptions** | • In case the business service is not available in the discovered location, the EMS will try to discover alternate locations that meet the client's requirements. If no such location is found, EMS reports failure to the SLA-Negotiator.<br>• The Negotiation EPR that is provided by the SLA-Negotiator service does not exist. This means that the EPR in speak is not valid because the Negotiation EPRs are persistent. EMS reports failure to the SLA-Negotiator.<br>• In case a fatal exception occurs that could not be handled by the EMS, a RemoteException is thrown. |

| | |
|---|---|
| **Use Cases used** | None |
| **Further Information** | |
| **Particular Requirements** | None |
| **Open Issues** | None |
| **Information Requirements** | None |

**Table 7: Execute business service use case**

| | |
|---|---|
| **Use Case ID** | UC_EMS_4 |
| **Use Case Name** | Execute business service |
| **Initiator** | Service Agent |
| **Primary Actor** | EMS Core |
| **Additional Actors** | EMS Execution, Metering, Monitoring, SLA-Access, SLA-Controller, SIP Broker, A4C |
| **Description** | The Service Agent requests the execution of a business service on resources reserved during the Advance Reservation phase. |
| **Pre-condition** | The Service Agent must pass EMS a valid EPR to a reservation resource, i.e. the reservation of the resources must always precede the execution.<br>All services involved in the Execution phase (See Figure 8: Sequence diagram for the Execution phase) must be up and running. |
| **Post-condition** | All information related to the execution of the service is stored into EMS. The result of the execution is returned to the Service Agent. |
| **Use Case Functionality** | |
| **Sequence** | See Section 2.1.1.1 for a full walk through of the sequence. |
| **Alternatives** | None |

| | |
|---|---|
| **Exceptions** | • In case the business service is not available in the discovered location, the EMS will try to recreate a duplicate resource on same location.<br>• In case the execution fails or a violation notification is received during the execution requesting reallocation, EMS will discover alternate locations that meet the client's requirements and proceed to reallocation. If no such location is found, EMS reports failure to the Service Agent.<br>• The Reservation EPR that is provided by the Service Agent does not exist. This means that the EPR in speak is not valid because Reservation EPRs are persistent. EMS reports failure to the Service Agent.<br>• In case a fatal exception occurs that could not be handled by the EMS, a *RemoteException* is thrown. |
| **Use Cases used** | None |
| **Further Information** | |
| **Particular Requirements** | None |
| **Open Issues** | None |
| **Information Requirements** | None |

### 2.1.1.3. *Interactions with other services*

The following table summarizes the interactions that EMS has with other Akogrimo services.

**Table 8: EMS interactions with other Akogrimo services**

| Akogrimo service | Interaction | Protocol |
|---|---|---|
| WP4.1 QoS Broker | -To use the network services of WP4.1.<br>-To request network resources availability and reservation | SOAP/HTTP |
| WP4.1 SIP Broker | -To find out the current location of mobile services<br>-To receive notifications related to movement of client from on mobile terminal to another | SOAP/HTTP |
| WP4.2 A4C | To propagate violation notifications for accounting purposes (to be implemented) | DIAMETER |
| WP4.3 Metering | To start and stop the metering of a business service execution | SOAP/HTTP |
| WP4.3 Monitoring | To start and stop the monitoring process | SOAP/HTTP |

| | | |
|---|---|---|
| WP4.3 SLA-Decisor | To receive notifications related to violations of the SLAs (to be implemented) | SOAP/HTTP |
| WP4.3 SLA-Controller | To activate the SLA-Enforcement for the execution of the business service | SOAP/HTTP |
| WP4.3 SSDS | To perform discovery of candidate locations (to be implemented) | SOAP/HTTP |
| WP4.3 Data Management | To perform disk storage reservation<br>(to be implemented) | SOAP/HTTP |
| WP4.4 SLA-Negotiator | -To check service and network availability<br>-To confirm reservations | SOAP/HTTP |
| WP4.4 Business services | -To create resources and manage their lifecycle<br>-To monitor their execution | SOAP/HTTP |
| WP4.4 BP Enactment | To initiate an execution and retrieve the results of it | SOAP/HTTP |
| WP4.4 SLA-Access | To receive the QoS parameters that have to be met | SOAP/HTTP |

### 2.1.2. EMS Implementation

The EMS has been implemented using the GT4 platform and runs under the standalone container that the toolkit offers. GT4 is an open source Grid middleware that provides the necessary functionality required to build and deploy fully operational Grid Services. It includes software for security, information infrastructure, resource management, data management, communication, fault detection, and portability. It supports the core specifications that define the Web Services architecture. It also supports and implements the WS-Security [19] and other specifications relating to security, as well as the WSRF, WS-Addressing and WS-Notification [15] specifications used to define, name, and interact with stateful resources.

#### *2.1.2.1. Involved Technologies*

The GT4 includes many high-level services, developed to take advantage of the potentials that the toolkit offers to the fullest. Leveraging existing high-level services is a hallmark of application development. Among the advantages of this approach is the one of advanced functionality and flexibility as well as the encouragement of service reuse. Depending on the explicit nature of the EMS, we have decided to leverage the GT4's WS-GRAM [22] and MDS4 [23] functionalities into it, using Java WS Core, and in this way create a much more powerful and flexible version of the EMS.

WS-GRAM, the core of the GT execution management, providing services to submit jobs and monitor their execution on a GT4 based Grid. "Jobs" within the WS-GRAM framework are considered to be binary executable files. Not all applications are good candidates for exposure as a Web Service. The WS-GRAM concept and the Web Service concept are two mutually exclusive approaches. WS-GRAM is meant to be used in situations in which the job requested by the client is not available as a Web Service but in the general form of a binary executable file. Through the usage of the WS-GRAM Client Java API and Java WS Core, the WS-GRAM functionality has been successfully integrated into the EMS. EMS is able to address this need for management of applications in the form of executable files, while at the same time is able to manage the execution of application-specific Web Services. From the client's perspective, there is no difference in the

EMS behaviour, whether it handles the execution of an executable file or a Web Service. The use of WS-GRAM is transparent to the client of the EMS.

MDS4 is a WSRF implementation of information services released with GT4. MDS4 builds on query, subscription, and notification protocols and interfaces defined by the WSRF and WS-Notification families of specifications and implemented by the GT4 Web Services Core. Building on this base, we have successfully integrated the functionality of the Index service, one of the two high-level services that the MDS4 provides, into the EMS and use it for advertisement and discovering purposes.

#### 2.1.2.2. Configuration and Deployment

In order for EMS to be fully functional, some basic setup, mostly related to WS-GRAM and MDS4 is needed. First of all, EMS should ideally be deployed in one machine per service provider domain. A full GT4 installation is required on each machine hosting the EMS so that the additional features of the EMS related to the WS-GRAM and MDS4 will be enabled and fully functional.

A Java core-only installation will not be enough for the machines that host the business services as it does not include the GT4 Index service necessary for the discovery process. When running the GT4 installer on each of these machines, the *wsmds* target must be specified. The local Index services of the machines (*downstream* Index services) must be registered to the EMS' Index service (*upstream* Index service).

In order to execute the business service through WS-GRAM, a full GT4 installation is strictly required on the machines that are hosting them. Each of these machines must be configured so as to trust the CA that issued the certificate that the GT4 installation on the machine hosting EMS is configured to work with.

### 2.1.3. Security Considerations

Our primary concern is to protect the EMS system from potential exploits while at the same time optimize the system performance. All communications among the EMS and the services at the Grid Layer should be secure and thus require mutual authentication. The same applies for the EMS interactions with services that reside in different layers and domains.

The EMS intends to make extended use of GSI (Grid Security Infrastructure) [18], the security framework provided by GT4. More specifically, GSI is a set of tools, libraries and protocols - layered on top of SSL with mutual authentication performed via X.509 certificates - used in GT4 to allow users and applications to securely access resources. It includes mechanisms implemented on top of the WS-Security [19] and WS-SecureConversation [20] specifications and also supports TLS.

EMS can use one or combinations of these security mechanisms, depending on the nature of the interaction. No matter what security scheme is used, it will be done so with digital signatures to guarantee authentication and integrity. Because encryption slows down the performance of any system, it will be used wherever the content of the messages that are exchanged is critical and thus, obtaining it poses a security threat to the EMS or any other Akogrimo service.

## 2.2. Data Management Service (DMS)

### 2.2.1. DMS Design

The Data Management Service has been designed and developed to satisfy the basic requirements of the Data Management in a Grid environment. Taking into account the functional requirements of the Akogrimo domain applications we have focused on three main areas: the transfer of data from one location to another, the storing of data and finally the access to the data stored. The Data Management Service has been designed taking into account the OGSA specification in order to be OGSA compliant.

After a deep survey of the available tools, it was decided to implement the Data Management Services based on OGSA-DAI (Open Grid Services Architecture Data Access and Integration) [11]. Thus, the architecture of the DMS is strongly related to the OGSA-DAI general concepts and architecture.

The DMS has been developed as a Web Service that exposes several interfaces. In addition, the DMS needs a client to be run on the machines where data needs to be copied. This client is a simple GridFTP server provided by GT4. One of the possible protocols on which OGSA-DAI relies for the data transfer is the GT4's GridFTP [24]; other protocols could be used but are not exploited by the DMS. As a consequence of the use of GT4, the DMS service implementation makes also use of the GSI provided by GT4 (see Security section for more details).

#### *2.2.1.1. Functionality*

The DMS is in charge of data handling; in particular it could be used for: 1) Storing data, 2) Retrieving data, 3) Transferring data from one location to another, 4) Querying stored data and 5) Accessing stored data. For each functionality used it is previously checked the identity of the user and the role inside the OpVO.

***1) Storing Data***

This functionality permits to upload and store data to the DMS. For each data file uploaded, a unique identifier is generated. It is possible to specify the filename and two attributes for each data file. The file uploaded is stored in a file type repository. In addition it is registered in a database, where the following parameters characterize the file: unique file identifier, file name, file owner, attribute1, and attribute2. The two attributes have been introduced to allow the user to characterize the file stored. An example of these attributes could be the following. Suppose the doctor wants to store two files about patient's data. The first file has been produced before a particular treatment has been given to the patient and the second file has been taken after the treatment has been completed. Attribute1 could be something like 'treatment Alpha' for both files. Attribute2 could be in one case 'before' and in one case 'after'. In this way the doctor could search all the files that have been produced before or after the treatment. More generally, it will be possible to find the file stored by specifying the two attributes instead of the unique file identifier in a second phase. The data file could be uploaded using the devoted Web Service interface provided by the DMS.

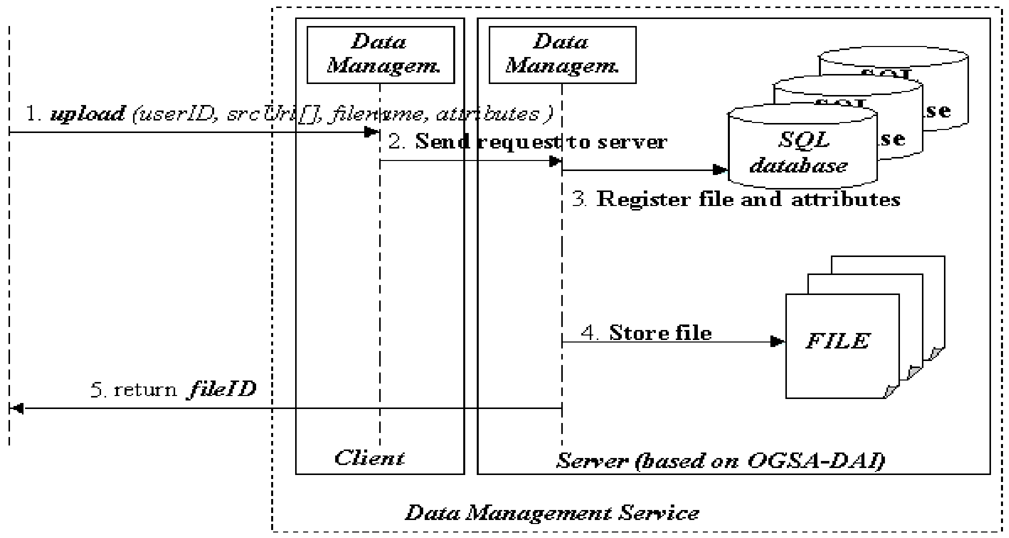
Data Managem.
Data Managem.
1. upload (userID, srcUrl[], filename, attributes )
2. Send request to server
SQL database
3. Register file and attributes
4. Store file
FILE
5. return fileID
Client
Server (based on OGSA-DAI)
Data Management Service

**Figure 10: Storing Data sequence diagram**

### *2) Retrieving Data*

This functionality allows retrieving the files that have been previously stored to the DMS. It is necessary to specify the filename of the data file and the destination URL where the file should be copied. The data file could be retrieved using the devoted Web Service interface provided by the DMS.

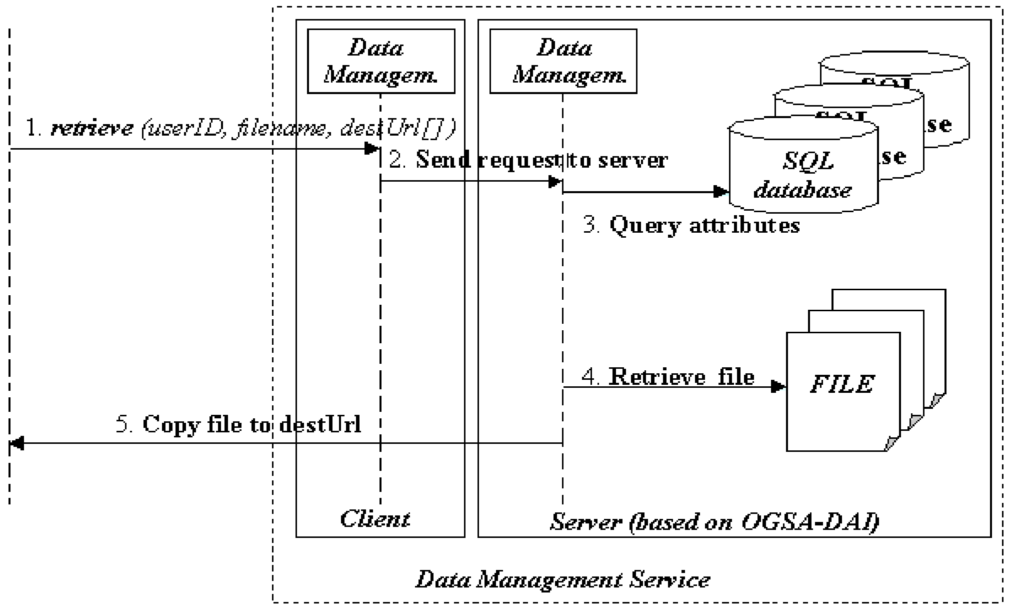

**Figure 11: Retrieving Data sequence diagram**

### *3) Transferring Data*

This functionality allows transferring data from one location to another. In order to be able to copy the file it is necessary to specify the URL of the source file and the URL of destination file. This functionality acts as a third party transfer. It is essential that the user knows exactly the location of the source file. The data file could be transferred using the devoted Web Service interface provided by the DMS.

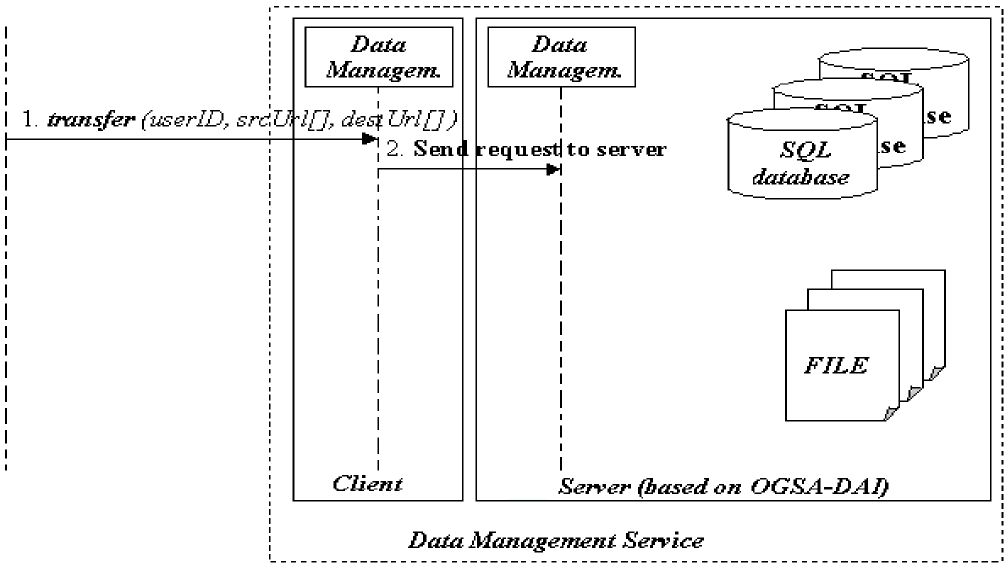
Data Managem.
Data Managem.
1. transfer (userID, srcUrl[], destUrl[])
2. Send request to server
SQL database
FILE
Client
Server (based on OGSA-DAI)
Data Management Service

**Figure 12: Transferring Data sequence diagram**

### *4) Querying Stored Data*

This functionality directly accesses the database where the data files are stored. It is possible to specify an SQL statement. Meta-information about the file and the unique file identifier are returned. The ability to retrieve information depends on the role of the user.

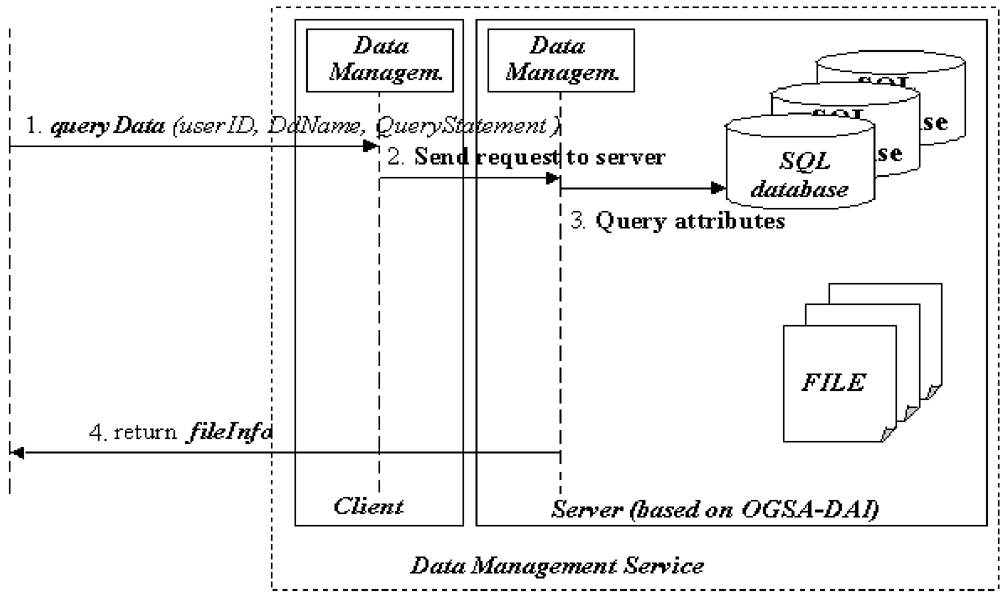

**Figure 13: Querying Stored Data sequence diagram**

### *5) Accessing Stored Data*

This functionality can be used in the case that the file stored is of text type. In such a case, it is possible to directly access the file content without previously retrieving it. This functionality allows access to the specified file and obtaining its content as a string. This ability it is very useful for data file of small dimension; it is possible to directly manipulate the content of the file (without affecting the original one) for any purpose.

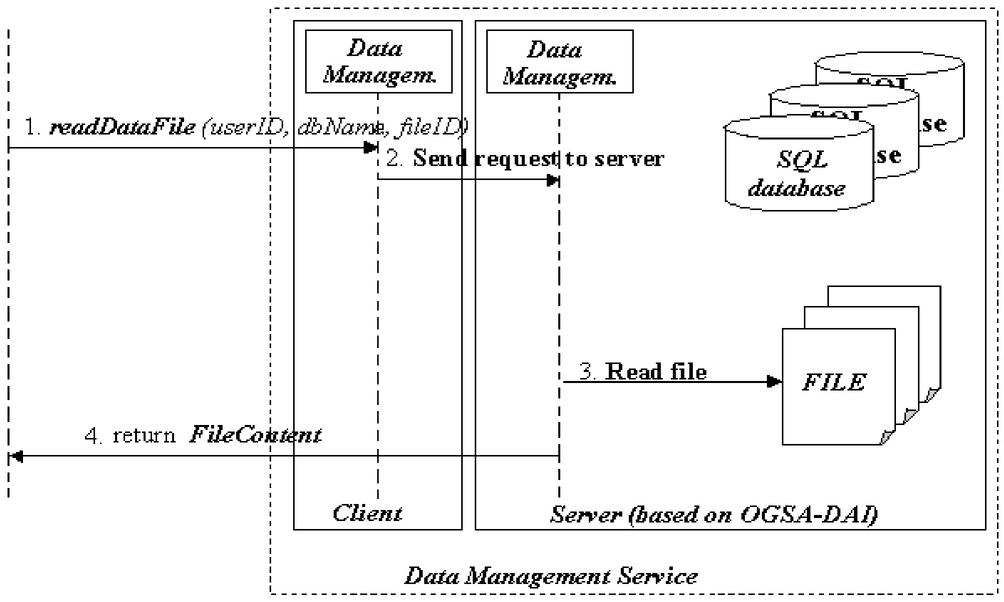

Figure 14: Access Stored Data sequence diagram

### 2.2.1.2. Use Cases

In this paragraph we present three relevant use cases: 1) Upload data files, 2) Retrieve data files, 3) Transfer data files.

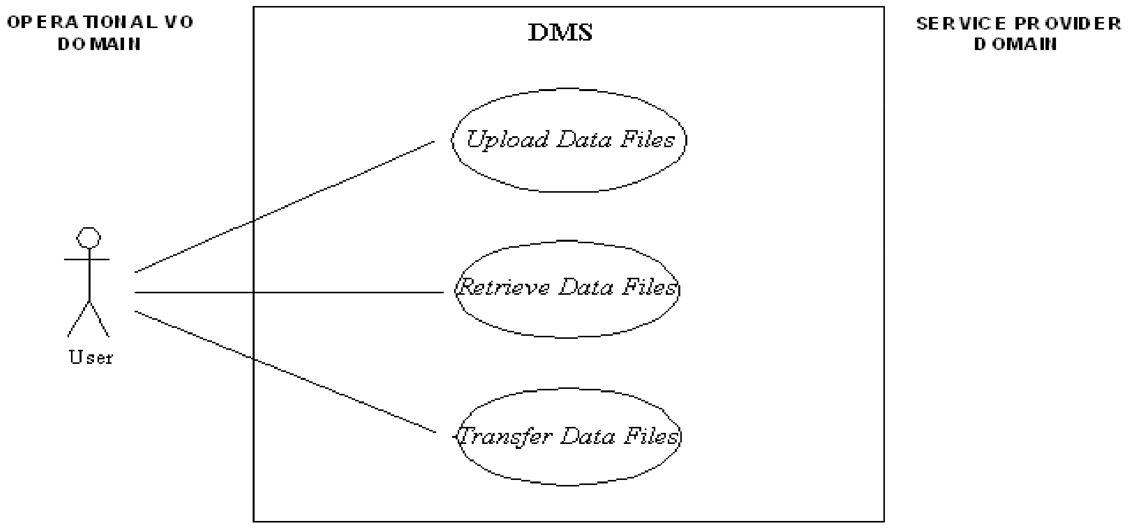

Figure 15: DMS use cases

Table 9: Upload Data use case

| | |
|---|---|
| **Use Case ID** | UC_DM_1 |
| **Use Case Name** | Upload Data |
| **Initiator** | User |
| **Primary Actor** | User |

| | |
|---|---|
| **Additional Actors** | Business Process Designer |
| **Description** | The user invokes the DMS in order to upload and store data. |
| **Pre-condition** | A DMS client should be available in the source location where user's data files reside. |
| **Post-condition** | Data files are uploaded and stored to the DMS. |
| **Use Case Functionality** | |
| **Sequence** | 1. Choose data files to be uploaded<br>2. Choose filename and attributes for each data file to be uploaded<br>3. Upload files |
| **Alternatives** | None |
| **Exceptions** | Upload of data fails because of: 1) user not authorized, 2) wrong input parameters, 3) GridFTP server not running on source machine |
| **Use Cases used** | None |
| **Further Information** | |
| **Particular Requirements** | None |
| **Open Issues** | None |
| **Information Requirements** | None |

**Table 10: Retrieve Data use case**

| | |
|---|---|
| **Use Case ID** | UC_DM_2 |
| **Use Case Name** | Retrieve Data |
| **Initiator** | User |
| **Primary Actor** | User |
| **Additional Actors** | Business Process Designer |
| **Description** | The user invokes the DMS in order to retrieve data previously stored. |

| | |
|---|---|
| **Pre-condition** | A DMS client should be available in the target location where user's data files should be copied. |
| **Post-condition** | Data files are retrieved from the DMS and copied to a local file system. |
| **Use Case Functionality** | |
| **Sequence** | 1. Identify files to be retrieved (filename or unique file identifier should be known)<br>2. Retrieve files |
| **Alternatives** | None |
| **Exceptions** | Retrieve of data fails because of: 1) user not authorized, 2) wrong input parameters, 3) GridFTP server not running on target machine |
| **Use Cases used** | None |
| **Further Information** | |
| **Particular Requirements** | None |
| **Open Issues** | None |
| **Information Requirements** | None |

**Table 11: Transfer Data use case**

| | |
|---|---|
| **Use Case ID** | UC_DM_3 |
| **Use Case Name** | Transfer Data |
| **Initiator** | User |
| **Primary Actor** | User |
| **Additional Actors** | Business Process Designer |
| **Description** | The user invokes the DMS in order to transfer data from one location to another. |

| | |
|---|---|
| **Pre-condition** | A DMS client should be available in the source location where user's data files reside.<br>A DMS client should be available in the target location where user's data files should be copied. |
| **Post-condition** | Data files are copied to a specified location. |
| **Use Case Functionality** | |
| **Sequence** | 1. Choose data files to be copied<br>2. Choose a destination target<br>3. Transfer files |
| **Alternatives** | None |
| **Exceptions** | Upload of data fails because of: 1) user not authorized, 2) wrong input parameters, 3) GridFTP server not running on one or both source/destination location |
| **Use Cases used** | None |
| **Further Information** | |
| **Particular Requirements** | None |
| **Open Issues** | None |
| **Information Requirements** | None |

### 2.2.1.3. *Interactions with other services*

The DMS does not interface with other components of Akogrimo. The DMS has been developed as a stand-alone service that could be directly used as an application service.

**Table 12: DMS interfaces**

| Interfaces | Protocols |
|---|---|
| Upload | SOAP/HTTP |
| Retrieve | SOAP/HTTP |
| Transfer | SOAP/HTTP |
| Cancel | SOAP/HTTP |
| ReadDataFile | SOAP/HTTP |

| | |
|---|---|
| queryData | SOAP/HTTP |

## 2.2.2. DMS Implementation

The first implementation of the DMS was based on the Reliable File Transfer (RFT) component of the GT4 toolkit. This choice has been modified after the first Akogrimo prototype implementation because the RFT was not flexible enough and didn't allow for some useful DMS functionalities to be developed. The OGSA-DAI software makes use of the GT4 toolkit for what concerns the file transfer but uses a different architecture for what concerns the data handling. It offers more possibilities and functionalities. In addition it is more configurable, depending on the needs of the user applications.

For the purposes of the Akogrimo DMS, it has been decided to focus on two key aspects of the Data Management trying to keep the module as simple as possible. These two aspects are: data management in general (upload, store, retrieve, and transfer data) and meta-data (keep information about stored files, search for files and query db). Having in mind the two key aspects of the DMS, it has been decided to configure OGSA-DAI in order to work on two types of resources: file system and SQL database. The file system has been chosen to physically store the data files; the SQL database has been chosen to store meta-data about files.

### *2.2.2.1. Involved Technologies*

The DMS has been developed in Java and it makes use of OGSA-DAI components. It is a Web Service that exposes several interfaces, one interface for each of the functionalities described in paragraph 2.2.1.1. The Web Service is the front end of the DMS that makes use of OGSA-DAI activities. An activity is a component within the OGSA-DAI software which provides a particular piece of functionality. For example, an activity is provided to perform an SQL query. Each data service resource (like file system, database, etc) supports a particular set of activities.

The OGSA-DAI software is compliant with two popular Web Services specifications: WSRF and Web Services Inter-operability (WSI). It has been decided to use the WSRF distribution of OGSA-DAI for two reasons. Firstly, the WSRF flavour is OGSA compliant and secondly, it offers some functionality based on the GT4 software like data movement using the GridFTP protocol. The latter is very important because in a Grid based environment like the one the Akogrimo framework is built upon, the GridFTP protocol is still the most powerful and reliable protocol for data management.

### *2.2.2.2. Configuration and Deployment*

In order to properly setup the DMS some third parties software should be installed and configured:

- OGSA-DAI provided by the University of Edinburgh,
- GridFTP server provided by GT4
- Tomcat provided by Jakarta
- MySQL provided by MySQL AB.

The following steps should be followed:

1. Install OGSA-DAI WSRF distribution and GridFTP under userA.
2. Install Tomcat under userA.
3. Run OGSA-DAI into Tomcat following the given instruction given by the OGSA-DAI documentation.

4. Create an ‘AkoDataService’ data service following the instruction given by the OGSA-DAI documentation.
5. Create two resources and link them with AkoDataService. The two resources are: AkoDataServiceRSql and AkoDataServiceRFile. (The two properties file should be configured).
6. Put the two resources in the configuration directory when deploy the data manager
7. Insert the necessary environment variable into .bashrc of userA
8. Install Tomcat under userB
9. Install Akogrimo DMS under Tomcat of userB
10. userB should be in the same group of userA.
11. Add the following into the .bashrc file:
    a. export DM_PATH=/path/to/DataManagementRepository/conf/
    b. umask g+w
12. Configure the DataManager.conf located in the DM_PATH

### 2.2.3. Security Considerations

As previously stated the DMS is based on OGSA-DAI. In particular, in terms of security, it relies on the use of the GridFTP server powered by the GT4 toolkit. As a consequence the DMS uses the GSI.

At the current implementation stage the GSI is used and configured with just one fake user. This means that there is no correspondence between the Akogrimo users registered in the Participant Registry and the X509 certificate used by the DMS fake user.

In order to integrate the DMS with the Akogrimo security infrastructure we can follow two different approaches. The first option is that the token created by the A4C should be used instead of the X509 certificate to authenticate the users with the GridFTP server. The second possibility is that each token is connected to an X509 certificate. This certificate should be used to authenticate the users against the GridFTP server.

## 2.3. Monitoring Service

### 2.3.1. Monitoring Service Design

The Monitoring component has been designed taking into account scalability and modularity according to the different functionalities it provides. Consequently, the Monitoring component can be seen as a set of four components that provide the following functionalities:

- Remote control concerning enabling/disabling of the monitoring process
- Reception of low level parameters from the Akogrimo producer services
- Sending of QoS object to the SLA enforcement components
- Storage of monitoring information about the different SLA/services being monitored

#### 2.3.1.1. Functionality

The Monitoring service establishes a control over the monitoring process by exposing operations to the EMS that allow the enabling/disabling of the monitoring process for a certain business service (resource) the SLA of which has been previously negotiated. Thus, the monitoring process and functionality are closely affiliated with the SLA enforcement process and functionality. Once, the EMS initiates the execution of a business service, it has to be monitored until its completion to ensure continuous conformation to the terms of the client's SLA contract.

The Monitoring service serves as a link between those components that produce information of parameters (so-called, low level performance parameters) included in the SLA, such as the Metering and the QoS Broker services and the SLA-Controller, which is the service in charge of collecting those parameters, giving them the right format and passing them on to the SLA-Decisor, which is the service responsible for deciding whether a violation has occurred depending on the agreed-upon SLA. From now on, in this section, the Metering and QoS Broker services will be referred to as the "producer services" and the SLA-Controller as the "consumer service".

Internally, the Monitoring service is composed of four different services that work together to carry out the tasks assigned to this service. These sub-services are: (1) the MonManagement service that offers to the EMS a remote control interface to enable/disable operations of the monitoring process, (2) the MonConsumer service, which is the service that receives the information of low level performance parameters from the producers' services, (3) the MonProducer service which is in charge of propagating to the consumer service the values of low level performance parameters and (4) the MonRegistry service, which stores valuable information related to the monitoring process of each service.

Since MonManagement service is a factory service, a resource has to be created by the EMS before invoking any operation on it, as part of the Advance reservation process. The sequence diagram corresponding to the resource creation is depicted in the following figure:

***Resource Creation***

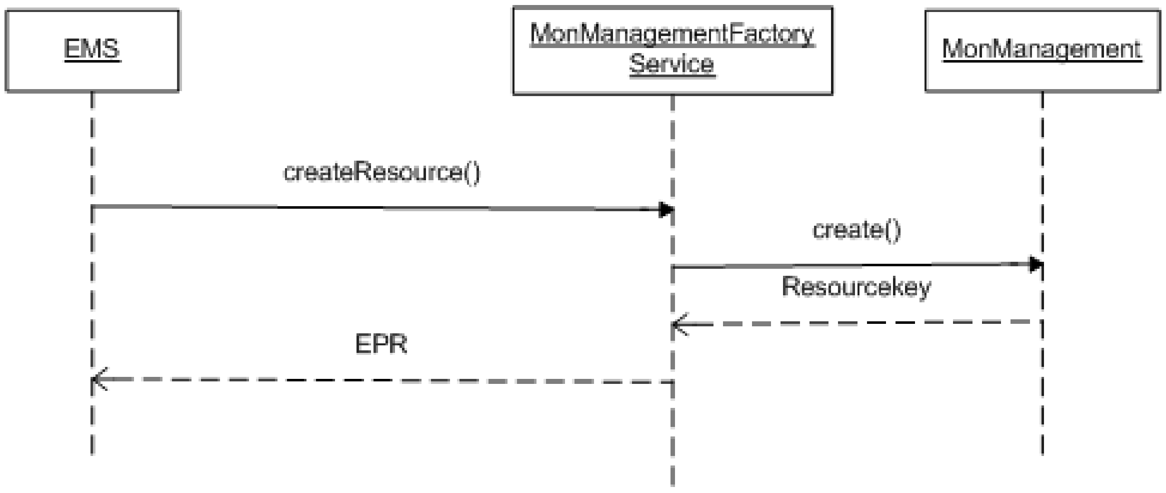

**Figure 16: Resource Creation sequence diagram**

Next, we present the sequence diagrams that correspond to the start and stop operations exposed by the MonManagement service and invoked by the EMS.

***Start Monitoring***

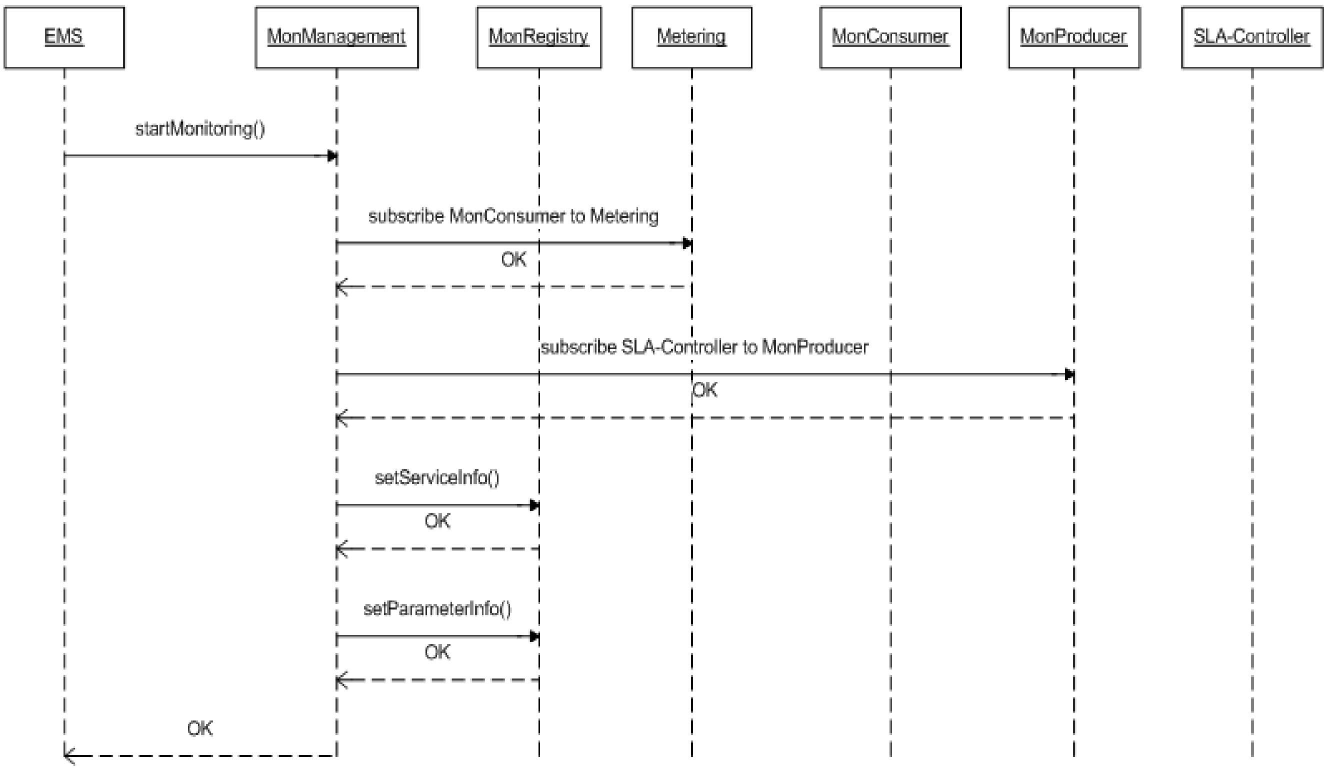

**Figure 17: Start Monitoring sequence diagram**

***Stop Monitoring***

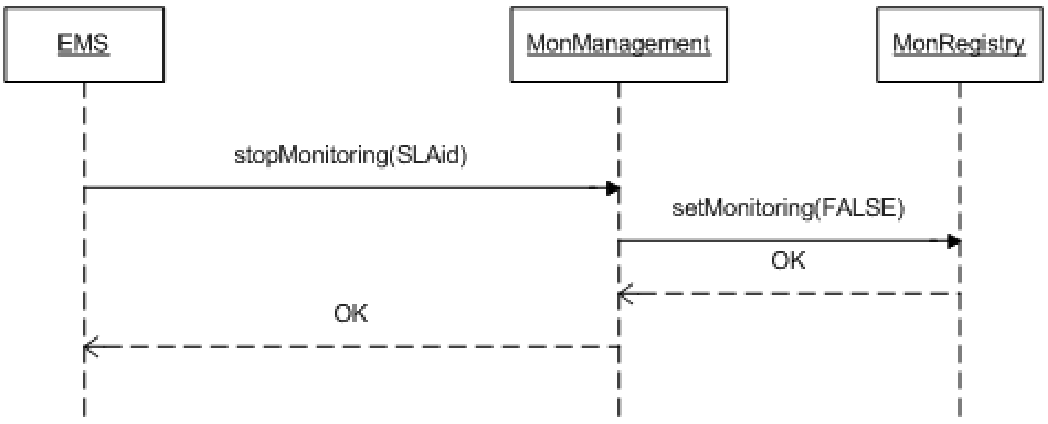

**Figure 18: Stop Monitoring sequence diagram**

The final sequence diagram shows the arrival of a notification message from a producer service (in is example we use the Metering service) to the MonConsumer service and how it is internally handled by the latter[6].

*Notification Handing*

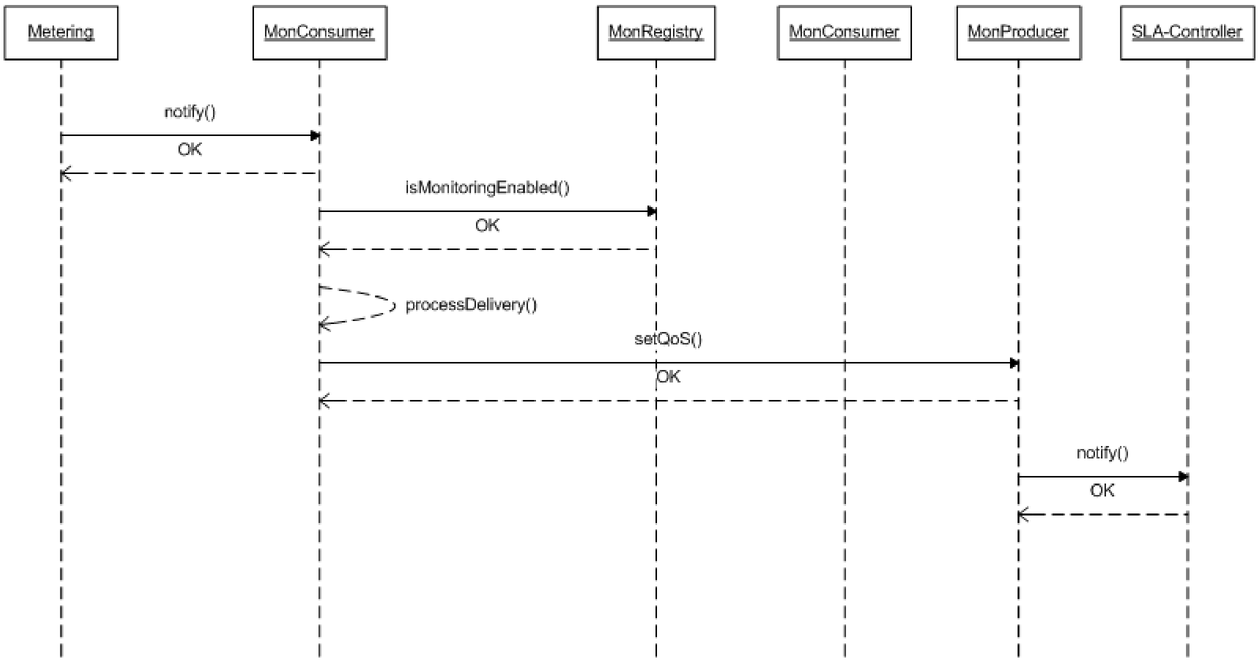

**Figure 19: Notification Handling sequence diagram**

### 2.3.1.2. Use Cases

**Table 13: Monitoring resource creation use case**

| | |
|---|---|
| **Use Case ID** | UC_MON_1 |
| **Use Case Name** | Monitoring resource creation |
| **Initiator** | EMS |
| **Primary Actor** | EMS |
| **Additional Actors** | None |
| **Description** | EMS creates a Monitoring resource per service. |
| **Pre-condition** | The Java WS-Core container where Monitoring component is deployed should be up and running. |

[6] The subscriptions to the Metering and the SLA-Controller service are being carried out during the monitoring enabling process (See Figure 17: Start Monitoring sequence diagram).

| | |
|---|---|
| **Post-condition** | The result of this invocation is the generation of a Monitoring Management resource and the EPR is returned back to EMS. |
| **Use Case Functionality** | |
| **Sequence** | EMS invokes the *create* operation exposed by the MonManagementFactory Service. |
| **Alternatives** | None |
| **Exceptions** | If a fatal internal error occurs that cannot be handled, a *RemoteException* is thrown. |
| **Use Cases used** | None |
| **Further Information** | |
| **Particular Requirements** | None |
| **Open Issues** | None |
| **Information Requirements** | None |

**Table 14: EMS enabling of the monitoring process on the previously created resources Use Case**

| | |
|---|---|
| **Use Case ID** | UC_MON_2 |
| **Use Case Name** | EMS enables the monitoring process on the corresponding resources that have been created on previous step. |
| **Initiator** | EMS |
| **Primary Actor** | EMS |
| **Additional Actors** | Metering service, SLA-Controller service |
| **Description** | EMS invokes the *startMonitoring* operation exposed by the Monitoring resource. |
| **Pre-condition** | Container up and running and the monitoring resources should have been created previously. |
| **Post-condition** | This action implies two subscriptions: MonConsumer to Metering and SLA-Controller to MonProducer. |

| Use Case Functionality | |
|---|---|
| **Sequence** | 1. MonConsumer to Metering subscriptions accomplished.<br>2. SLA-Controller to MonProducer service accomplished<br>3. Storage of monitoring information associated to the service to the Monitoring Registry.<br>4. Enable of the monitoring service. |
| **Alternatives** | None |
| **Exceptions** | None |
| **Use Cases used** | None |
| **Further Information** | |
| **Particular Requirements** | None |
| **Open Issues** | None |
| **Information Requirements** | None |

**Table 15: EMS disabling of the monitoring process on a previously created resource Use Case**

| | |
|---|---|
| **Use Case ID** | UC_MON_3 |
| **Use Case Name** | EMS disables the monitoring process on a resource previously created. |
| **Initiator** | EMS |
| **Primary Actor** | EMS |
| **Additional Actors** | None |
| **Description** | EMS invokes the *stopMonitoring* operation exposed by the Monitoring resource. |
| **Pre-condition** | Container up and running and the monitoring resources should have been created previously. |
| **Post-condition** | This action implies the update in the registry of the status of the monitoring process associated to this resource. |
| **Use Case Functionality** | |

| | |
|---|---|
| **Sequence** | Update of the monitoring process status to FALSE associated to the resource. |
| **Alternatives** | None |
| **Exceptions** | None |
| **Use Cases used** | None |
| **Further Information** | |
| **Particular Requirements** | None |
| **Open Issues** | None |
| **Information Requirements** | None |

**Table 16: Notification sent from Metering to Monitoring Consumer resource Use Case**

| | |
|---|---|
| **Use Case ID** | UC_MON_4 |
| **Use Case Name** | Metering sends a notification to Monitoring Consumer resource. |
| **Initiator** | Metering |
| **Primary Actor** | Monitoring Consumer |
| **Additional Actors** | Monitoring Producer, SLA-Controller |
| **Description** | Once a change on a low level SLA parameter has occurred in a particular host, the Metering service sends a notification to the corresponding Monitoring Consumer resource. |
| **Pre-condition** | Container up and running and the monitoring resources should have been created previously.<br>The subscription process has been taken placed successfully. |
| **Post-condition** | Depending on whether the notification topic is of any interest to the service, Monitoring Consumer modifies the corresponding Monitoring Producer resource property to allow the launch of a notification to the SLA-Controller. |
| **Use Case Functionality** | |

| | |
|---|---|
| **Sequence** | 1. Metering sends a notification to the corresponding Monitoring Consumer resource.<br>2. Monitoring Consumer resource checks if monitoring is enabled and if the topic modified is of any interest to the service.<br>3. In affirmative case, Monitoring Consumer modifies the corresponding Monitoring Producer resource property to allow the launch of a notification to the SLA-Controller. |
| **Alternatives** | None |
| **Exceptions** | None |
| **Use Cases used** | None |
| **Further Information** | |
| **Particular Requirements** | None |
| **Open Issues** | None |
| **Information Requirements** | None |

### 2.3.1.3. Interactions with other services

Next we summarize in a table the interactions that the Monitoring Service has with other Akogrimo components. As previously explained, the functionality of the Monitoring service is targeted at monitoring the fulfilment of SLAs that have been negotiated for each service. Consequently, its interactions are closely related with the monitoring manager process component (EMS), the producer components (Metering and QoS Broker) and the consumer components (SLA-Controller).

**Table 17: Monitoring interactions with other Akogrimo services**

| Akogrimo service | Interaction | Protocol |
|---|---|---|
| EMS | -To create a Monitoring resource and manage its lifecycle<br>-To start/stop the monitoring process | SOAP/HTTP |
| Metering | To receive notifications about the values of the low level performance parameters | SOAP/HTTP |
| SLA-Controller | To send QoS notifications | SOAP/HTTP |
| QoS Broker | To receive notifications about network parameters (to be implemented) | SOAP/HTTP |

## 2.3.2. Monitoring Service Implementation

The implementation of the Monitoring service has been done by splitting up the service into four different services: Monitoring Management, Monitoring Consumer, Monitoring Producer and Monitoring Registry. The first three subsystems represent different Grid Services deployed in a container and the last component is just a file directory, where monitoring information about the different services that are being monitored is stored.

**Monitoring Management**

This service is the management interface that the Monitoring service exposes to EMS in order to allow the enabling (*startMonitoring*) and disabling (*stopMonitoring*) of the monitoring process of a certain service. This service has been implemented following the well-known Factory design pattern which consists in developing a factory service in charge of creating resource instances of the actual service. Thus, the creation of the Monitoring management resource has to be carried out by invoking the Create operation of the MonManagementFactory service. Once, the Monitoring Management resource has been created, the start/stop monitoring can be invoked on this resource

The start operation implies the subscription process of MonConsumer service to Metering (and QoSBroker) and the subscription of SLA-Controller to MonProducer. So, once the start operation is invoked, the monitoring process begins and the Monitoring resource can receive notifications by the Metering service related to the topics that it has subscribed to. The subscription topics are *cpuUtil*, *memoryUtil* and *diskUtil*. When a notification is received by the Monitoring Consumer resource, some previous checks have to be performed: firstly, the monitoring process has to be enabled for this service and secondly, not all topics are of interest to every service. If monitoring is enabled and the topic modified is of interest for the service, then a change of the Monitoring Producer topic (Resource Property) is performed in order to launch the corresponding notification to the SLA-Controller resource. The stop operation simply disables the monitoring process for a particular service.

**Monitoring Consumer**

This subsystem represents the monitoring interface to the producers' services. Producers' services are considered to be services that can provide low level SLA parameters information to the Monitoring consumer. So far, this information is provided only by the Metering service (*cpuUtil*, *memoryUtil* and *diskUtil*) but the QoS Broker service could also provide info regarding network bandwidth.

**Monitoring Producer**

This service represents the monitoring interface to the SLA-Controller service which is the one interested in the low level SLA parameters information in order to check if SLA contract is being violated.

**Monitoring Registry**

This service is a data storage used internally for storing monitoring data. The Monitoring service stores information about the status of the monitoring process for every service (allowed/denied), the topics that each service is interested in and some other internal useful monitoring information. So far, this storage consists of files stored in the directory specified by the property "*service_registry_file*" that appears in *monitoring.properties* file.

#### 2.3.2.1. Involved Technologies

Monitoring component has been implemented using the WSRF implementations provided and included within the GT4.0.1 distribution. The specifications involved are WS-ResourceProperties, WS-ResourceProperties [13] and WS-BaseNotification [15]. All the functionalities of the monitoring component have been built above these specifications. The component can be deployed in any Java WS-Core version 4.X. In order to deploy the component on Tomcat container, some minor changes have to be accomplished.

#### 2.3.2.2. Configuration and Deployment

The monitoring service software is split up in three different sub-services as already explained in previous sections:

1. Monitoring Consumer involves the following software and configuration files: *es_tid_div2115_akogrimo_services_monitoring_monConsumer.gar*, *monitoringConsumer.properties*
2. Monitoring Producer involves the following software: *es_tid_div2115_akogrimo_services_monitoring_monProducer.gar*
3. Monitoring Management involves the following software and configuration files: *es_tid_div2115_akogrimo_services_monitoring_monManagement.gar*, *monitoring.properties*.

The deployment process implies the realization of following steps:

1. Deployment of Monitoring Producer gar:

   *globus-deploy-gar es_tid_div2115_akogrimo_services_monitoring_monProducer.gar*
2. Deployment of Monitoring Consumer gar and the copy of configuration file to $GLOBUS_LOCATION

   *globus-deploy-gar es_tid_div2115_akogrimo_services_monitoring_monConsumer.gar*

   *cp monitoringConsumer.properties $GLOBUS_LOCATION/*
3. Deployment of Monitoring Management gar and the copy of configuration file to $GLOBUS_LOCATION

   *globus-deploy-gar es_tid_div2115_akogrimo_services_monitoring_monManagement.gar*

   *cp monitoring.properties $GLOBUS_LOCATION/*

The configuration process means the adjustments of the parameters values enclosed in the configuration files *monitoring.properties* and *monConsumer.properties*. So, it is needed to check the content of this file in order to adjust the corresponding values. Here, a list of the variables is shown for *monitoring.properties* and *monConsumer.properties*:

- *service_registry_file*: The name (with full path) of the file where the main features of the services to be monitored are stored. For instance a "true" value at the end of the record (line) means that monitoring is enabled.
- *service_file_path*: The directory name where all the registry files are going to be stored.
- *Separator*: The combination of keys that are used to separate the features associated to a service. Each service is stored in a single line and each property is separated by the "separator" keys combination.
- *MonProducerURI* and *MonConsumerURI*: The URL which the Monitoring producer and consumer are deployed at.

It is important to create first the directories where the service information is going to be stored (Monitoring Registry).

### 2.3.3. Security Considerations

The security implemented in this service is linked to the GT4 security since the Monitoring service has been designed to be a WSRF-compliant Grid Service deployed in GT4 platform. So far, a full installation of the GT4 has not been carried out in all machines in the prototype platform so the security is reduced to those aspects that can be included with no use of the Grid Security Infrastructure provided by GT4.

The Monitoring service interacts only with the EMS. The latter, is the only service which can invoke the start/stop monitoring operations. No other service knows the corresponding EPR of the Factory service to create a monitoring resource and consequently to use it on behalf of it.

## 2.4. Service Level Agreement Enforcement Services (SLA-Enforcement Services)

### 2.4.1. SLA-Enforcement Services Design

Two modules have been identified in the design of SLA Management architecture as far as WP4.3 is concerned: the SLA-Controller and SLA-Decisor service.

#### *2.4.1.1. Functionality*

**SLA-Controller**

This service is in charge of guaranteeing that all the agreements included in a SLA contract are respected. It receives and processes all measurements related to the execution of a business service sent by the Monitoring service and therefore it should be deployed in the same target machine as the Monitoring service to reduce communication overhead. It checks whether the measurements are within the thresholds defined in the SLA contract for QoS metrics. Whenever the execution of a business service does not satisfy these conditions, the SLA-Controller notifies (using a WS-Notification mechanism) the SLA-Decisor service, which is in charge of starting the appropriate recovery action.

**SLA-Decisor**

This service receives and manages notifications from the SLA-Controller service. In turn, it contacts the Policy Manager service in order to retrieve the most suitable affiliated policy, depending on the business execution context. This policy includes the actions that must be undertaken, such as showing informational messages, increasing priorities of processes, applying discounts, destroying the service, re-instantiating the service, etc. Depending on the seriousness of the violation, the context of the business service and the overall status of the system, this event can be solved at domain level or at VO level. In the first case, it may be only necessary to notify the EMS which will take corrective actions, whereas in the second case, it may be necessary to notify higher layers (e.g., workflow manager or other subsystem) to recover from the situation by taking global corrective actions.

The next figure depicts the interactions among SLA Enforcement services and other subsystems. There are two distinct phases: the Activation phase and the Service Use phase. Detailed use cases are provided below.

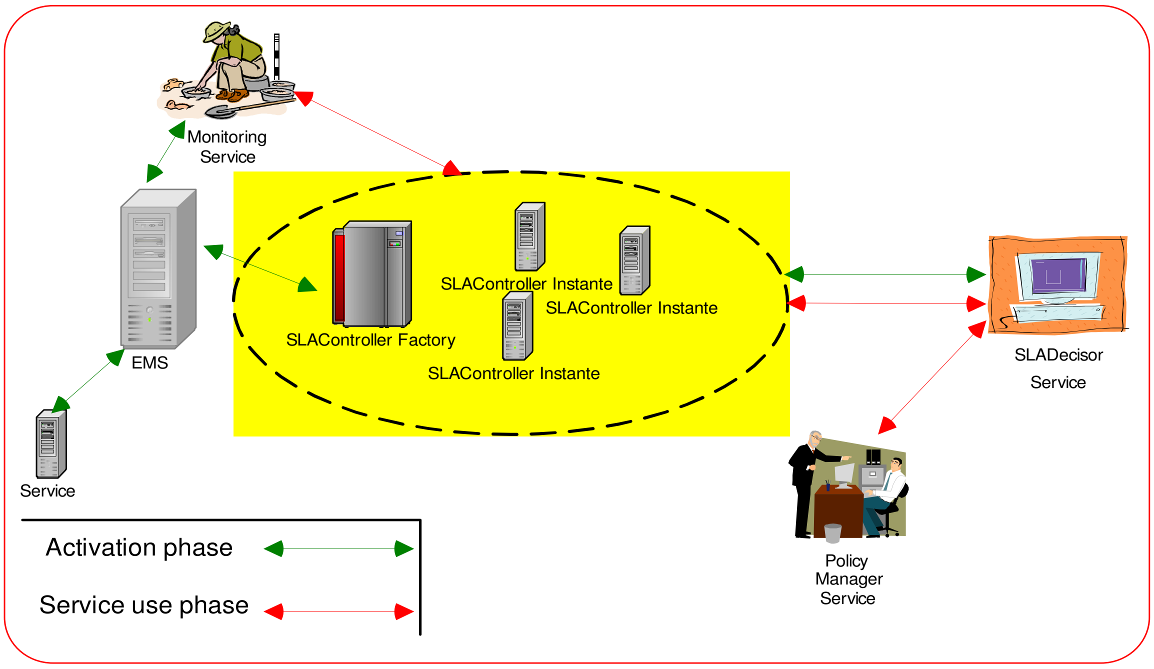

**Figure 20: SLA interfaces**

The following figures show the sequence diagrams which describe the functionality of the SLA-Enforcement group of service.

During the Activation phase that is part of the EMS' Advance Reservation phase, the EMS must create an agent to allow the status control over the business service execution. Afterwards, the SLA-Controller instance is registered to the SLA-Decisor service, so that the latter will be able to receive notifications from the SLA-Controller instance.

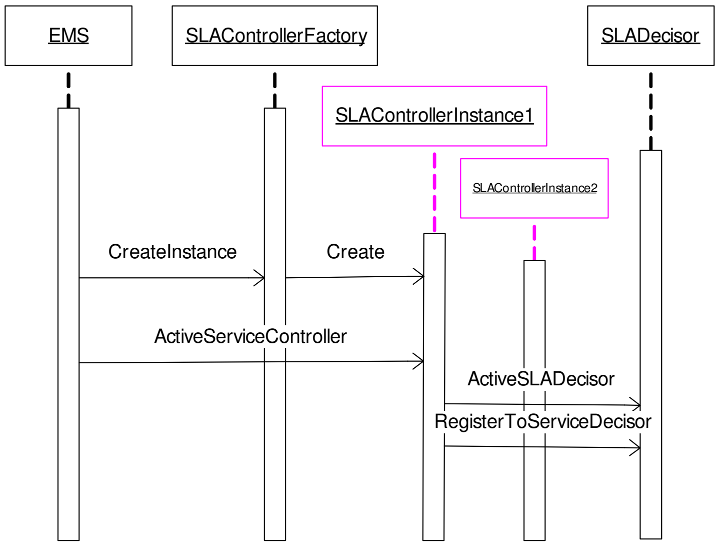

**Figure 21: Sequence diagram for the Activation phase**

During the Service Use phase (i.e. the EMS' Execution phase), the Monitoring service sends measurements to the SLA-Controller service. The latter decides whether the service is running under the expected conditions (i.e. the values of the QoS parameters are within the specified thresholds). If a violation is detected, the SLA-Decisor service interacts with the Policy Manager service, in order to find out what appropriate recovery actions should be taken.

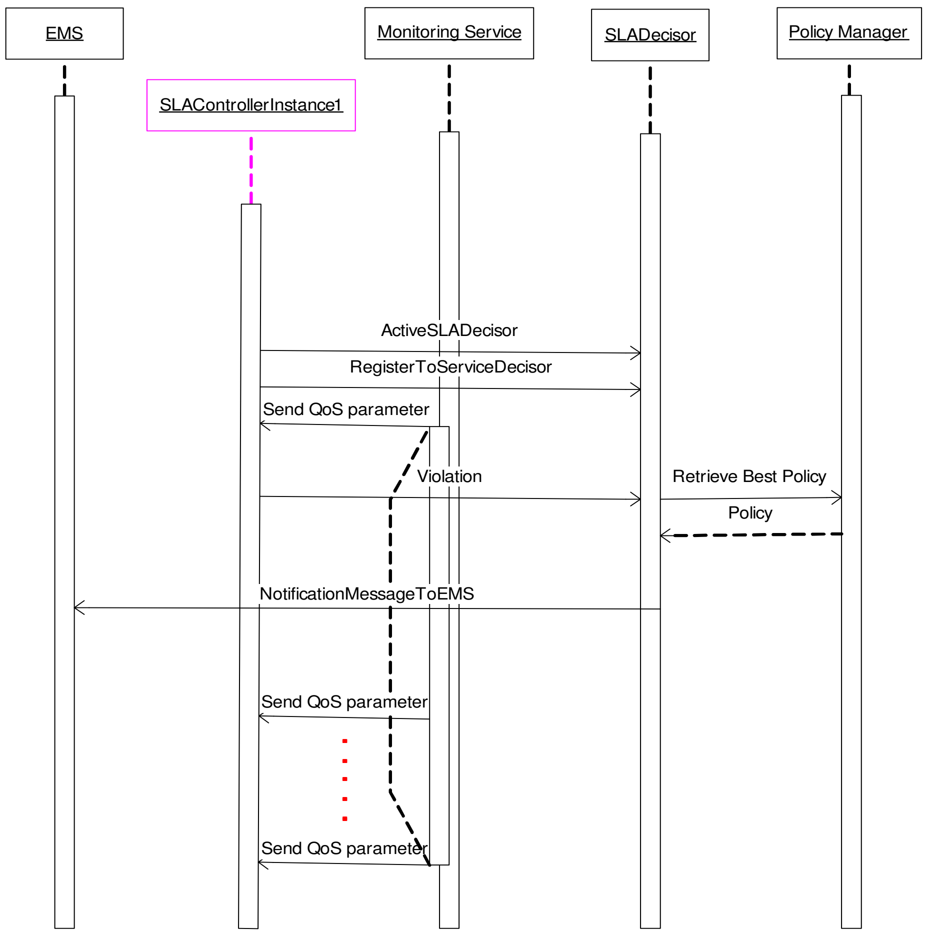

**Figure 22: Sequence diagram for the Service Use phase**

### 2.4.1.2. Use Cases

**Table 18: SLA creation and execution Use Case**

| | |
|---|---|
| **Use Case ID** | UC_SLA-Enforcement_1 |
| **Use Case Name** | SLA-Controller resource creation and execution |
| **Initiator** | EMS |
| **Primary Actor** | SLA-Controller |
| **Additional Actors** | Monitoring Service, SLA-Decisor |
| **Description** | During the Activation phase, the EMS creates an agent (SLA-Controller resource) to allow control over the execution status of the service in speak. Upon creation, the SLA-Controller resource subscribes to SLA-Decisor. The Monitoring system sends QoS measurements related to business service execution to the SLA-Controller resource. |

| | |
|---|---|
| **Pre-condition** | The EMS knows the URI of the SLA-Controller service.<br>The EMS is able to retrieve (by interacting with the SLA-Access) the applied QoS bundles |
| **Post-condition** | An SLA-Controller instance has been created and is able to receive notification message containing QoS values from the Monitoring service. |
| **Use Case Functionality** | |
| **Sequence** | 1. The EMS creates a new SLA-Controller instance.<br>2. SLA-Controller creates a subscription mechanism to the SLA-Decisor.<br>3. The Monitoring service creates a notification mechanism with the SLA-Controller.<br>4. The Monitoring Service sends notifications containing QoS measurements to the SLA-Controller resource. |
| **Alternatives** | The SLA-Controller service can interact directly with the Monitoring service to get the QoS measurements of the service.<br>Flow of events:<br>• SLA-Controller asks for the current status of the service<br>• The Monitoring service reads the current QoS<br>• The Monitoring service sends these values to the SLA-Controller |
| **Exceptions** | None |
| **Use Cases used** | None |
| **Further Information** | |
| **Particular Requirements** | None |
| **Open Issues** | None |
| **Information Requirements** | None |

**Table 19: Violation Detection and Best Policy Application Use Case**

| | |
|---|---|
| **Use Case ID** | UC_SLA-Enforcement_2 |
| **Use Case Name** | Violation detection & Best Policy application |
| **Initiator** | SLA-Controller |
| **Primary Actor** | SLA-Controller |
| **Additional Actors** | SLA-Controller, SLA-Decisor, Policy Manager, EMS, Monitoring Service |
| **Description** | The SLA-Controller evaluates the received QoS measurements and applies the known filters. If an SLA violation is detected, the SLA-Decisor is notified. The latter communicates with the Policy Manager in order to be informed about which policy should be applied.<br>SLA-Decisor informs EMS or other components about which corrective actions should be taken (e.g., abort the process, move the service to another machine, increase the process priority, etc) |
| **Pre-condition** | • An SLA-Controller resource is correctly created<br>• A notification mechanism with the Monitoring service is established<br>• The SLA-Controller resource receives notification messages from the Monitoring (QoS performance parameters values) |
| **Post-condition** | The SLA-Decisor has interpreted the policy and has informed the corresponding service about the action to be taken (e.g., it sends a notification message to the EMS including the recovery action to be taken). |
| **Use Case Functionality** | |
| **Sequence** | 1. The SLA-Controller filters the received QoS values.<br>2. When the SLA-Controller detects a violation, it sends a notification message to the SLA-Decisor<br>3. The SLA-Decisor contacts the Policy Manager.<br>4. The Policy Manager searches for the best policy to be applied.<br>5. The SLA-Decisor sends a notification message to the EMS |
| **Alternatives** | None |
| **Exceptions** | None |
| **Use Cases used** | None |
| **Further Information** | |

| | |
|---|---|
| **Particular Requirements** | None |
| **Open Issues** | None |
| **Information Requirements** | None |

#### 2.4.1.3. Interactions with other services

The table below shows all the interactions between the SLA-Enforcement group of services and other Akogrimo services as well as the interactions amongst the SLA-Enforcement sub-services. See Annex B for more details.

**Table 20: SLA-Enforcement interactions with other Akogrimo services**

| Akogrimo service | Interaction | Protocol |
|---|---|---|
| **SLA - Controller** | | |
| EMS | To create a new SLA-Controller resource | SOAP/HTTP |
| SLA-Decisor | -To establish a notification mechanism for posterior communications between these two modules<br>-To receive notifications from SLA-Controller when a violation is detected. | SOAP/HTTP |
| Monitoring | -To establish a notification mechanism for posterior communications between these two modules<br>-To notify the SLA-Controller about the value of the QoS parameters related to the service | SOAP/HTTP |
| **SLA-Decisor** | | |
| Policy Manager | To ask for the best policy to be applied in case of violation | SOAP/HTTP |
| SLA-Controller | -To establish a notification mechanism for posterior communications between these two modules<br>-To inform SLA-Decisor about a violation | SOAP/HTTP |

| | | |
|---|---|---|
| EMS | To notify about the action to be taken when a violation is detected during the business service execution | SOAP/HTTP |

## 2.4.2. SLA-Enforcement Services Implementation

### *2.4.2.1. Involved Technologies*

The SLA implementation can be based on two specifications: WS-Agreement [25] and WSLA [26]. There are not full implementations of SLA standard specifications but the possible alternatives that can be considered will be shown.

On one hand, WS-Agreement implementation can be found in the Cremona framework included in IBM Emerging Technologies Toolkit (ETTK) [28]. Cremona (Creation, Monitoring and Management of WS-Agreement) allows managing the relationships between a service provider and a service consumer and also provides implementations for creating agreement templates.

On the other hand, there is an implementation of the WSLA framework called "SLA Compliance Monitor" which is part of the IBM Web Services Toolkit [27], that IBM has developed especially for Web Services. The WSLA framework consists of a flexible language based on a XML schema that specifies metrics, penalties, SLA parameters and a way of measuring them.

The WS-Agreement specification document is still in draft format. It defines the structure of agreements and their templates, but it could be extended and complemented by other terms. In the Akogrimo project, the mix of both technologies (WS-Agreement and WSLA) could be considered to obtain the best template, since it is difficult to find implementations that support these specifications completely.

In terms of SLA Contract definition, the WS-Agreement specification is considered. It defines the structure of agreements and their templates and it can be extended and complemented by other terms. However, the analysis of approaches gathered in WSLA (IBM specification) has been taken into account to define the contract template.

Finally, the SLA-Controller and SLA-Decisor are developed in Microsoft .NET platform and therefore they use the .NET framework 1.1 with WS Enhancements (WSE 2.0 SP3). The SLA-Controller is a ***transient Grid Service***, while the SLA-Decisor is a ***persistent Grid Service***. They make use of WSRF.NET implementation of the University of Virginia (V. 2.1.0). The communication between these two services is based on the WS-Notification specification.

In terms of SLA Contract definition, the WS-Agreement specification is considered. It defines the structure of agreements and their templates and it can be extended and complemented by other terms.

However, the analysis of approaches gathered in WSLA (IBM specification) has been taken into account to define the contract template.

### *2.4.2.2. Configuration and Deployment*

There can be several configurations for deploying SLA Enforcement sub-services. This will depend on the strategy to configure the Akogrimo platform. In this section we present a standard approach with the mentioned version of having part of SLA functionalities (mainly the SLA-Decisor) in a trusted third party.

In the reference configuration, we foresee to deploy the SLA-Controller within the Akogrimo environment on each machine containing the Monitoring module, whereas the SLA-Decisor will be centralized in one host per SP domain.

- The SLA-Controller will be deployed on every machine the Monitoring subsystem will be deployed to so that it receives all measurements provided by each specific machine which is hosting the service. The SLA-Controller cannot be centralized due to the fact that for each service one of these components will be associated to it. Having one SLA-Controller in the same machine with the Monitoring Service will allow avoiding unnecessary network communications. The permanent interaction between these two components makes it necessary to deploy them together. After the analysis of the measurements received, the SLA-Controller will communicate with the SLA-Decisor only if/when it decides is necessary.
- The SLA-Decisor will be deployed once for SP domain, in the same way as EMS will do. This component will receive all notifications produced by active SLA-Controllers and after choosing the right policies it will decide what the EMS should do. It is not necessary to have one SLA-Decisor component for each machine because it will only receive direct communications by the SLA-Controller module, and then it must contact the EMS subsystem or other modules. As it was pointed out, this module can be deployed in a trusted third party of VO that may store the SLA contract, in order to guarantee the independence of verifications and the generation of violations.

### 2.4.3. Security Considerations

The SLA Enforcement subsystem doesn't have (at least in this phase) particular strict security requirements. The messages exchanged among the services (internal to the subsystem and external) are related to "administration issues". However, an X509-based system could be used in order to protect the communication between the services, whereas the facility provided by WSE in order to create a "secure communication" could be also used.

## 2.5. Metering Service

### 2.5.1. Metering Service Design

The Metering service is positioned in the Grid middleware layer. Through the measurement of the resources that are being consumed, the Metering service is able to support the monitoring and the accounting within the Akogrimo Infrastructure by interacting with the Monitoring service and the A4C system respectively.

#### *2.5.1.1. Functionality*

The functionality of the Metering service can be divided into the following categories:

- *Monitoring*: The Metering service notifies the Monitoring service about the changes in the values of the low level performance parameters related to the execution of a service.
- *Accounting*: An aggregated version of the information that is calculated by the Metering service is communicated to the A4C system of WP4.2 and used for accounting purposes at the end of the service execution. The following figure shows a vertical approach of the interaction between the Metering service and the A4C system.

The low level performance parameters that are measured are listed below:

- CPU usage

- Memory usage
- Disk usage
- Wall clock time (time that elapsed while the business service was running)

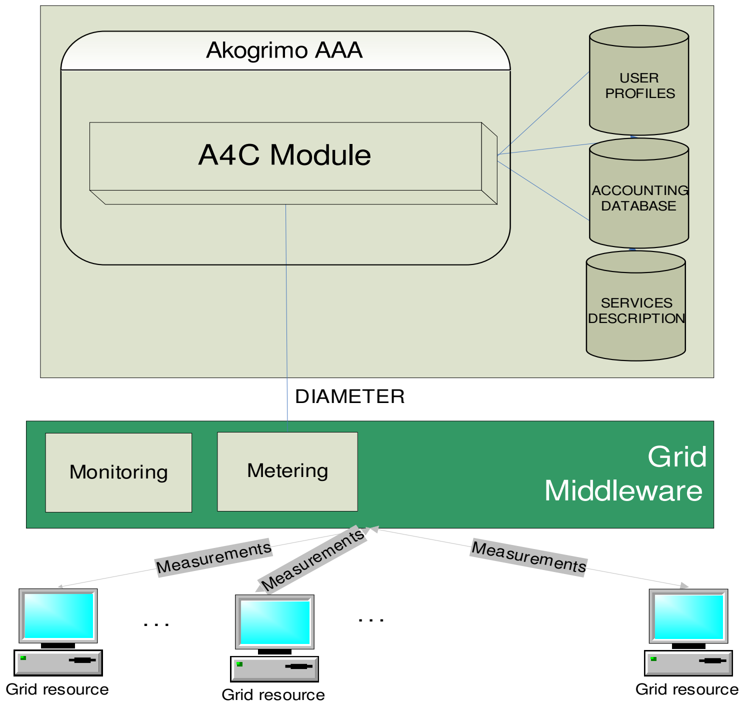

**Figure 23: Metering component and A4C in Akogrimo**

Figure 24: Sequence diagram of the Metering Service execution, shows the sequence diagram of the interactions performed by the Metering service

**Advance reservation phase**

1. EMS provides the IDs of the service, client and SLA and submits request for the creation of a Metering resource.
2. The Metering service creates a Metering WS-Resource.
3. The termination time of the Metering resource is set to the time the SLA contract expires.

4-6. Upon creation, the Monitoring resource subscribes to the notification mechanism of the Metering resource.

**Execution phase**

1. The sequence begins with the EMS asking the Metering resource to start metering the consumption of resources related to a business service execution, providing the ID of the client's SLA contract.
2. Once started, the Metering resource retrieves information related to the execution of the business service periodically.
3. The information retrieved in previous step is used as input to algorithms that calculate low level performance parameters related to the execution of the business service (CPU, memory, etc).
4. If the values of low level performance parameters have changed since the previous calculation, the Metering service notifies the Monitoring service about this change and includes the values of the parameters that have changed in the notification message. In order

to avoid SOAP message traffic, the Metering Service notifies the Monitoring Service only if a significant change in the values of the performance parameters has taken place.

5. After the end of the business service execution, the EMS stops the Metering service.
6. The Metering service aggregates the values of the low level performance parameters that have been calculated during the execution of the service and communicates this information to the A4C system in the form of accounting records.

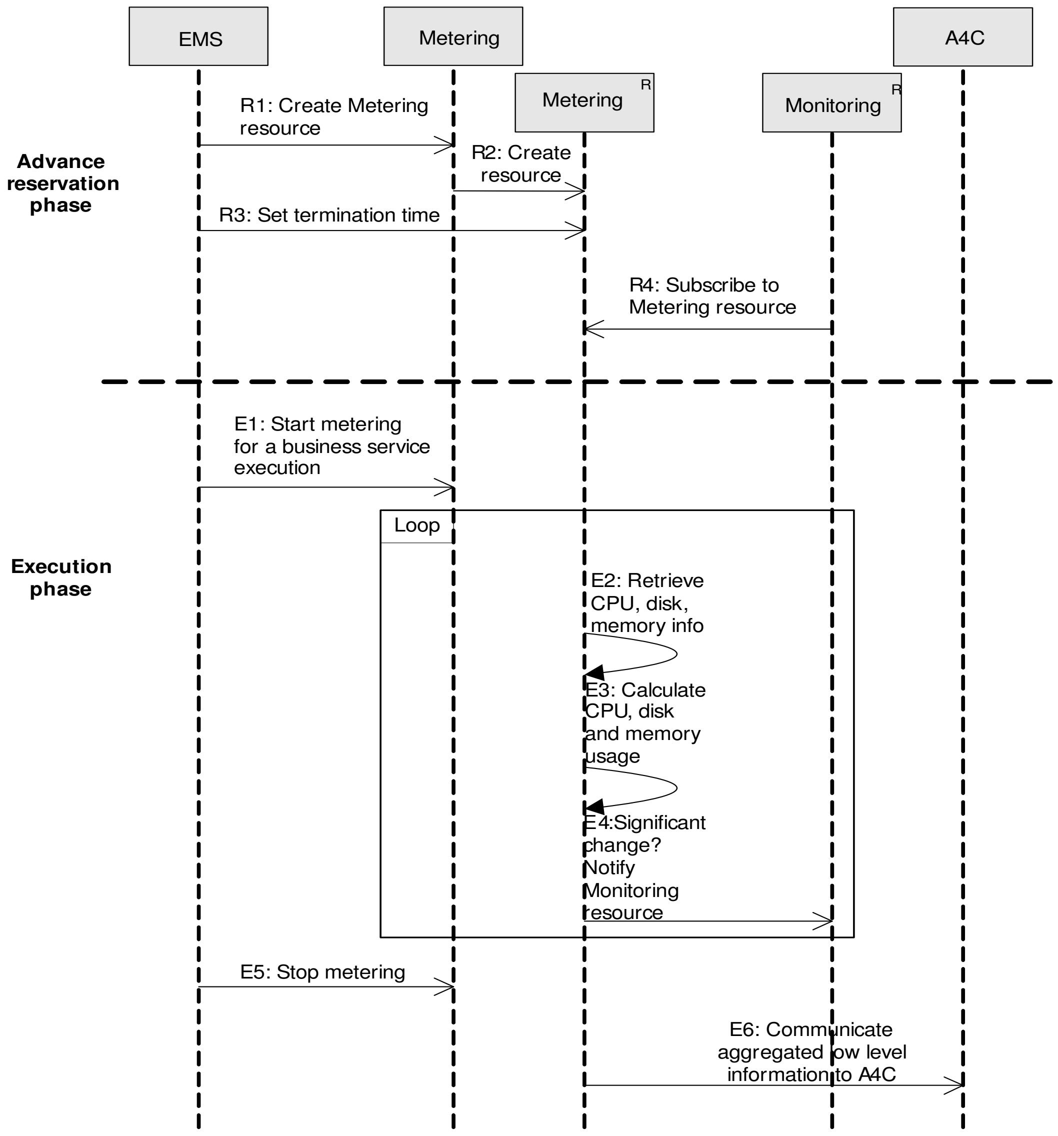
EMS
Metering
A4C
Metering R
Monitoring R
Advance reservation phase
R1: Create Metering resource
R2: Create resource
R3: Set termination time
R4: Subscribe to Metering resource
Execution phase
E1: Start metering for a business service execution
Loop
E2: Retrieve CPU, disk, memory info
E3: Calculate CPU, disk and memory usage
E4:Significant change? Notify Monitoring resource
E5: Stop metering
E6: Communicate aggregated low level information to A4C

**Figure 24: Sequence diagram of the Metering Service execution**

The high-level view of the interactions shown in the above figure in conjunction with Table 21: Services involved in the execution of the Metering service provides a full description of the design details and behavior of the Metering service. For more information refer to section B.5 of the Annex.

**Table 21: Services involved in the execution of the Metering service**

| Service | Interaction | Interface |
|---|---|---|
| EMS | Step R0 | *EndpointReferenceType createMeteringResource(CreateMeteringResource params)* |
| | Step R3 | *org.oasis.wsrf.lifetime.SetTerminationTimeResponse setTerminationTime(org.oasis.wsrf.lifetime.SetTerminationTime setTerminationTimeRequest)* |
| | Step E1 | *boolean startMetering(String slaID)* |
| | Step E5 | *boolean stopMetering(StopMetering params)* |
| Monitoring | Step R4 | *org.oasis.wsn.SubscribeResponse subscribe(org.oasis.wsn.Subscribe subscribeRequest* |
| A4C | Step R6 | *A4C Java Client API* |

### 2.5.1.2. Use Cases

We consider three use cases for the Metering service: (1) creation and lifecycle management of a Metering resource (performed by the EMS), (2) collection of metering information (performed by the Monitoring service) and (3) collection of accounting information (performed by the A4C system):

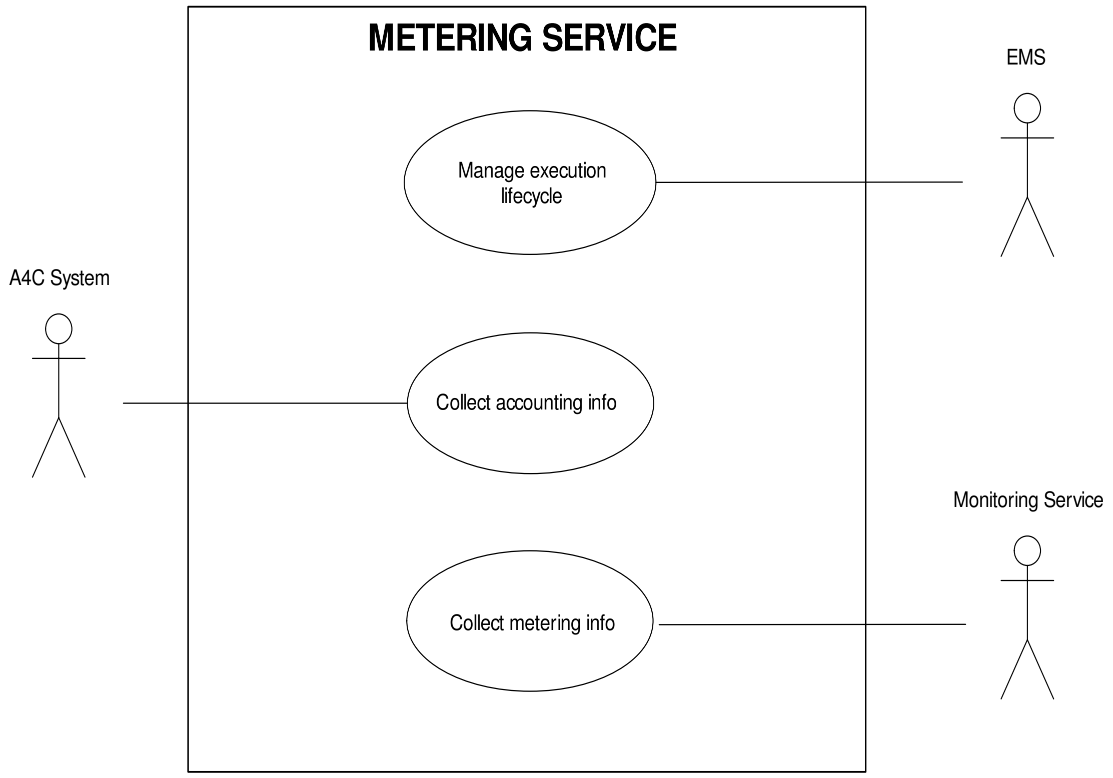

**Figure 25: Metering Service use cases**

**Table 22: Manage of lifecycle Use Case**

| | |
|---|---|
| **Use Case ID** | UC_METER_1 |
| **Use Case Name** | Management of lifecycle |
| **Initiator** | EMS |
| **Primary Actor** | Metering service |
| **Additional Actors** | None |
| **Description** | EMS creates a Metering WS-Resource and manages its execution lifecycle. |
| **Pre-condition** | The Metering service is functional only on machines with Linux OS. The Metering service must be up and running on the machine where the execution of the business service will take place. |
| **Post-condition** | A Metering WS-Resource is created and its and its EPR returned to the EMS. This resource is scheduled to self-destruct when the affiliated SLA expires. |
| **Use Case Functionality** | |
| **Sequence** | 1. EMS invokes the *createMeteringResource* operation on the Metering service in the machine where the execution will take place<br>2. A Metering WS-resource is created and inside it all information is stored. The EPR of this resource is returned to the EMS<br>3. EMS invokes the *setTerminationTime* operation on the resource. If successful, the Metering resource is programmed to self-destruct when the SLA contract becomes invalid. |
| **Alternatives** | None |
| **Exceptions** | In case an error occurs that Metering service cannot handle, *RemoteException* is thrown. |
| **Use Cases used** | None |
| **Further Information** | |
| **Particular Requirements** | None |
| **Open Issues** | None |
| **Information Requirements** | None |

**Table 23: Collect Metering Info Use Case**

| | |
|---|---|
| **Use Case ID** | UC_METER_2 |
| **Use Case Name** | Collect metering info |
| **Initiator** | EMS |
| **Primary Actor** | Metering service |
| **Additional Actors** | Monitoring service |
| **Description** | EMS activates the Metering WS-Resource |
| **Pre-condition** | The Metering WS-Resource must be created at the reservation phase. The Metering service must be up and running on the machine where the execution of the business service will take place. |
| **Post-condition** | The values of low level performance parameters are being calculated periodically and notifications are sent to the Monitoring service when there is a significant change. |
| **Use Case Functionality** | |
| **Sequence** | 1. EMS activates a Metering WS-Resource<br>2. The Metering WS-Resource calculates the values of low level parameters periodically and sends out notifications to the Monitoring service. |
| **Alternatives** | None |
| **Exceptions** | In case an error occurs that Metering service cannot handle, *RemoteException* is thrown. |
| **Use Cases used** | None |
| **Further Information** | |
| **Particular Requirements** | None |
| **Open Issues** | None |
| **Information Requirements** | None |

**Table 24: Collect Accounting Info Use Case**

| | |
|---|---|
| **Use Case ID** | UC_METER_3 |
| **Use Case Name** | Collect accounting info |
| **Initiator** | EMS |
| **Primary Actor** | Metering service |
| **Additional Actors** | A4C |
| **Description** | The EMS deactivates the Metering resource. |
| **Pre-condition** | The Metering service is functional only on machines with Linux OS. The Metering service must be up and running on the machine where the execution of the business service will take place. |
| **Post-condition** | The Metering resource aggregates the values of the low level performance parameters that have been calculated during the execution of the service and communicates this information to the A4C system in the form of accounting records |
| **Use Case Functionality** | |
| **Sequence** | 1. EMS activates a Metering WS-Resource<br>2. The Metering WS-Resource calculates the Values of low level parameters periodically and sends out notifications to the Monitoring service. |
| **Alternatives** | None |
| **Exceptions** | In case an error occurs that Metering service cannot handle, *RemoteException* is thrown. |
| **Use Cases used** | None |
| **Further Information** | |
| **Particular Requirements** | None |
| **Open Issues** | None |
| **Information Requirements** | None |

### 2.5.1.3. *Interactions with other services*

The following table summarizes the interactions that Metering has with other Akogrimo services. The services involved, the communication protocol that is being used along with a short description of the interactions are included.

**Table 25: Metering Service interactions with other Akogrimo services**

| Akogrimo services | Interaction | Protocols |
|---|---|---|
| EMS | -To create a Metering WS-Resource<br><br>-To manage its lifecycle<br><br>-To activate the Metering WS-Resource<br><br>-To deactivates the Metering WS-Resource | SOAP/HTTP |
| Monitoring | Monitoring subscribes to a Metering resource in order to receive notifications | SOAP/HTTP |
| A4C | To send accounting information to the A4C system | DIAMETER |

## 2.5.2. Metering Service Implementation

The Metering service was developed on the GT4 platform and makes use of the WS-Resource Factory pattern and the WS-Notification family of specifications. In particular, it communicates with the Monitoring service by using a notification mechanism established between the two services. The notification mechanism is developed on the basis of the WS-BaseNotification [15], a specification that belongs to the WS-Notication family of specifications that enables the use of the notification design pattern with Grid Services. More formally, the Metering service has the role of the notification producer, while the Monitoring service acts as the notification consumer. The Metering service publishes a set of topics that the Monitoring service can subscribe to, and receive a notification whenever the value of the topic changes. The subscription to the topics is performed during the initiation of the execution of the Monitoring service by the EMS. The topics that the Metering service publishes correspond to the low level performance parameters that are measured during its execution. Each time the value of a low level performance parameter changes significantly[7], a notification message is automatically generated by the Metering service and sent to the Monitoring service. The notification message is an XML-like string that includes all the necessary information about the performance parameter and the service that is affiliated with.

[7] This threshold has a different value for each of the parameters that are being measured. For disk usage: |oldValue – newValue|>20%, for memory usage: |oldValue-newValue|>15% and for CPU usage: |oldValue-newValue|>10%. These thresholds have been chosen as the most appropriate within the Akogrimo Framework, with respect to the business services. These values could be easily changed in order to adjust to any new requirments.

#### 2.5.2.1. Involved Technologies

The technologies involved in the implementation of the Metering service are listed below:

- WSRF and WS-Notification specifications.
- Java WS-Core 4.0.1 included in the GT4 toolkit.
- Java JDK 1.5.0

#### 2.5.2.2. Configuration and Deployment

The Metering component should ideally be deployed in every machine that is hosting a business service. A full installation of the GT4 is not required on those machines as the Metering service is fully functional by installing just the Java WS Core component. The current version of the Metering service has been tested and is fully functional only on Linux platforms.

### 2.5.3. Security Considerations

All the communications between the Metering service and the services that it interacts with must be secure. This means that the services must be mutually authenticated. In order to secure the communication between the Metering service and other GT4 services such as the EMS and the Monitoring service, we shall take advantage of the security infrastructure that the GT4 provides. Securing the communication between the Metering service and the A4C system needs a different approach, since the services are build on different platforms and use different protocols for communication.

## 2.6. Policy Manager Service

### 2.6.1. Policy Manager Service Design

The Policy Manager exists as a distributed service, within the Akogrimo distributed computing environment. The role of the Policy Manager is to coordinate the activities in the Akogrimo environment. The Akogrimo policies originate from two main areas of influence. We distinguish between the global policies that exist in the VO domain and the local policies that exist within the SP domain. Policies in both of these two domains although they are specific to the rules of the individual domain, have to function in a manner to ensure interoperability between services from both domains.

In order to achieve interoperability of services in terms of policy in different domains, the Policy Manager was positioned inside the VO domain of the Akogrimo framework. The key aim of this service is to ensure there is no conflict between policies of services using the VO. Within this policy hierarchy the policy rules in the VO domain take priority over the rules in the SP domain. Policy requirements have the form of a hierarchical tree. At each node of the tree, there is an instance of the Policy Manager. The top level node, at the root of the tree, is the VO Policy Manager whereas the nodes at the lower level of the tree are the SP Policy Managers. In order to achieve this hierarchy, the SP Policy Managers are configured to refer to the VO Policy Manager.

When a Policy Manager instance is asked for a policy, it forwards the request to the upper policy node to retrieve the global policies and once this is complete then it performs a search in its local database for matching policies to apply. When all policies (local and global) are retrieved, they are merged and the resulting policy is returned to the requestor.

#### 2.6.1.1. *Functionality*

The main functionalities carried out by a Policy Manager instance are:

- *local policy database management,* which includes all the operations used by the administrator to maintain policies applicable in the domain of this instance
- *policy selection,* which allows getting the associated to the request policies
- *policy intersection,* which returns a single policy to the applicant
- *query service,* which is the external interface of the service
- *upper node request handling,* which retrieves policies enforced by upper levels.

Most functionalities of the Policy Manager are of low complexity. The functionality that mainly draws attention is the request of a policy, performed by the operation *requirePolicy*, as it can involve an upper Policy Manager instance (i.e. BaseVO Policy Manager from an OpVO Policy Manager). In following figure, the sequence diagram of this interaction is depicted.

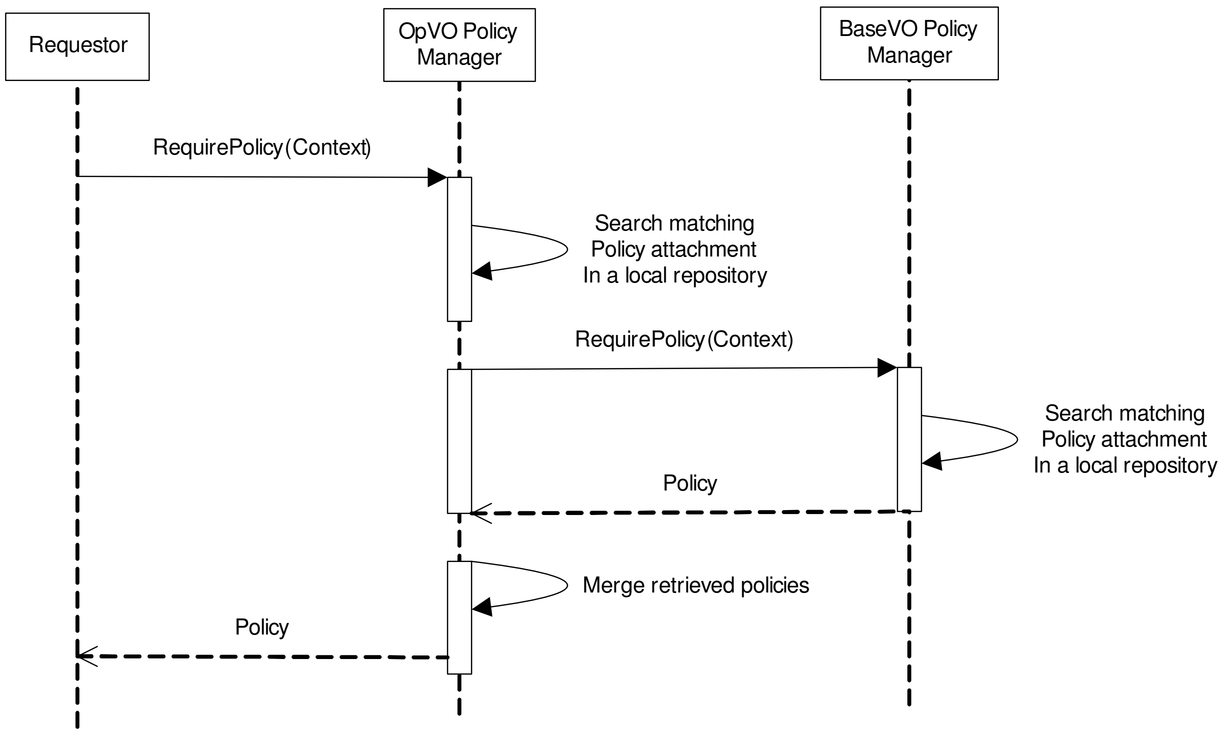

**Figure 26: Require Policy sequence diagram**

#### 2.6.1.2. *Use Cases*

**Table 26: Require Policy Use Case**

| | |
|---|---|
| **Use Case ID** | UC_PM_1 |
| **Use Case Name** | Require policy |

| | |
|---|---|
| **Initiator** | Applicant, i.e. an Akogrimo component (service, application, ...) that needs to know policy information (at this moment, SLA subsystem interfaces with the Policy Manager) |
| **Primary Actor** | Applicant |
| **Additional Actors** | Policy Manager Service. |
| **Description** | The Applicant needs information to perform its own operations and contacts own Policy Manager to retrieve that. |
| **Pre-condition** | The Applicant knows its authoritative Policy Manager reference. |
| **Post-condition** | The result from the Policy Manager (a policy) will be used to perform the operation of the Applicant. |
| **Use Case Functionality** | |
| **Sequence** | Applicant – Policy Manager: The Applicant fills out a Policy Context with relevant information about the situation that it has to handle and asks the Policy Manager for a policy sending this Policy Context<br>Policy Manager: Checks the policies applicable to the Policy Context received and eventually forwards the same request to its own upper Policy Manager<br>Policy Manager – Applicant: The policies retrieved from the local policy store and from the upper Policy Manager service are merged and the result is returned to the Applicant. |
| **Alternatives** | If the Policy Manager instance has its own upper Policy Manager configured, it queries the latter too. |
| **Exceptions** | If, at any time, the Policy Manager encounters an error, it raises an exception to notify the Applicant about the incapability to provide the information |
| **Use Cases used** | None |
| **Further Information** | |
| **Particular Requirements** | None |
| **Open Issues** | None |
| **Information Requirements** | Obviously, the Policy Managers involved should have meaningful Policy Attachments concerning the Policy Context of the Applicant. |

#### 2.6.1.3. *Interactions with other services*

The Policy Manager is the key resource for VO related policies affecting the Akogrimo application as a whole. It offers to any Akogrimo component a single interface for policy requests (see Annex section B.6). The Policy Manager is a standalone service, that is, it does not rely on other services to accomplish its tasks. The only essential relationship it has is with another instance of Policy Manager that provides higher priority policies when configured.

At the moment, the SLA subsystem is using the Policy Manager to retrieve policies. The administration of the Policy Manager should be performed manually with the administration client and no Akogrimo component should use the administration interfaces.

**Table 27: Policy Manager interactions with other services**

| Akogrimo service | Interaction | Protocol |
|---|---|---|
| SLA-Decisor | The SLA-Decisor retrieves from the Policy Manager a policy related to an SLA violation | SOAP/HTTP |
| SLA-Negotiator | The SLA-Negotiator retrieves from the Policy Manager the policy related to the SLA HighLevel-LowLevel Parameter mapping | SOAP/HTTP |
| upper PolicyManager | The local Policy Manager asks its own upper Policy Manager (VO), if configured, to retrieve global policies | SOAP/HTTP |

### 2.6.2. Policy Manager Service Implementation

While most of the service interface parameters are defined to be compliant to WS standards (WS-Policy [29] and WS-PolicyAttachment [30]), some parts have been defined by the developers. In particular, these parts include the policy context provided by the client to the Policy Manager service and the policy scope of WS-PolicyAttachment (the “AppliesTo” tag). Some parts of the XML language are inspired by XACML specification. Below an example of Policy Context used in the requests is provided.

```
<?xml version="1.0" encoding="UTF-8" ?>
<PolicyContext xmlns=”http://www.akogrimo.org/namespaces/PolicyManager”>
  <Subject>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
      DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">
      <AttributeValue>seth@users.example.com</AttributeValue>
    </Attribute>
    <Attribute AttributeId="group"
     DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>developers</AttributeValue>
    </Attribute>
  </Subject>
  <Resource>
```

```
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
      DataType="http://www.w3.org/2001/XMLSchema#anyURI">
      <AttributeValue>http://server.example.com/code/docs/developer-guide.html</AttributeValue>
    </Attribute>
  </Resource>
  <Action>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>read</AttributeValue>
    </Attribute>
  </Action>
</PolicyContext>
```

**Figure 27: Policy Context**

The Policy Context is partitioned into three sections (Subject, Action and Resource) and each section contains a list of attributes; each attribute is featured by AttributeId, DataType and the AttributeValue. Below an example of Policy Attachments, stored in the local database, is provided.

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsp:PolicyAttachment xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
  <wsp:AppliesTo>
    <PolicyContext xmlns="http://www.akogrimo.org/namespaces/PolicyManager">
      <Subject>
        <Attribute AttributeId="group"
          DataType="http://www.w3.org/2001/XMLSchema#string">
          <AttributeValue>developers</AttributeValue>
        </Attribute>
      </Subject> <Resource>
        <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
          DataType="http://www.w3.org/2001/XMLSchema#anyURI">
          <AttributeValue>http://server.example.com/code/docs/developer-guide.html</AttributeValue>
        </Attribute>
      </Resource>
      <Action>
        <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
          DataType="http://www.w3.org/2001/XMLSchema#string">
          <AttributeValue>read</AttributeValue>
        </Attribute>
      </Action>
    </PolicyContext>
  </wsp:AppliesTo>
  <wsp:Policy>
    <sec:ProtectionLevel
      xmlns:sec="http://bscw.hlrs.de/bscw/namespaces/Security">privacy</sec:ProtectionLevel>
    <mon:LogLevel
      xmlns:mon="http://bscw.hlrs.de/bscw/namespaces/Monitoring">NOTICE</mon:LogLevel>
  </wsp:Policy>
</wsp:PolicyAttachment>
```

**Figure 28: Policy Attachment**

The current implementation uses a very simple algorithm for matching the requested context with the context contained in the section "AppliesTo" of the Policy Attachment: all the attributes contained in the latter have to be present in the first one. Hence, for this algorithm, the Policy Attachment example does match the previous Policy Context example.

In the design documents, a caching functionality was outlined, but it isn't currently implemented. The reason is that, after testing and discussing about the usage of the Policy Manager with the other partners, the need of a tool to easily create and edit the Policy Attachment took priority over the caching. However the caching comprises only a performance improvement and doesn't obstruct the functionality of the Policy Manager.

#### 2.6.2.1. Involved Technologies

The Policy Manager Service is implemented on WSRF.NET platform. It does not require other elements, like database, libraries, etc. The implementation has been as compliant as possible to existing WS-Policy and WS-PolicyAttachment standards. The implementation language is C#. Beyond the WSRF.NET libraries, standard .NET distribution libraries are used.

#### 2.6.2.2. Configuration and Deployment

The Policy Manager Service is distributed with an installation package (MSI), an administration client, a simple Policy Attachment editor and a test client. Enclosed with the test client, some test files containing Policy Context, Policy Attachment and Policy are provided. Also a library containing Policy, Policy Attachment and Policy Context objects is included in the distribution; this can be used to develop clients interacting with the service.

After the installation, a WSRF service is installed and the default WSRF resource is reachable at URL:

*http://<host>/PolicyManager/PolicyManagerService/PolicyManagerService.asmx*

Its local database is void and the uplink (to the upper level PolicyManager instance) is null.

In the local file system, the service, the clients and libraries are placed into the IIS root directory (usually *"C:\Inetpub\wwwroot"*) at the relative path "PolicyManager".

To configure the Policy Manager instance the administration client is used: this client allows the setup of the uplink reference and the loading (or removal) of Policy Attachments.

Using the administration client, you can access a WSRF instance (not the default resource) appending to the URL "#<ResourceId>".

### 2.6.3. Security Considerations

Currently, the service does not perform any internal security checks, and does not have any internal security policy. Therefore, all tasks related to security are delegated to the hosting platform, in this case WSRF.NET/IIS as well as the VO it is a member of.

There is a definite requirement for the administrative interfaces of the service to be secured. This is a key issue as the alteration of policies can break the security and policy balance of the whole Akogrimo VO.

The security of Policy information from services is a complex issue due to the mix of public and private information. This issue requires better handling of the applicant identity and its security attributes (authorizations). However, at the moment, we don't have detected peculiar requirements to protect individual information in the Akogrimo environment.

In future Akogrimo implementations it is planned to secure the communication between the SP and VO domains by using encrypted messaging and security tokens.

## 2.7. Semantic Service Discovery Service (SSDS)

The Semantic Service Discovery Service (SSDS) is a central repository of service meta-information, including low level, high level parameters as well as SLAs. The collection of such meta-information in a central repository enables advanced browsing, searching and discovery mechanisms.

### 2.7.1. SSDS Design

The SSDS gives the possibility to enrich the service description with semantic information and provide a plug-in based discovery framework. SSDS also provides an import/export mechanism for technical (WSDL, BPEL) and semantic (OWL, OWL-S) service descriptions. A modelling interface enables the individual adaptation of a service meta-information.

All interfaces are available in two forms:

1. as GUIs for developers and administrator to manually adapt the service meta-data and use the discovery mechanisms as well as
2. in the form of Web Services that are exclusively accessed by the two proxy services: EMS for WP4.3 and GrSDS for WP4.4.

The provided interfaces are implemented by two components: (1) the Semantic Service Modelling Component (SMC) which provides a GUI and a Web Service to edit the meta-information of services and (2) the Semantic Service Discovery Component (SDC) which provides a GUI and a Web Services to discover services. Both components use the Data Access Object (DAO) that communicates with the repository that keeps the meta-information of the services. This repository is an existing Web Service that uses rational databases to implement a meta-model database. Available scripts and configuration files are used by the DAO to adapt the repository for the needs in Akogrimo. More specifically, the SSDS has the following components:

**Semantic Service Modelling Component (SMC)**

The SMC provides a model based design interface. The interface offers methods to list, load, save, rename or delete models and also a method to load the model library definitions. The SMC offers a Java Applet GUI or can be alternatively called as a Web Service.

**Semantic Service Discovery Component (SDC)**

The SDC provides 4 interfaces: (1) the Semantic Service Admin Interface provides methods to upload, activate or deactivate plug-ins or also to change the plug-ins settings, (2) the Semantic Service Discovery Interface can be used to get a list with the active discovery plug-ins (discovery algorithms) and to call a discovery plug-in, (3) the Semantic Service Browsing Interface defines a set of methods that can be used to browse the services by different point of view (by service output, by activity, by context, by provider, etc) and (4) the Semantic Service Import/Export interface defines methods to import or export service definitions (WSDL), service orchestration (BPEL, OWL-S) or semantic service descriptions (OWL, Topic Maps).

The Discovery Framework offers a Java Struts GUI and an eclipse plug-in GUI. The offered functionality is also exposed through a Web Service.

**Data Access Object (DAO)**

The Data Access Object encapsulates the data access through the Meta-model service interface. It is also concerned with caching, load balancing and the implementation of fault resistance mechanisms to guarantee the required performance.

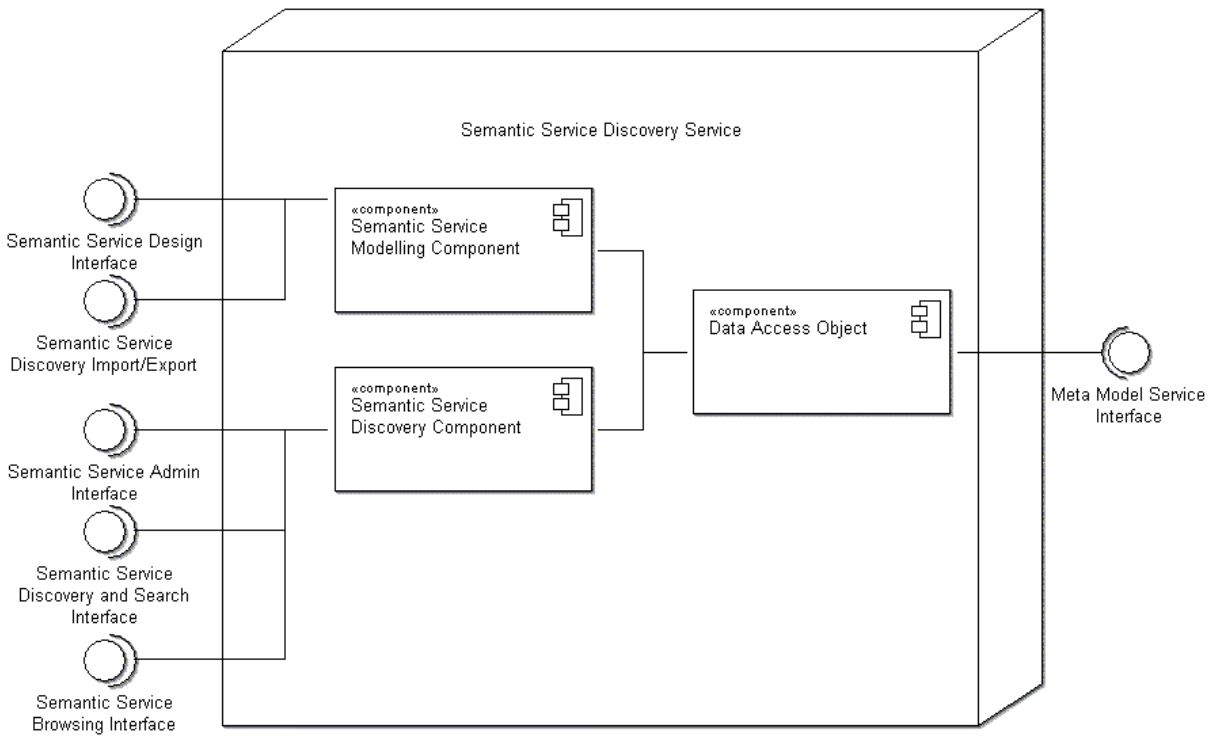
Semantic Service Discovery Service
«component» Semantic Service Modelling Component
«component» Semantic Service Discovery Component
«component» Data Access Object
Semantic Service Design Interface
Semantic Service Discovery Import/Export
Semantic Service Admin Interface
Semantic Service Discovery and Search Interface
Semantic Service Browsing Interface
Meta Model Service Interface

**Figure 29: Semantic Service Discover Service: Interfaces and Components**

### 2.7.1.1. Functionality

According to the OGSA specifications, the SSDS is a preparatory service for the EMS, providing support to the discovery of candidate locations. As the semantic discovery mechanisms are not limited to low level parameters, the SSDS is also a preparatory service for the GrSDS (Grid Service Discovery Service). The objectives that are met by the SSDS are listed below:

- ***Discovery of services:*** The SSDS should be able to "discover" services according low-level, high-level parameters and service level agreements. The parameters for the discovery are application-depending customisation of the SSDS. Standard discovery mechanisms are provided and can be adapted by the requesting service.
- ***Modelling of services:*** The SSDS enables the storing of service meta-data in one central repository. Services can be modelled by other services such as indexing or agent services, or by developers or service providers that would like their services to be used under specific – modelled – circumstances.
- ***Management of service discovery:*** The SSDS enables to access the service meta-information to evaluate the service description and analyse the according service execution. The comparison of real service execution data and the service description enables optimisation by appropriate model changes.

The functionality of the SSDS can be categorised into three components: (1) Semantic Service Model Component (SMC), (2) Semantic Service Discovery Component (SDC) and (3) Data Access Object (DAO).

### Semantic Service Model Component (SMC)

The SMC enables methods to describe a service via meta-information such as low-level parameter, high level parameter and service level agreements. The service description via meta-information is seen as a model. There are different types of models distinguishing the type of the service description. A service model describes the service coordinates and how to contact and access it, a workflow model type describes the sequence of services or a semantic adaptive workflow describes the sequence of services with additional semantic annotation.

The Semantic Service Model Component provides an interface to edit the service description either via a Java Applet GUI or via direct Access to the interface Web Service.

In the following the model type architecture is presented distinguishing between requirement models that describe the service requirements and the providing models that describe what services can provide.

**Table 28: Meta-Model for Service Discovery**

| Model type | Description |
|---|---|
| ***Process Requirements*** | |
| **Semantic Workflow** | Contains all the activities, sub process calls and decisions that are needed to rich a goal. Every activity can be enriched with information like data needed by the activity, the expected service level, needed resources and effects. This information is used as the requirement for the service discovery. |
| ***Semantic Level*** | |
| **Topic map model** | The Topic Map model is a low-level ontology language that is restricted to some modelling concepts defined in the topic map standard. |
| **Ontology model** | If the semantic description requires additional concepts to the one provided in the topic map, the service modeller can build its own ontology. |
| ***Service Provision*** | |
| **Aggregated Service Model** | The aggregated Service Model describes aggregated services using BPEL templates referring to Atomic Service Models. |
| **Atomic Service Model** | Contains the service definitions. Every service can provide one or more methods. One method can be provided by more then one services. |
| ***Service Description Models*** | |
| **Provider model** | This model is used to model the service provider and the offered services from the provider. |

| | |
|---|---|
| **SLA Model** | Contains the definitions of the SLA Contracts. Every SLA Contract has the 4 standard quality of service (QoS) parameters (response time, latency, availability, and throughput). Every service can have also service specific parameters. |
| **Service Source Model** | Defines the service sources (input/output) as middle layer between the service method and the data type's model. |
| **Data Types Model** | The data type definitions according to WSDL. |
| **Service Effects Model** | This model type contains the service effects. The effects are system state changes that are result or precondition of the service calls. |

Table 28: Meta-Model for Service Discovery introduces the meta-model used for Service Discovery by listing the model types and the according service meta-information. The service requirements to be discovered are represented in form of semantic workflows and semantic models.

The provisioning models describe available services with some additional information regarding effect, input/output and data types and their semantic description.

All functionalities necessary to model service requirements and service provision are provided by the Semantic Service Model Component.

## *Semantic Service Discovery Component (SDC)*

The SDC offers a plug-in based discovery engine. In this scenario we distinguish between:

- *Browsing,* for viewing and looking through existing models following references form one model to another,
- *Searching,* for identifying the location of a given service or identify its existence and
- *Discovering,* for identifying new insights for a new situation.

Although searching and discovering are very closely related terms – and sometimes used as a synonym – it is made the distinction to separate searching functionality that is concerned with the identification of terms in the model repository from discovery algorithms that identify possible services based on requirement models. The main difference is that using the searching features, the requestor can identify, compare and analyse information that is explicitly stored in the model repository, whereas the discovery feature enables the approximation of information which is not explicitly stored in the model repository but strongly depends on the discovery algorithm.

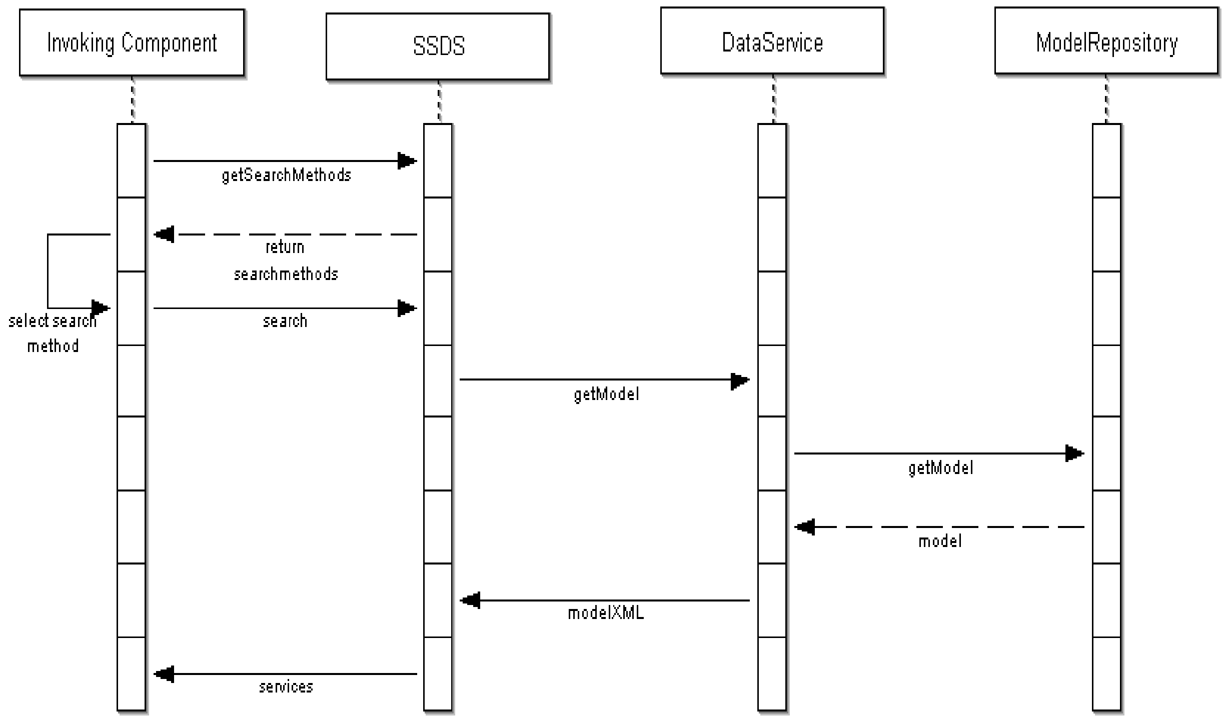

**Figure 30: Sequence diagram Service Discovery**

The Semantic Service Discovery Component offers data access over the Data Access Object using the browsing interface and a generic discovery plug-in framework. Discovery plug-ins can be written by any Java programmer using the discovery plug-in API.

Every plug-in must implement the interface DiscoveryMethodPlugIn and write the following methods:

- *getMethod* – the method returns the name of the discovery method and a list with all parameters needed by the discovery.
- *install* – this method initialises the plug-in. The method becomes the Discovery Dao Factory and the plug-in properties values as Input.
- *discover* – this method becomes as input the parameter vector and do the discovery. The return value is a list with services.
- *getProperties* – return a list with the properties of the plug-in containing the default values.

Discovery plug-ins can be uploaded, activated and configured by the admin interface. Basic search and discovering mechanisms are provided by default, but there is the opportunity that each proxy service requires a set of individual discovery functions.

The SSDS can be queried for the available discovery methods. As soon as one of them is selected a search can be invoked. The discovery plug-in received via the Data Access Object the data from the repository. A set of services will be found, according the search-parameters specified (see Figure 30: Sequence diagram Service Discovery).

### *Data Access Object*

The Data Access Object is responsible to provide data from the model repository that is needed by the other components. The Data Access Object covers also the caching, failover and the load balancing. Section B.7 in Annex B provides an overview of the offered methods.

The Data Access Object provides all functionalities required for the other components in the SSDS to access the model repository. The DAO interface is not provided externally.

### 2.7.1.2. Use Cases

We identify three different users of the SSDS: (1) the Service Network Manager who is responsible for the correct modelling of the services. This user specifies the level of semantic detail and framework that can be used by the service provider to register their services, (2) the SP who offers either a single service or a set of services and is able to register services in the framework specified by the Service Network Manager. In case the SP registers a set of services, it is also able to provide discovery strategies and (3) the Service Consumer who is able to interact with three different discovery mechanisms by first contacting the SSDS to get a list of discovery methods. The service consumer selects the appropriate discovery method and discovers. The use-cases identified for these three users are listed below.

**Service Network Manager**

The Service Network Manager can interact with the system in three different ways:

- Modelling Service Requirements (Defining service requirements)
- Modelling Service Semantic (Creating a semantic description, for the classification of the service)
- Modelling Service Provision (Registering the information, how the service will be provided)

**Service Provider**

For the Service Provider two uses-cases were identified:

- Register Service (Register the service with its WSDL description and its categories, so that is can be discovered)
- Upload Discovery Plug-In (Register a new Plug-In that allows discovering services)

**Service Consumer**

For the Service Consumer there are three use-cases:

- Browsing for services (Find services not by specifying criteria directly but browse through categories or output)
- Searching for services (Find services by specifying criteria)
- Discovering services (Find services with the discovery Plug-Ins)

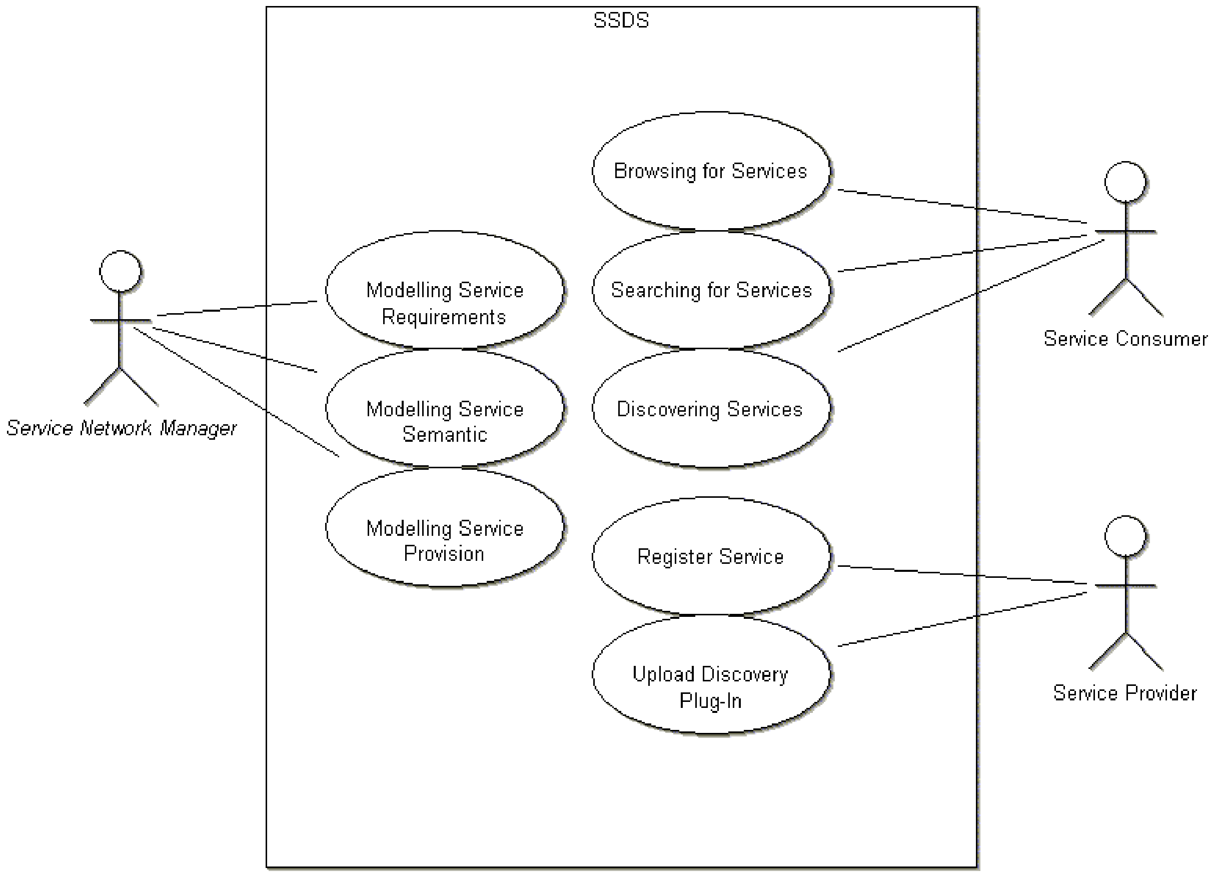

**Figure 31: SSDS use cases**

**Table 29: Modelling Service Requirements Use Case**

| | |
|---|---|
| **Use Case ID** | UC_SSDS_1 |
| **Use Case Name** | Modelling Service Requirements |
| **Initiator** | Service Network Manager |
| **Primary Actor** | Service Network Manager |
| **Additional Actors** | None |
| **Description** | Modelling the service requirements through the modelling component |
| **Pre-condition** | None |
| **Post-condition** | None |
| **Use Case Functionality** | |
| **Sequence** | Invoke the modelling mechanisms for service requirements |

| | |
|---|---|
| **Alternatives** | None |
| **Exceptions** | None |
| **Use Cases used** | None |
| **Further Information** | |
| **Particular Requirements** | None |
| **Open Issues** | None |
| **Information Requirements** | None |

**Table 30: Modelling Service Semantic Use Case**

| | |
|---|---|
| **Use Case ID** | UC_SSDS_2 |
| **Use Case Name** | Modelling Service Semantic |
| **Initiator** | Service Network Manager |
| **Primary Actor** | Service Network Manager |
| **Additional Actors** | None |
| **Description** | Create a semantic description to classify services later on |
| **Pre-condition** | None |
| **Post-condition** | Semantic description in form of categories, etc. exists |
| **Use Case Functionality** | |
| **Sequence** | Invoke the modelling mechanisms for semantic description |
| **Alternatives** | None |
| **Exceptions** | None |
| **Use Cases used** | None |
| **Further Information** | |

| | |
|---|---|
| **Particular Requirements** | None |
| **Open Issues** | None |
| **Information Requirements** | None |

**Table 31: Modelling Service Provision Use Case**

| | |
|---|---|
| **Use Case ID** | UC_SSDS_3 |
| **Use Case Name** | Modelling Service Provision |
| **Initiator** | Service Network Manager |
| **Primary Actor** | Service Network Manager |
| **Additional Actors** | None |
| **Description** | Registering the information for the service provision, e.g. endpoint, SLA |
| **Pre-condition** | Service is registered |
| **Post-condition** | Provision information for service exists |
| **Use Case Functionality** | |
| **Sequence** | Invoke the modelling mechanisms to register information for service provision |
| **Alternatives** | None |
| **Exceptions** | None |
| **Use Cases used** | None |
| **Further Information** | |
| **Particular Requirements** | None |
| **Open Issues** | None |
| **Information Requirements** | None |

**Table 32: Register Service Use Case**

| | |
|---|---|
| **Use Case ID** | UC_SSDS_4 |
| **Use Case Name** | Register Service |
| **Initiator** | Service Provider |
| **Primary Actor** | Service Provider |
| **Additional Actors** | None |
| **Description** | A Service is registered and classified in order to search for it later on |
| **Pre-condition** | Service semantics were modelled |
| **Post-condition** | Service(s) are registered |
| **Use Case Functionality** | |
| **Sequence** | Register service with WSDL information and categories |
| **Alternatives** | None |
| **Exceptions** | Service definition is not valid (WSDL) |
| **Use Cases used** | None |
| **Further Information** | |
| **Particular Requirements** | None |
| **Open Issues** | None |
| **Information Requirements** | WSDL-file of the service or URI of WSDL-file,<br>Categories for service-classification |

**Table 33: Upload Discovery plug-in Use Case**

| | |
|---|---|
| **Use Case ID** | UC_SSDS_5 |
| **Use Case Name** | Upload Discovery Plug-in |
| **Initiator** | Service Provider |

| | |
|---|---|
| **Primary Actor** | Service Provider |
| **Additional Actors** | None |
| **Description** | Upload a new plug-in to discover services. The service supports different discovery mechanisms which can be uploaded on demand. |
| **Pre-condition** | None |
| **Post-condition** | One or more discovery plug-ins exist |
| **Use Case Functionality** | |
| **Sequence** | Upload plug-in |
| **Alternatives** | None |
| **Exceptions** | Plug-in is already existing |
| **Use Cases used** | None |
| **Further Information** | |
| **Particular Requirements** | None |
| **Open Issues** | None |
| **Information Requirements** | None |

**Table 34: Browsing for Services Use Case**

| | |
|---|---|
| **Use Case ID** | UC_SSDS_6 |
| **Use Case Name** | Browsing for Services |
| **Initiator** | Service Consumer |
| **Primary Actor** | Service Consumer |
| **Additional Actors** | None |
| **Description** | Find services not by specifying criteria directly but browse through categories or output |

| | |
|---|---|
| **Pre-condition** | Discovery Service is running, Search Plug-Ins were registered, Service descriptions are registered |
| **Post-condition** | None |
| **Use Case Functionality** | |
| **Sequence** | 1. Choose the way of browsing<br>2. Choose category<br>3. View result |
| **Alternatives** | None |
| **Exceptions** | None |
| **Use Cases used** | None |
| **Further Information** | |
| **Particular Requirements** | None |
| **Open Issues** | None |
| **Information Requirements** | None |

**Table 35: Searching Services Use Case**

| | |
|---|---|
| **Use Case ID** | UC_SSDS_7 |
| **Use Case Name** | Searching Services |
| **Initiator** | Service Consumer |
| **Primary Actor** | Service Consumer |
| **Additional Actors** | None |
| **Description** | Find services according to criteria to later invoke them |
| **Pre-condition** | Discovery Service is running, Search Plug-Ins were registered, Service descriptions are registered |
| **Post-condition** | None |

| Use Case Functionality | |
|---|---|
| **Sequence** | 1. Specify query to search for services<br>2. Result of query will be returned |
| **Alternatives** | None |
| **Exceptions** | The criteria do not match the allowed values |
| **Use Cases used** | None |
| **Further Information** | |
| **Particular Requirements** | None |
| **Open Issues** | None |
| **Information Requirements** | None |

**Table 36: Discovering Services Use Case**

| | |
|---|---|
| **Use Case ID** | UC_SSDS_8 |
| **Use Case Name** | Discovering Services |
| **Initiator** | Service Consumer |
| **Primary Actor** | Service Consumer |
| **Additional Actors** | None |
| **Description** | Find services according to criteria to later invoke them |
| **Pre-condition** | Discovery Service is running, Search Plug-Ins were registered, Service descriptions are registered |
| **Post-condition** | None |
| **Use Case Functionality** | |

| | |
|---|---|
| **Sequence** | 1. Get the list of available Discovery Methods<br>2. Select one method<br>3. Send the criteria to according to the Discovery Method to the Search engine<br>4. Result of query will be returned |
| **Alternatives** | None |
| **Exceptions** | No Discovery methods are available<br>The criteria do not match the allowed values |
| **Use Cases used** | None |
| **Further Information** | |
| **Particular Requirements** | None |
| **Open Issues** | None |
| **Information Requirements** | None |

### 2.7.1.3. *Interactions with other services*

All interfaces are Web Service calls using SOAP protocol from the proxy services. In the following table a more detailed look on the offered interfaces is provided.

**Table 37: SSDS interactions with other Akogrimo services**

| Akogrimo service | Interaction | Protocol |
|---|---|---|
| EMS | -To discover candidate locations for the business service execution<br>-To index and model the business service locations | SOAP/HTTP |
| GrSDS | -To discover candidate SPs<br>-To index and model the SP related information | SOAP/HTTP |

## 2.7.2. SSDS Implementation

The SSDS is a preparatory service that interacts with defined proxy services. The SSDS provides a flexible discovery framework that enables browsing, searching and discovering of services in a meta-data repository and according modelling and administration features that are used by the proxy services and offered to other Akogrimo services via the proxy.

- **EMS**

  The EMS is a proxy service that uses the SSDS for service candidate location in WP 4.3. The discovery interface acts not only as a requestor for service discovery but also as an interface for indexing and modelling the service locations by other WP 4.3 Akogrimo services. Each registering service provides additional meta-information for the service discovery; the proxy service forwards the service meta-information to the SSDS.

- **GrSDS**

  The GrSDS is a proxy service that uses the SSDS for Grid service discovery in WP 4.4. The GrSDS acts not only as a requestor for service discovery but also as an interface for indexing and modelling for Grid Services by other WP 4.4 Akogrimo services. Each registering service provides additional meta-information for the service discovery; the proxy service forwards the service meta-information to the SSDS.

**Import /Export Service Meta-information**

Indexing services can enter their information via the import/export interface of the proxy service.

**Administration**

The proxy service administrates the SSDS by up- / downloading the preferred discovery mechanism. This enables that the discovery service decides based on the application scenario the selected discovery algorithm.

#### 2.7.2.1. Involved Technologies

The SSDS uses a meta-model repository based on the meta2model framework of MOF. The description of the services is made in WSDL format and in case of aggregated services in BPEL format.

For semantic annotation of workflows, the OWL-S notification is used and enriched by semantic models Topic Maps and OWL. Standard formats like OWL-S, OWL and SWRL xfire.codehaus.org will be used to include open source reasoner in advanced discovery plug-ins.

#### 2.7.2.2. Configuration and Deployment

The SSDS is a Java service that runs within Tomcat to provide the Web Service interfaces via Xfire [34]. A JDK 1.4.2_06 or higher (incl. 1.5.x) is required to run all the interfaces of the SSDS.

The Data Access Object has to be implemented on a Windows XP machine, as it is a gSoap Web Service in a C++ runtime. The installation of the Data Access Object requires also the installation and configuration of the rational database MSDE.

The communication between the Semantic Service Model / Discovery Component and the Data Access Object is using SOAP. So it is possible to install the Tomcat and Java service environment on a Linux machine and separate the Data Access Object to a Windows XP machine.

### 2.7.3. Security Considerations

The SSDS communicates exclusively with registered proxy services and trusts that the proxy services establish safe communication with other Akogrimo services. The requesting and responding xml streams will be encrypted.

# 3. Interoperability issues

## GT4 vs WSRF.NET

For the development of the Grid Services in WP4.3, two of the most popular platforms in the development of Grid Services were used: GT4 and WSRF.NET. The following table summarizes the services which services were developed on which platform.

**Table 38: GT4 and WSRF.NET services**

| | |
|---|---|
| **GT4 services** | EMS, DMS, Monitoring, Metering |
| **WSRF.NET** | SLA-Controller, SLA-Decisor, Policy Manager |

The Akogrimo architecture specifies many interactions between services that were built on those different platforms. For example, EMS and Monitoring (GT4) interact with the SLA Enforcement group of services (WSRF.NET). For this reason, these interactions were not only challenging in terms of design and efficiency but in terms of interoperability as well, since it allowed for checking whether these two popular Grid development tools, implement the WSRF and WS-related specifications in a transparent and interoperable way. During the establishment of communication between them, a small number of inconsistencies were detected and were handled in different ways and at different levels with some of them even requiring changing the form of the SOAP messages. These interoperability problems have been gathered in the Akogrimo internal document "Interoperability Issues" [21].

# 4. Conclusions

This deliverable presented the design and implementation of the services that belong to the WP4.3 Grid Infrastructure Services Layer of the Akogrimo project. In this Work Package, the Grid Infrastructure Services Layer defined a service-oriented architecture which consists of a set of services capable of providing the core Grid functionality. These capabilities included issues such as service discovery, advance reservation, execution management, monitoring, SLA enforcement, policy management and others. The presented services involve the implementation efforts that the WP4.3 partners carried out until project month 30. The maintenance of these software components as well as their enhancements (with respect to security aspects and integration within the final Demonstrator) will be continued until project month 36.

# References

[1] Akogrimo Deliverable D4.3.1 "Architecture of the Infrastructure Services Layer" https://bscw.hlrs.de/bscw/bscw.cgi/0/36376.

[2] Akogrimo Deliverable D4.3.2."Prototype Implementation of the Infrastructure Services Layer" https://bscw.hlrs.de/bscw/bscw.cgi/0/36380.

[3] Akogrimo Deliverable D3.1.1 "Overall Architecture Version 1" https://bscw.hlrs.de/bscw/bscw.cgi/0/36348.

[4] Akogrimo Deliverable D3.1.3 "Overall Architecture Definition and Layer Integration" https://bscw.hlrs.de/bscw/bscw.cgi/0/109536.

[5] *Global Grid Forum (GGF)*. http://www/ggf/org/.

[6] *OASIS WSRF Technical Committee*. http://www.oasisopen.org/committees/ tc_home.php?wg_abbrev=wsrf.

[7] *OASIS*. http://www.oasis-open.org/.

[8] *GlobusToolkit 4*. http://www.globus.org/toolkit/.

[9] *The Globus Alliance*. http://www.globus.org/.

[10] *WSRF*.NET. http://www/cs.virginia.edu/~gsw2c/wsfr.net.html.

[11] *OGSA-DAI*. http://www.ogsadai.org.uk/.

[12] *"WS-Resource specification"*. 1.2 Working Draft 03. Web Services Resource Framework (WSRF) TC. OASIS. March 8, 2005

[13] *"WS-ResourceProperties specification"*. 1.2 Working Draft 01. Web Services Resource Framework (WSRF) TC. OASIS. June, 2004

[14] *"WS-ResourceLifetime specification"*. 1.2 Working Draft 01. Web Services Resource Framework (WSRF) TC. OASIS. June, 2005

[15] *"WS-BaseNotification specification"*. 1.2 Working Draft 03. Web Services Notification (WSN) TC. OASIS. June, 2005

[16] *"WS-Topics specification"*. 1.2 Working Draft 03. Web Services Notification (WSN) TC. OASIS. June, 2005

[17] K.Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling and S. Tuecke. "From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring & Evolution". March 5, 2004. http://www.106.ibm.com/developerworks/library/ws-resource/ogsi_to_wsrf_1.0.pdf.

[18] *Official Globus Security Documentation*. http://www.globus.org/toolkit/docs/4.0/security/.

[19] *"WS-Security specification"*. 1.0. Web Services Security (WSS) TC. OASIS, January, 2004

[20] *"Web Services Secure Conversation Language (WS-SecureConversation)"*. IBM et al., May, 2004. http://www-106.ibm.com./developersworks/library/specification/ws-secon/.

[21] *"Interoperability issues"*. Akogrimo internal document. http://bscw.hlrs.de/bscw/bscw.cgi/0/162191.

[22] *GT4.0 WS-GRAM*. http://www-unix.globus.org/toolkit/docs/4.0/execution/wsgram/.

[23] *GT Information Services: Monitoring & Discovery System (MDS)*. http://www.globus.org/toolkit/mds/.

[24] *GT Data Management: GridFTP*. http://www.globus.org/toolkit/data/gridftp/.

[25] *"WS-Agreement specification"*. March 8, 2005. https://forge.gridforum.org/projects/graap-wg/document/WS-AgreementSpecificationDraft.doc/en/10

[26] Web Service Level Agreement (WSLA) Language Specification. http://www.research.ibm.com/people/a/akeller/Data/WSLASpecV1-20030128.pdf

[27] IBM Web Services Toolkit. http://www.alphaworks.ibm.com/tech/webservicestoolkit

[28] Emerging Technologies Toolkit (ETTK). http://www.alphaworks.ibm.com/tech/ettk

[29] *"WS-Policy specification"*. OASIS. September, 2004. ftp://www6.software.ibm.com/software/developer/library/ws-policy.pdf

[30] *"WS-PolicyAttachement specification"*. OASIS. September, 2004. ftp://www6.software.ibm.com/software/developer/library/ws-polat.pdf

[31] *"Policy Driven Management for Distributed System*". Morris Sloman, Journal of Network and System Management, Plenum Press. Vol.2 No. 4, 1994

[32] *"Conflicts in Policy-Based Distributed Systems Managemen"*. Emil Lupu and Morris Sloman. IEEE Transaction On Software Engineering, Vol 25, No. 6, Nov/Dec 1999

[33] *"Automated Policy-Based Resource Construction in Utility Computing Environments"*. Akhil Sahai, Sharad Singhal, Rajeev Joshi and Vijiay Machiraju. http://www.hpl.hp.com/techreports/2003/HPL-2003-176.pdf

[34] *Codehaus Xfire.* http://xfire.codehaus.org/.

# Annex A. Design Details

## EMS – SIP Broker interaction

### A. Setup

1. EMS subscribes to the SIP Broker's Notification Producer service in order to receive notifications about the availability and the location of the mobile users.
2. EMS requests the connection details of each registered mobile user by invoking the *getConnectionDetails* method on the SIP Broker WSRF service.
3. SIP Broker WSRF service contacts the Broker Engine requesting a SIP session for the mobile user.
4. The SIP Broker Engine responds with the connection details of the mobile user.
5. The connections details of the mobile user are sent to the EMS via the SIP Broker WSRF service.

### B. Runtime (Mobile user's connection lost)

1. The sequence is triggered when the Broker Engine detects that the connection of a mobile user is lost.
2. The Broker Engine informs the SIP Broker Notification Producer about the lost connection.
3. The SIP Broker Notification Producer service launches a notification.
4. The notification is received by the EMS.
5. The EMS updates the registry it uses to store information about the mobile users. In particular the mobile user and the services he was in charge of are marked as unavailable.

### C. Runtime (Mobile user's connection recovered)

1. The sequence is triggered when the Broker Engine detects that the connection of a mobile user has been recovered.
2. The Broker Engine informs the SIP Broker Notification Producer service about the recovered connection.
3. The SIP Broker's Notification Producer service launches a notification.
4. The notification is received by the EMS.
5. EMS requests the new connection details of the mobile user from SIP Broker WSRF service.
6. SIP Broker WSRF service contacts the Broker Engine requesting a SIP session for the mobile user.
7. The Broker Engine responds with the connection details of the mobile user.
8. The new connections details of the mobile user are sent to the EMS via the SIP Broker WSRF service.
9. The EMS updates the registry it uses to store information about the mobile users. In particular the mobile user and the services he is now in charge of are marked as available. The new URI of the services are also stored in the registry.

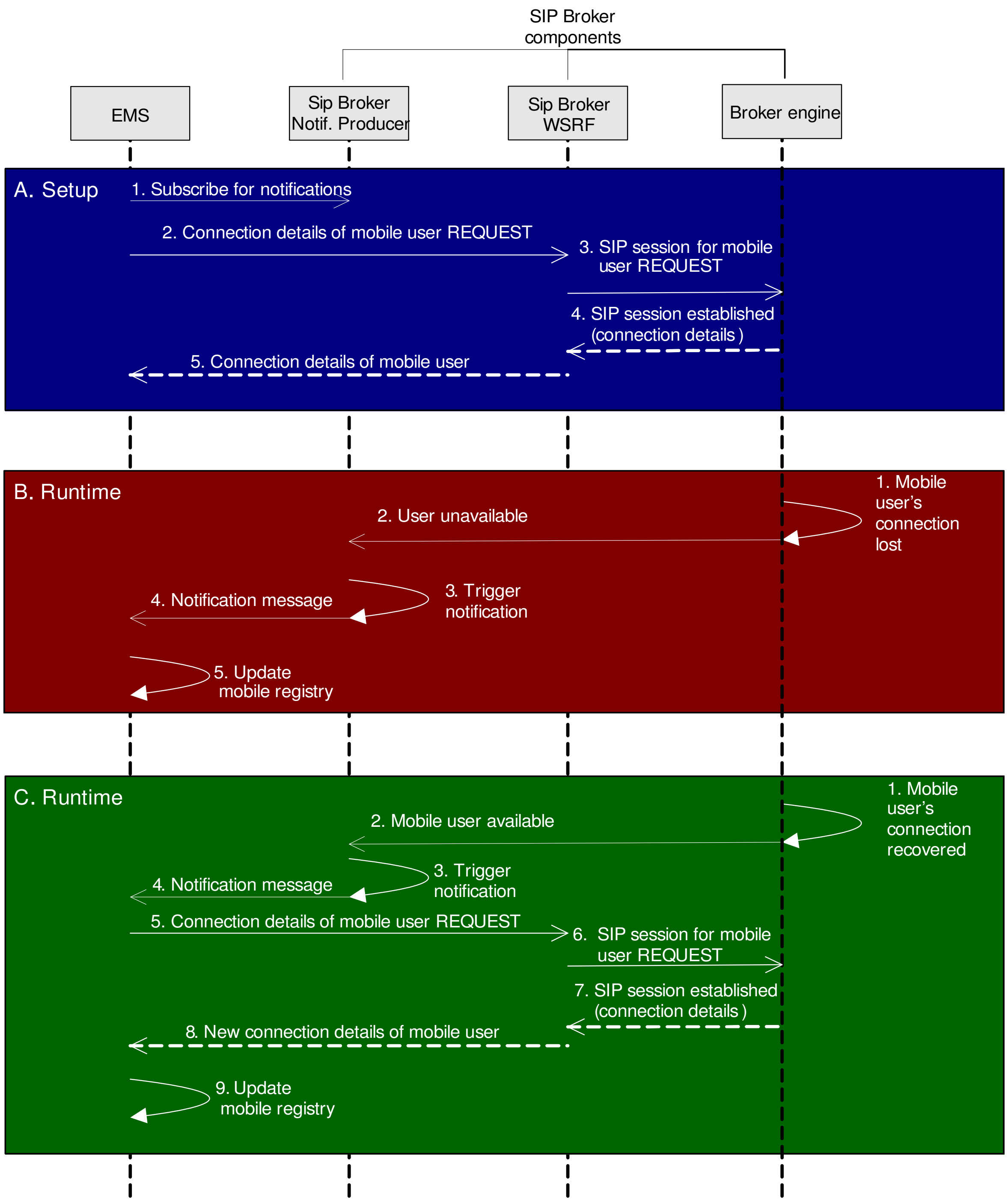
SIP Broker components
EMS
Sip Broker Notif. Producer
Sip Broker WSRF
Broker engine
A. Setup
1. Subscribe for notifications
2. Connection details of mobile user REQUEST
3. SIP session for mobile user REQUEST
4. SIP session established (connection details )
5. Connection details of mobile user
B. Runtime
1. Mobile user's connection lost
2. User unavailable
3. Trigger notification
4. Notification message
5. Update mobile registry
C. Runtime
1. Mobile user's connection recovered
2. Mobile user available
3. Trigger notification
4. Notification message
5. Connection details of mobile user REQUEST
6. SIP session for mobile user REQUEST
7. SIP session established (connection details )
8. New connection details of mobile user
9. Update mobile registry

**Figure 32: EMS-SIP Broker interactions**

# Annex B. API of Services

## B.1. EMS API

**Table 39: EMS Interface**

| Method | Description |
|---|---|
| **Core Gateway service** | |
| **checkResourcesAvailability** | **Input:** CheckResourcesAvailability (a customized object containing the serviceID, the clientID and the values for the low level parameters)<br>**Output:** An EPR to the Negotiation resource, serialized into a string |
| **performAdvanceReservation** | **Input:** PerformExecution (a customized object containing the Reservation EPR, the slaID, the name of the method that will be invoked on the business service and the input needed by this method)<br>**Output:** An EPR to the Negotiation resource, serialized into a string |
| **performExecution** | **Input:** An EPR to a Reservation resource<br>**Output:** A string containing the result of the execution |
| **Negotiation service** | |
| **negotiate** | **Input:** Negotiate (a customized object containing the serviceID, the clientID and the values for the low level parameters)<br>**Output:** NegotiateResponse (a customized object containing the EPR to the Negotiation resource and the URI of the discovered business service) |
| **Reservation service** | |
| **reserve** | **Input:** Reserve (a customized object containing an EPR to the Negotiation resource)<br>**Output:** ReserveResponse (a customized object containing the EPR to the Reservation resource) |
| **Execution service** | |
| **execute** | **Input:** Execute (a customized object containing the Reservation EPR, the slaID, the name of the method that will be invoked on the business service and the input needed by this method)<br>**Output:** ExecuteResponse (a customized object containing the result of the business service execution. In case the execution fails, the object contains the reason for the failure) |

| Discovery service | |
|---|---|
| **discover** | **Input:** Discover (a customized object containing the serviceID, and the values for the low level parameters)<br>**Output:** DiscoverResponse (a customized object containing a list of candidate locations) |
| **Advertisement service** | |
| **advertiseService** | **Input:** AdvertiseService (a customized object containing the serviceID and the values of the low level QoS properties)<br>**Output:** AdvertiseServiceResponse (a customized object containing the EPR to the Advertisement resource) |

## B.2. DMS API

**Table 40: DMS Interface**

| Method | Description |
|---|---|
| **upload** | **Input**: usrID (user identifier), srcUrl (source url of the file to be uploaded), filename (filename of the file to be uploaded), attributes (a string that can help identifying the file)<br>**Output**: fileID (unique file identifier) |
| **retrieve** | **Input**: userID (user identifier), filename (filename of the file to be retrieved), destUrl (destination url where the file should be copied)<br>**Output**: Void |
| **transfer** | **Input**: userID (user identifier), srcUrl (source url of the file to be transferred), destUrl (destination url of the file)<br>**Output**: Void |
| **cancel** | **Input**: userID (user identifier), fileID (unique file identifier, returned by the upload method)<br>**Output**: Void |
| **readDataFile** | **Input**: userID (user identifier), DBname (database identifier where the files have been stored), fileID (unique file identifier, returned by the upload method)<br>**Output**: fileCont (file content) |
| **queryData** | **Input**: userID (user identifier), DBname (database identifier where the files have been stored), queryStatement (MySQL statement that contains the query to be performed)<br>**Output**: fileID (unique file identifier, returned by the upload method), data (answer to the query statement) |

## B.3. Monitoring Service API

Table 41: Monitoring Interface

| Method | Description |
|---|---|
| **startMonitoring** | **Input:** (1) SLAid (SLA Contract ID to be monitorized) , (2) MeteringEPR (EPR of the resource supplying low level parameters info - Subscription mechanism is established between Metering and Monitoring), (3) SLAControllerEPR (EPR of the resource in charge of checking if a SLA violation has occurred - Subscription mechanism is established between Monitoring and SLAController) and (4) QoSParams ( Range of acceptable values of the low level parameters including in the SLA Contract)<br>**Output:** A string:<br>"200 OK" in case the activation/subscription is successful<br>"200 KO + Exception" in case of failure |
| **stopMonitoring** | **Input:** Void<br>**Ouput:** A string:<br>"200 OK" in case the deactivation is successful<br>"200 KO + Exception" in case of failure |

## B.4. SLA Enforcement Service API

Table 42: SLA Enforcement Interface

| Method | Description |
|---|---|
| **SLA-Controller** | |
| **ActiveServiceController** | **Input:** (1) serviceID (the ID of the service instance has to be controlled -string), (2) serviceType (the type of the service instance – string) (3) ObjQoSData (an object containing information about the QoS parameters to be measured) and (4) contractID (the SLA Contract document identifier - string) (5) topic (the topic for the notification process – string)<br>**Output:** Void |
| **receiveNotify** | **Input:** objQoS (an object representing the QoS parameters to measured)<br>**Output:** Void |
| **SLA-Decisor** | |
| **ActiveSLADecisor** | **Input:** (1) sourceEPR (string–ServiceControllerInstanceEPR) and (2) ServiceId (string – the ID of the service that is going to be |

| | |
|---|---|
| | controlled)<br>**Output:** EPR to the SLA-Decisor |
| **receiveNotify** | **Input:** objViolation (an object representing the violation, containing information like the type of the violation, the time the violation occurred, etc)<br>**Output:** Void |
| **retrieveBestPolicy** | **To be implemented:** The SLA-Decisor shall interact with the Policy Manager to retrieve the best policy related to the business service. The later shall be able to understand the violation typology (soft, medium or hard) and decide what necessary actions must be undertaken, depending on the affiliated policy and the status of the system. |
| **SendViolationInformation** | **Input:** obj_violInfo (an object representing the information on the type of violation and the corrective action that should be taken)<br>**Output:** Void |

## B.5. Metering Service API

**Table 43: Metering Interface**

| Method | Description |
|---|---|
| **createMeteringResource** | **Input:** CreateMeteringResource (an object containing information for the initialization of the Metering resource)<br>**Output:** the EPR to the Metering resource |
| **startMetering** | **Input:** slaID (the ID of the SLA contract)<br>**Output:** Boolean value indicating success |
| **stopMetering** | **Input:** StopMetering (an object containing information used for locating the correct Meter thread)<br>**Output:** Boolean value indicating success |

## B.6. Policy Manager Service API

**Table 44: Policy Manager Interface**

| Method | Description |
|---|---|
| **Service Interface** | |
| **requirePolicy** | **Input:** PolicyContext (a structure containing attributes about the context for which the policy is required; for details see the implementation doc) |

| | |
|---|---|
| | **Output:** Policy (the returned Policy, a WS-Policy compliant object) |
| **Administration Interface** | |
| **listPolicyAttachmentIds** | **Input:** None<br>**Output:** An array of strings (the list of PolicyAttachment loaded into the local database; each id will be used to operate over the related PolicyAttachment) |
| **addPolicyAttachment** | **Input:** (1) policyAttachmentId (a string, the key for later access the PolicyAttachment stored inside the database; no duplicated id is permitted) and (2) policyAttachment (the PolicyAttachment to store into the database)<br>**Output:** Void |
| **removePolicyAttachment** | **Input:** policyAttachmentId (a string, the id of the PolicyAttachment to remove from datatabase)<br>**Output:** Void |
| **getPolicyAttachment** | **Input:** policyAttachmentId (a string, the id of the PolicyAttachment to be retrieved from database)<br>**Output:** Void |
| **setUplinkReference** | **Input:** uplinkReference (an EPR, the reference of the upper PolicyManager service)<br>**Output:** Void |
| **getUplinkReference** | **Input:** None<br>**Output:** An EPR (the reference of the upper PolicyManager service) |

## B.7. SSDS API

**Table 45: SSDS interface**

| Method | Description |
|---|---|
| **Semantic Service Discovery Interface** | |
| **getDiscoveryMethods** | **Input:** None<br>**Output:** List with all discovery methods |
| **discover** | **Input:** Filter (this object contains the search method name and a list with key value pairs)<br>**Output:** List with services that match the criteria vector |
| **getEndpoints** | **Input:** (1) serviceName (the name of the service) and (2) slaParameterList (list with sla parameter that must be |

| | |
|---|---|
| | covered by the endpoint)<br>**Output:** List with service endpoints |
| **Semantic Service Browsing Interface** | |
| **getCategories** | **Input:** None<br>**Output:** Returns the hierarchical concepts structure. |
| **getTableOfContents** | **Input:** modeltype (an array with model types, that restrict the table of contents)<br>Output: returns the model directory tree containing the models that match the model type filter. |
| **getServicesByCategory** | **Input:** concept (a concept)<br>**Output:** a list of services that match the concept |
| **getServicesByActivity** | **Input:** (1) businessProcess (the name of the process ) and (2) activity (the name of the activity)<br>**Output:** List with the services that are mapped to this activity |
| **getServicesByOutput** | **Input:** outputName (the name of the output)<br>**Output:** List of services that produce the needed output |
| **getModelImage** | **Input:** (1) modelid (the id of the model) and (2) scale (the scale in percent ( 0 – 100))<br>**Output:** Byte array containing the image source in png format<br>This method should be used together with the getModelImageMap method, to allow browsing through the modelling objects. |
| **getModelImageMap** | **Input:** (1) modelid (the id of the model) and (2) scale (the scale in percent ( 0 – 100) )<br>**Output:** XML stream containing the ids and the coordinates of the objects |
| **Semantic Service Design Admin Interface** | |
| **createTopicMap** | **Input:** topicMapName (the name of the topic map to be created)<br>**Output:** Boolean value indicating success |
| **removeTopicMap** | **Input:** topicMapName (the name of the topic map to be removed)<br>**Output:** Boolean value indicating success |
| **createCategory** | **Input:** (1) topicMapName (the name of the topic map, in which a category will be created) and (2) categoryName (the |

| | |
|---|---|
| | name of the category that will be created)<br>**Output**: Boolean value indicating success |
| **removeCategory** | **Input:** (1) topicMapName (the name of the topic map, in which a category will be created) and (2) categoryName (the name of the category that will be created)<br>**Output:** Boolean value indicating success |
| **createSLAContractPool** | **Input:** SLAContractPoolName (the name of the model to be created)<br>**Output:** Boolean value indicating success |
| **removeSLAContractPool** | **Input:** SLAContractPoolName (the name of the model to be created)<br>**Output:** Boolean value indicating success |
| **createSLA** | **Input:** (1) slaContractPoolName (the name of the model where the slaContract can be found) and (2) slaName (the name of the slaContract )<br>**Output:** Boolean value indicating success |
| **removeSLA** | **Input:** (1) slaContractPoolName (the name of the model where the slaContract can be found) and slaName (the name of the slaContract)<br>**Output:** Boolean value indicating success |
| **createSLAParameter** | **Input:** (1) slaContractPoolName (the name of the model where the slaContract can be found), (2) slaName (the name of the slaContract), (3) slaParameterName (the parameter to be set and created) and (4) slaParameterValue (the value to be assigned)<br>**Output:** Boolean value indicating success |
| **setSLAParameterValue** | **Input:** (1) slaContractPoolName (the name of the model where the slaContract can be found), (2) slaName (the name of the slaContract), (3) slaParameterName (the parameter to be set) and (4) slaParameterValue (the value to be assigned)<br>**Output:** Boolean value indicating success |
| **removeSLAParameter** | **Input:** (1) slaContractPoolName (the name of the model where the slaContract can be found), (2) slaName (the name of the slaContract) and (3) slaParameterName (the parameter to be removed)<br>**Output:** Boolean value indicating success |
| **createServiceDeploymentModel** | **Input:** modelName (the name of the model to be created)<br>**Output:** Boolean value indicating success |

| | |
|---|---|
| **removeServiceDeploymentModel** | **Input:** modelName (the name of the model to be removed)<br><br>**Output:** Boolean value indicating success |
| **createRTE** | **Input:** (1) modelName (the name of the model where the runtime-environment will be stored) and (2) rteName: the name of the runtime-environment to be stored<br><br>**Output:** Boolean value indicating success |
| **removeRTE** | **Input:** (1) modelName (the name of the model where the runtime-environment is stored) and (2) rteName (the name of the runtime-environment to be removed)<br><br>**Output:** Boolean value indicating success |
| **registerEndPoint** | **Input:** (1) serviceModelName (the name of the model where the service can be found), (2) serviceName (the name of the service that will be offered at this endpoint). (3) serviceDeploymentModelName (the name of the model where the endpoint will be is stored), (4) rteName (the name of the runtime-environment), (5) slaContractPoolName (the name of the model where the slaContract can be found), (6) slaName (the name of the slaContract valid for the endpoint) and (7) endPointUrl ( the url of th endpointy to be deleted)<br><br>**Output:** Boolean value indicating success |
| **removeEndPoint** | **Input:** (1) serviceDeploymentModelName (the name of the model where the runtime-environment is stored), (2) rteName (the name of the runtime-environment) and (3) endPointUrl (the url of the endpoint to be deleted)<br><br>**Output:** Boolean value indicating success |
| **saveService** | **Input:** (1) service (an array of services to be saved), (2) categories (the categories that correspond to the services), (3) datatypes (datatypes of the service) and (4) overwrite (a boolean value indicating if a existing service should be overwritten)<br><br>**Output:** Boolean value indicating success |
| **removeService** | **Input:** (1) serviceModelName (the name of the service model, where the service is stored) and (2) serviceName (the name of the service to be removed)<br><br>**Output:** Boolean value indicating success |
| **removeServiceModel** | **Input:** serviceModelName (the name of the service to be removed)<br><br>**Output:** Boolean value indicating success |
| **getEndpointsByService** | **Input:** serviceName (the name of service the endpoints of which should be found) |

| | |
|---|---|
| | **Output:** An array of Strings that contain the address of the endpoints |
| **listRTE** | **Input:** None<br>**Output:** An array of  ServiceDeploymentModels |
| **listSLA** | **Input:** None<br>**Output:** An array of the SLAPools that are stored in the models. The SLAPools itself contain the SLA-contracts. |
| **listCategories** | **Input:** None<br>**Output:** Categories of the topic maps in form of an array |
| **Semantic Service Import/Export Interface** | |
| **exportWSDL** | **Input:** serviceName (the name of the service )<br>**Ouput:** WSDL stream containing the service definition. The WSDL does not include the service endpoint, because the service can be provided by more the one service provider with different SLA. A list of endpoints can be discovered by the discovery service. |
| **importWSDL** | **Input:** (1)wsdlSource (the wsdl), (2) baseUrl (is needed when the wsdl file references other wsdl files) and (3) overwrite (boolean value that indicates if the existing services should be overwritten)<br>**Output:** Boolean value indicating the success |
| **exportBPEL** | **Input:** bpelName (the name of the BPEL workflow)<br>**Output:** BPEL stream |
| **importBPEL** | **Input:** (1) bpelSource (the BPEL definition that should be imported) and (2) overwrite (boolean value that indicates if the existing workflow should be overwritten).<br>**Output:** Boolean value indicating the success |
| **exportOWL** | **Input:** ontologyName (the name of the ontology that should be exported)<br>**Output:** OWL stream |
| **importOWL** | **Input:** (1) owlSource (the owl source that should be imported) and (2) overwrite (a boolean value that indicates if the existing workflow)<br>**Output:** Boolean value indicating the success |
| **exportOWLS** | **Input:** workflowName (the name of the workflow)<br>**Output:** OWSL stream |
| **importOWLS** | **Input:** (1) owlsSource (the owl source that should be |

| | |
|---|---|
| | imported) and (2) overwrite (a boolean value that indicates if the existing workflow)<br>**Output:** Boolean value indicating the success |
| **Data Access Object Interface** | |
| **getTableOfContents** | **Input:** None<br>**Output:** XML stream containing the model directory structure |
| **getModelXml** | **Input:** modelid (the id of the model)<br>**Output:** XML stream containing all modelling object with all attributes and values. |
| **getModelPicture** | **Input:** (1) modelid (the id of the model) and (2) scale (the scale in percent (0 – 100))<br>**Output:** Byte array with the model picture source in png format |
| **getImageMap** | **Input:** (1) modelid (the id of the model) and (2) scale (the scale in percent (0 – 100))<br>**Output:** XML stream containing the coordinates for every object and relation in the model |
| **createModelDir** | **Input:** (1) name (the name of the model directory) and (2) parentId (the id of the parent directory, null means create in the root model directory)<br>**Output:** Boolean value indicating the success |
| **saveModel** | **Input:** (1) name (the name of the model), (2) dirId (the id of the directory) and (3) adl (the model adl (ADONIS proprietary format))<br>**Output:** the id of the saved model or null if the save fails |
| **renameModel** | **Input:** (1) modelId (the id of the model) and (2) name (the new name)<br>**Output:** Boolean value indicating the success |
| **renameDirectory** | **Input:** (1) directoryId (the id of the directory) and (2) name (the new name)<br>**Output:** Boolean value indicating the success |
| **deleteModel** | **Input:** modelId (the id of the model)<br>**Output:** Boolean value indicating the success |
| **deleteDirectory** | **Input:** directoryId (the id of the directory)<br>**Output:** Boolean value indicating the success |