D4.3.1

Architecture of the Infrastructure Services Layer V1

WP 4.3 Grid Infrastructure Services Layer

Dissemination Level: Public

Lead Editor: Antonis Litke, ICCS/NTUA

30/07/2005

Status: Final

SIXTH FRAMEWORK PROGRAMME

PRIORITY IST-2002-2.3.1.18

Grid for complex problem solving

Proposal/Contract no.: 004293

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

a. **"Collective Work"** means a work, such as a periodical issue, anthology or encyclopedia, in which the Work in its entirety in unmodified form, along with a number of other contributions, constituting separate and independent works in themselves, are assembled into a collective whole. A work that constitutes a Collective Work will not be considered a Derivative Work (as defined below) for the purposes of this License.
b. **"Derivative Work"** means a work based upon the Work or upon the Work and other pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which the Work may be recast, transformed, or adapted, except that a work that constitutes a Collective Work will not be considered a Derivative Work for the purpose of this License. For the avoidance of doubt, where the Work is a musical composition or sound recording, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered a Derivative Work for the purpose of this License.
c. **"Licensor"** means the individual or entity that offers the Work under the terms of this License.
d. **"Original Author"** means the individual or entity who created the Work.
e. **"Work"** means the copyrightable work of authorship offered under the terms of this License.
f. **"You"** means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

2. Fair Use Rights. Nothing in this license is intended to reduce, limit, or restrict any rights arising from fair use, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

a. to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works;

b. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works;

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats, but otherwise you have no rights to make Derivative Works. All rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights set forth in Sections 4(d) and 4(e).

4. Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

a. You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any reference to such Licensor or the Original Author, as requested.
b. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.
c. If you distribute, publicly display, publicly perform, or publicly digitally perform the Work, You must keep intact all copyright notices for the Work and give the Original Author credit reasonable to the medium or means You are utilizing by conveying the name (or pseudonym if applicable) of the Original Author if supplied; the title of the Work if supplied; and to the extent reasonably practicable, the Uniform Resource Identifier, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work. Such credit may be implemented in any reasonable manner; provided, however, that in the case of a Collective Work, at a minimum such credit will appear where any other comparable authorship credit appears and in a manner at least as prominent as such other comparable authorship credit.
d. For the avoidance of doubt, where the Work is a musical composition:
    i. **Performance Royalties Under Blanket Licenses**. Licensor reserves the exclusive right to collect, whether individually or via a performance rights society (e.g. ASCAP, BMI, SESAC), royalties for the public performance or public digital

performance (e.g. webcast) of the Work if that performance is primarily intended for or directed toward commercial advantage or private monetary compensation.

ii. **Mechanical Rights and Statutory Royalties**. Licensor reserves the exclusive right to collect, whether individually or via a music rights agency or designated agent (e.g. Harry Fox Agency), royalties for any phonorecord You create from the Work ("cover version") and distribute, subject to the compulsory license created by 17 USC Section 115 of the US Copyright Act (or the equivalent in other jurisdictions), if Your distribution of such cover version is primarily intended for or directed toward commercial advantage or private monetary compensation.

e. **Webcasting Rights and Statutory Royalties.** For the avoidance of doubt, where the Work is a sound recording, Licensor reserves the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions), if Your public digital performance is primarily intended for or directed toward commercial advantage or private monetary compensation.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Collective Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted

under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

a. Each time You distribute or publicly digitally perform the Work or a Collective Work, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.
b. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
c. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.
d. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

Context

Activity **<4>**	Detailed Architecture, Design & Implementation
WP **<4.3>**	Grid Infrastructure Services Layer Architecture, Design & Implementation
Dependencies	This deliverables uses specifically input of the D3.1.1 Overall Architecture v1, in order to produce an internal architecture in line with the overall one. It will be the basis for D4.3.2 Prototype Implementation of the Infrastructure Services Layer deliverable

Contributors:

Contributors (in alphabetical Order):	Reviewers:[1]
Annalisa Terracina (DATAMAT)	Ignaz Mueller (USTUTT-HLRS)
Anniello Rovezzi (CRMPA)	Brian Ritchie (CCLRC)
Antonis Litke (NTUA)	Ruth del Campo (USTUTT-RUS)
Daniele Bocci (CRMPA)	Bastian Koller (USTUTT-HLRS)
Dimitris Skoutas (NTUA)	Robert Piotter (USTUTT-HLRS)
Dimos Kyriazis (NTUA)	
Dimitris Halkos (NTUA)	
Francesco Verdino (CRMPA)	
Francesco Furfari (CRMPA)	
Giulio Negro (CRMPA)	
Giuseppe Cirillo (CRMPA)	
Giuseppe Laria (CRMPA)	
Ignaz Mueller (USTUTT-HLRS)	
Isabel Alonso Mediavilla (TID)	
Jordi Albacar (ATOS Origin)	
Josep Martrat (ATOS Origin)	
Jürgen Jähnert (USTUTT-RUS)	
María Aránzazu Toro Escudero (TID)	
Massimo Serrano (CRMPA)	
Raul Bori (ATOS)	
Sotiris Chatzis (NTUA)	

Approved by: Stefan Wesner, University of Stuttgart, Germany, as IP Manager

[1] Due the nature of this document being a central document of the project it was not possible to determine completely independent reviewers. The approach chosen was to assign sections not written by the authors themselves to be reviewed and consolidate the results.

Version	Date	Authors	Sections Affected
0.1	2/2/2005	A.Litke	Table of Contents
0.2	27/4/2005	WP4.3 partners	First contribution compiled into the document. Sections affected: EMS, Data Management, SLA, Security, Monitoring
0.3	27/5/2005	WP4.3 partners	Refinements on previous sections. Add of Policy Manager input.
0.4	21/6/2005	WP4.3 partners	Refinements/updates/addendums on previous sections.
0.5	4/7/2005	WP4.3 partners	Refinements/updates/addendums on previous sections. Provision of Metering component description. Input on introductory sections (OGSA description, WSRF, Layered architecture of WP4.3, etc)
0.8 (prefinal)	13/7/2005	WP4.3 partners	Circulation for internal review.
1.0 (final)	30/7/2005	WP4.3 partners	Final version, provision of last missing sections. Refinements on previous sections Internal reviewers comments incorporated.

Table of Contents

List of Figures

List of Tables

Abbreviations

A table with used abbreviations is strongly recommended

Akogrimo	Access To Knowledge through the Grid in a Mobile World
A4C	Authentication, Authorization, Accounting, Auditing and Charging
AR	Advanced reservation
CPU	Central Processing Unit
CSG	Candidate Set Generator
EMS	Execution Management Services
EPR	Endpoint Reference
EPS	Execution Planning Service
JM	Job Manager
MUWS	Management Using Web Services
OASIS	Organization for the Advancement of Structured Information Standards
OpVOBroker	Operative VO Broker
OGSA	Open Grid Services Architecture
SC	Service Consumer
SP	Service Provider
SLA	Service Level Agreement
SOAP	Simple Object Access Protocol
QoS	Quality of Service
VO	Virtual Organization
WSDM	Web Services Distributed Management
WSRF	Web Services Resource Framework
WSDL	Web Service Definition Language
XML	Extensible Markup Language

1. Summary

This document describes the architecture that will be adopted within the WP4.3 Grid Infrastructure Services Layer of the Akogrimo project. It has a relation to the overall architecture already presented in Deliverable D3.1.1 but its aim is to provide a more detailed view of the architecture of the Grid infrastructure itself.

The document gives an overview of the Open Grid Services Architecture (OGSA) which is the basis upon which the architecture is built. In the sequel it presents the Web Services Resource Framework (WSRF) and some implementations that are met in the current trends of the appropriate players/stakeholders.

The document includes an overview of the requirements that need to be met by an OGSA based Grid in the project and shows in a layered approach the positioning of the WP4.3 components within the Akogrimo overall architecture.

The components that have been identified as building blocks of the WP4.3 architecture are the Execution Management Services component, Data Management, Monitoring component, Service Level Agreement Enforcement, Policy Manager, Metering component and the Security framework that will be applied. The discussion on the modules concerns the technologies that are applied as well as the interfaces they use to interact with each other.

Finally, the Annex presents several cases and examples in order to clarify the functionality and the interactions of these modules.

2. Introduction

2.1. Open Grid Services Architecture

2.1.1. Overview

One of the focus points of the Akogrimo project is to utilise the concept expressed by the *Open Grid Services Architecture (OGSA)* [1] in order to provide a Grid middleware layer as the basic Infrastructure services layer on top of the Network Middleware layer, which will provide the core Grid functionality. As defined in the OGSA draft specification, 'Grid systems and applications aim to integrate, virtualize, and manage resources and services within distributed, heterogeneous, dynamic Virtual Organizations'. However, such a goal requires an abstraction of the computing systems themselves so that computers, application services, data, and other resources (physical and logical) can be accessed as and when required, regardless of physical location and administrative domains. For this reason the OGSA Working Group (OGSA-WG) aims to standardization these key aspects in order to allow mainly for interoperability, portability, reusability and security in the Grid systems.

The OGSA Working Group defines, within a service-oriented architecture, a set of core capabilities and behaviours that address key concerns in Grid systems. These concerns include issues such as: service discovery and management, identity establishment, membership management within virtual organizations, service collections organization and monitoring, policy expression and management, and others.

This informational document (GWD-I), a product of the Global Grid Forum's OGSA working group, defines OGSA version 1. The OGSA working group is committed to releasing a recommendation (GWD-R) version of this document in the future.

2.1.2. Requirements

The analysis of the various use cases that have been made within the OGSA working group have identified a set of requirements the most important of which, from Akogrimo's viewpoint, are presented in the sequel:

1. Dynamic and Heterogeneous Environment Support

OGSA must enable interoperability between these diverse, heterogeneous, and distributed resources and services as well as reduce the complexity of administrating heterogeneous systems. Requirements for heterogeneous systems support include resource virtualization, common management capabilities, resource discovery and query, standard protocols etc.

2. Resource Sharing Across Organizations

Resource sharing requirements include global name space, metadata services, site autonomy, mechanisms and standard schemas for collecting and exchanging usage information across organizations, e.g., for the purpose of accounting, billing, etc.

3. Optimization

Optimization applies to both suppliers and consumers of resources and services. One common case of supply-side optimization is resource optimization. Resource utilization can be improved by flexible resource allocation policies, including advance reservation of resources with a bounded time period, the pooling of backup resources, etc.

4. Quality of Service (QoS) Assurance

QoS assurance requirements includes service level agreement and attainment mechanisms for monitoring services quality, estimating resource utilization, and planning for and adjusting resource usage.

5. Job Execution

OGSA must provide manageability for execution of user-defined tasks (jobs) throughout their lifetime. Functionalities such as scheduling, provisioning, job control and exception handling of jobs must be supported, even when the job is distributed over a great number of heterogeneous resources. Job execution requirements include support for various job types, job management, scheduling and resource provisioning.

6. Data Services

Efficient access to and movement of huge quantities of data is required in more and more fields of science and technology. In addition, data sharing is important, for example enabling access to information stored in databases that are managed and administered independently. In business areas, archiving of data and data management are essential requirements. Data services requirements include data access, consistency, persistency, integration and location management.

7. Security

Safe administration requires controlling access to services through robust security protocols and according to provided security policy. For example, obtaining application programs and deploying them into a Grid system may require authentication and authorization. Also sharing of resources by users requires some form of isolation mechanism. In addition, standard, secure mechanisms are required which can be deployed to protect Grid systems while supporting safe resource-sharing across administrative domains.

2.1.3. Capabilities

OGSA is intended to facilitate the seamless use and management of distributed, heterogeneous resources. The term "distributed" could refer to a spectrum from geographically-contiguous resources linked to each other by some connection fabric to global, multi-domain, loosely- and intermittently-connected resources. The term "Resources" could refer to any artefact, entity or knowledge required to complete an operation in the system. The utility provided by such an infrastructure is realized as a set of capabilities. Figure 1 shows the logical, abstract, semi-layered representation of some of these capabilities. Three major logical and abstract layers are envisioned in this representation.

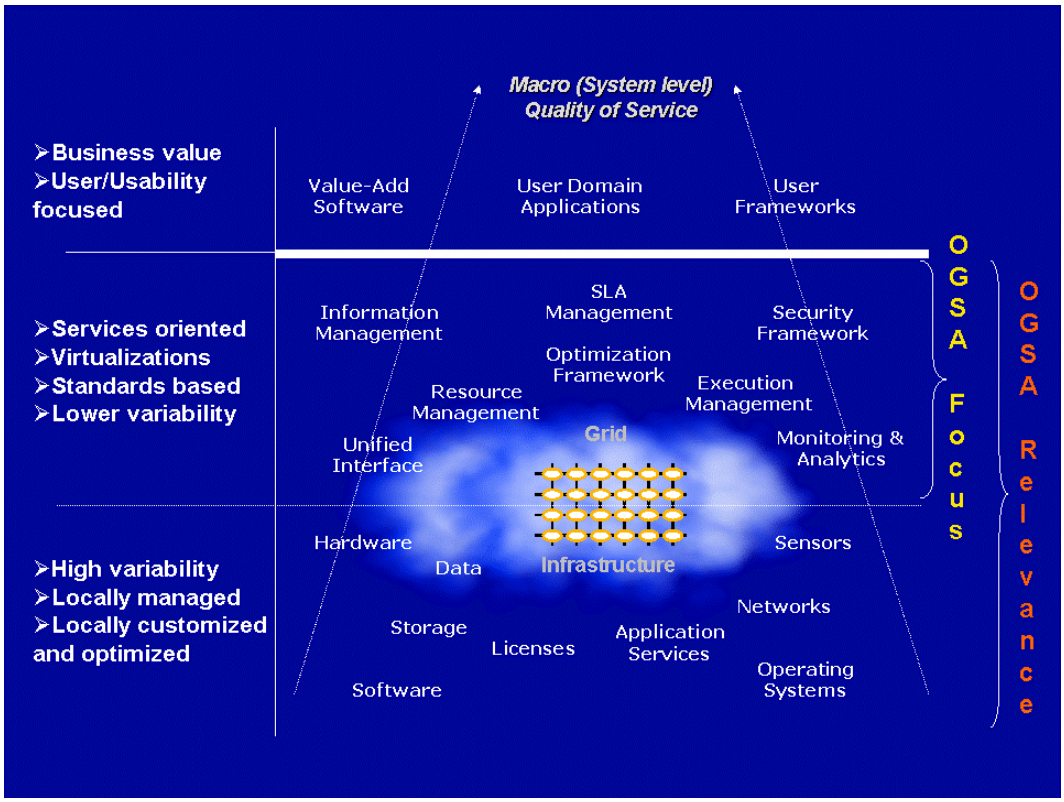

Figure 1: Conceptual view of Grid infrastructures

The capabilities and resources depicted in the diagram are not an exhaustive list, and have been kept minimal for clarity. The first (bottom) layer of the representation in Figure 1 depicts the base resources. Base resources are those resources that are supported by some underlying entities/artefacts that may be physical or logical, and that have relevance outside of the OGSA context. Examples of such physical entities include CPUs, memory, disks etc., and examples of such logical artefacts include licenses, contents, OS processes etc. These resources are usually locally owned and managed, but without excluding any remote share of them.

The second (middle) layer represents a higher level of virtualization and logical abstraction. The virtualization and abstraction are directed toward defining a wide variety of capabilities that are relevant to OGSA Grids. These capabilities can be utilized individually or composed as appropriate to provide the infrastructure required to support higher-level applications or "user" domain processes.

There is no clear boundary between the middle and bottom layers in the logical diagram. This is because OGSA has to comprehend the forms and types of resources, and use this understanding to frame the discussion of capabilities in the middle layer. In addition these resources (i.e., virtualizations) will be driven, used and/or managed in realizing many of the capabilities in middle layer. This close relationship in the OGSA discussion is indicated by the finer (thin line) demarcation of the bottom and middle layers in Figure 1.

In the third (top) layer in the logical representation are the applications and other entities that use the OGSA capabilities to realize user- and domain-oriented functions and processes, such as business processes.

2.1.4. OGSA framework

OGSA realizes the logical middle layer in Figure 1 in terms of services, the interfaces that these services expose, the individual and collective state of resources belonging to these services, and

the interaction between these services within a service-oriented architecture (SOA). The OGSA services framework is shown in Figure 2. These are built on Web service standards, with semantics, additions, extensions and modifications that are relevant to Grids.

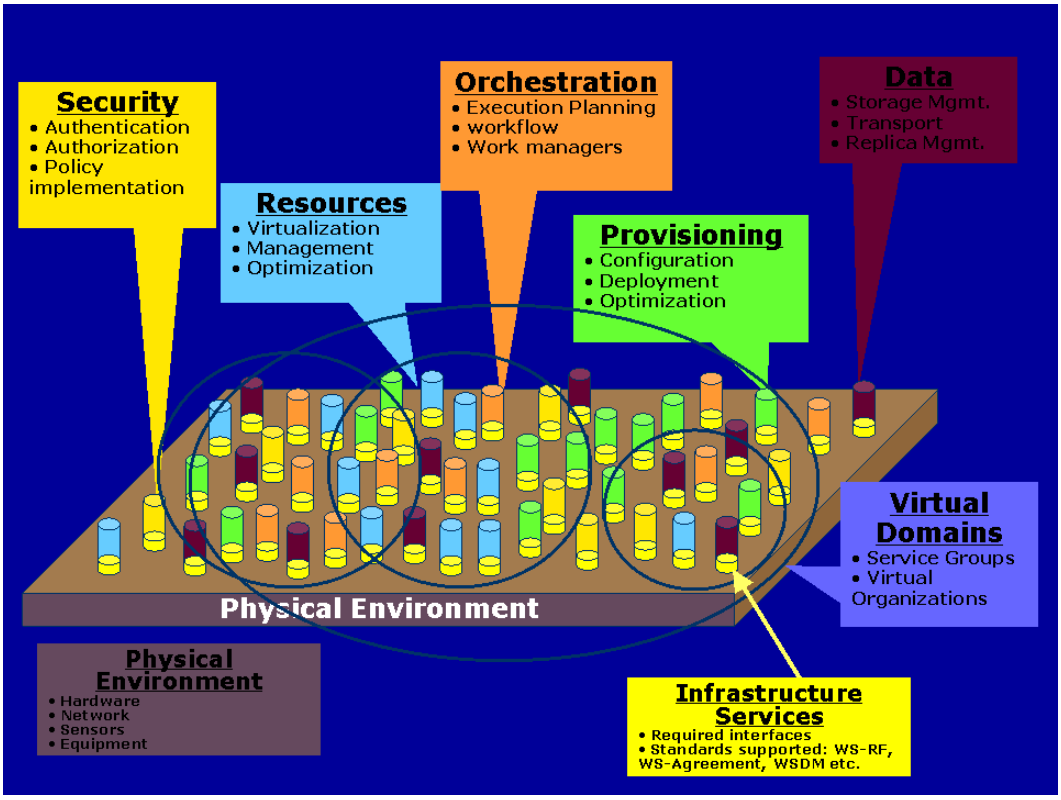

Figure 2: OGSA framework

Services are loosely coupled peers that, either singly or as part of an interacting group of services, realize the capabilities of OGSA through implementation, composition, or interaction with other services (see Figure 2). For example: to realize the "orchestration" capability a group of services might be structured such that one set of services in the group drives the orchestration (i.e., acts as the "orchestrator"), while other services in the group provide the interfaces and mechanisms to be orchestrated (i.e., be the "orchestratees"). A specific service may implement and/or participate in multiple collections and interactions to realize different capabilities. The services may be part of, or participate in, virtual collections called virtual domains (see Figure 2) to realize a capability, as in service groups, or to share a collective context or manageability framework, as in virtual organizations.

2.1.5. The OGSA services

Execution Management Services

Execution Management Services (OGSA-EMS) are concerned with the problems of instantiating and managing tasks. Within OGSA-EMS a task refers to a single unit of work to be managed, for example a legacy batch job, a database server, a servlet running in a Java application server container, etc.

Data Services

OGSA data services are concerned with the movement, access and update of data resources.

Resource Management Services

Resource management performs several forms of management on resources in a Grid. In an OGSA Grid there are three types of management that involve resources: (i) Management of the resources themselves (e.g., rebooting a host, or setting VLANs on a network switch), (ii) Management of the resources on Grid (e.g., resource reservation, monitoring and control), (iii) Management of the OGSA infrastructure, which is itself composed of resources (e.g., monitoring a registry service).

Security Services

OGSA security services exist to facilitate the enforcement of security-related policies within a (virtual) organization. The security policy enforcement is there to ensure that objectives can be met without comprising the security aspects (like integrity, confidentiality, etc.)

Self-Management Services

Self-management was conceived as a way to help reduce the cost and complexity of owning and operating an IT infrastructure. In such an environment, system components—from hardware such as desktop computers and mainframes to software such as operating systems and business applications—are self-configuring, self-healing and self-optimizing.

Information Services

The ability to efficiently access and manipulate information about applications, resources and services in the Grid environment is an important OGSA capability. The term "information" refers to: dynamic data or events used for status monitoring; relatively static data used for discovery; and any data that is logged. In practice, an information service needs to support a variety of QoS requirements for reliability, security, and performance.

2.2. Web Services Resource Framework

2.2.1. Overview

From the description above it can be easily inferred that the OGSA demands some form of distributed middleware on which to base its architecture. The technology chosen as the most proper one by the Grid community is the Web Services technology. However, although the Web Services Architecture was certainly the best option, it did not meet one of OGSA's most important requirements, as described above: the underlying middleware had to be stateful. Unfortunately, although Web services in theory can be either stateless or stateful, they are usually stateless and there is no standard way of making them stateful.

The Web Services Resource Framework (WSRF) [2] solves this problem in the following way: it keeps the Web service and the state information completely separate. Instead of putting the state in the Web service (thus making it stateful, which is generally regarded as a bad idea) it is kept in a separate entity called a resource, which stores all the state information. Each resource has a unique key, so whenever we want a stateful interaction with a Web service we simply have to instruct the Web service to use a particular resource.

The Web Services Resource Framework defines a family of specifications for accessing stateful resources using Web services. It includes the WS-ResourceProperties, WS-ResourceLifetime, WS-BaseFaults, and WS-ServiceGroup specifications. The motivation for these new

specifications is that while Web service implementations typically do not maintain state information during their interactions, their interfaces must frequently allow for the manipulation of state, that is, data values that persist across and evolve as a result of Web service interactions. For example, an online airline reservation system must maintain state concerning flight status, reservations made by specific customers, and about the system itself: its current location, load, and performance. Web service interfaces that allow requestors to query flight status, make reservations, change reservation status, and manage the reservation system must necessarily provide access to this state. In the Web Services Resource Framework we model state as stateful resources and codify the relationship between Web services in terms of an implied resource pattern.

For example, let us consider an accumulator example. As shown in the Figure 3, our Web service could have three different resources (A, B, C) to choose from. If we want the integer value to be "remembered" from invocation to invocation, the client simply has to specify that he wants a method invoked with a certain resource.

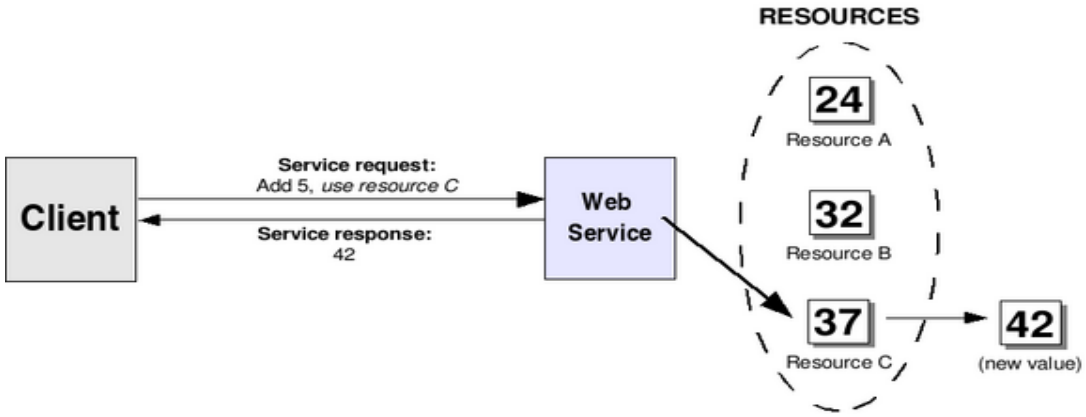

Figure 3: The resource approach to statefulness

In the figure the client requests the add operation to be invoked with resource C. When the Web service receives the add request, it will make sure to retrieve resource C so that add is actually performed on that resource. The resources themselves can be stored in memory, on secondary storage, or even in a database. Also, notice how a Web service can have access to more than one resource.

Of course, resources can come in different shapes and sizes. A resource can contain multiple values (not just a simple integer value, as shown in the previous figure). For example, our resources could represent files:

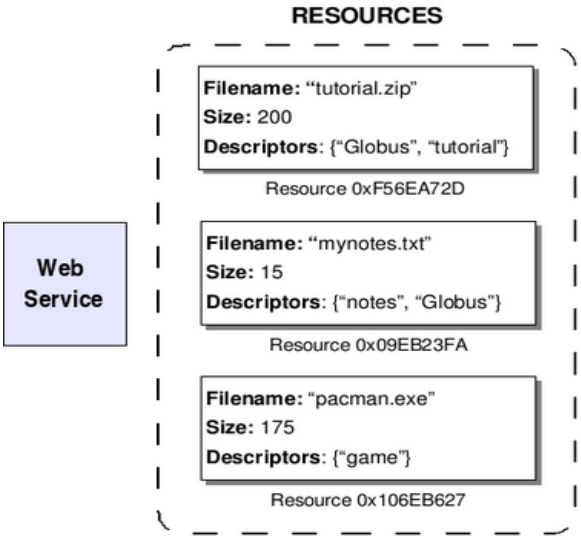

Figure 4: A Web Service with several resources. Each resource represents a file

A URI might be enough to address the Web service, but how do we specify the resource on top of that? The preferred way to do it is to use a relatively new specification called WS-Addressing which provides a more versatile way of addressing Web Services (when compared to plain URIs).

A pairing of a Web service with a resource is called a WS-Resource. The address of a particular WS-Resource is called an endpoint reference (in WS-Addressing terminology).

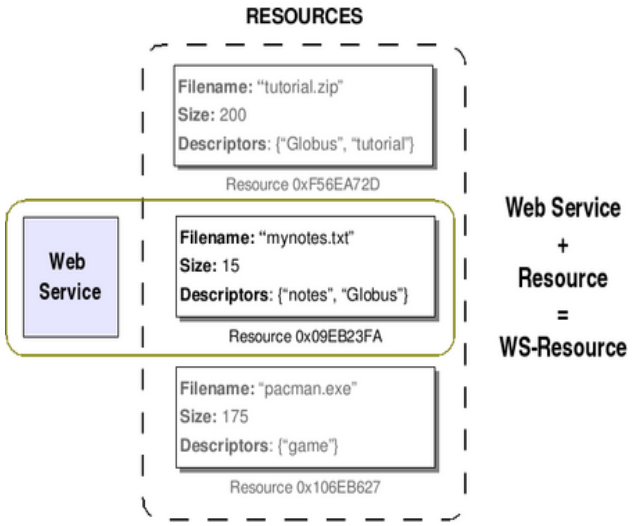

Figure 5: WS-Resource

In the following we will present the basic characteristics of the WSRF - specifications.

The **WS-ResourceProperties** specification defines how the data associated with a stateful resource can be queried and changed using Web services technologies. This allows a standard means by which data associated with a WS-Resource can be accessed by clients. The declaration of the WS-Resource's properties represents a projection of or a view on the WS-Resource's state. This projection represents an implied resource type which serves to define a basis for access to the resource properties through Web service interfaces.

WS-ResourceLifetime defines two ways of destroying a WS-Resource: immediate and scheduled. This allows designers flexibility in designing how their Web services applications can clean up resources that are no longer needed.

WS-BaseFaults defines an XML Schema type for a base fault, along with rules for how this fault type is used by Web services. A designer of a Web services application often uses interfaces defined by others. Managing faults in such an application is more difficult when each interface uses a different convention for representing common information in fault messages. Support for problem determination and specifying Web services fault messages in a common way can enhance fault management. When the information available in faults from various interfaces is consistent, it is easier for requestors to understand faults. It is also more likely that common tooling can be created to assist in the handling of faults.

WS-ServiceGroup defines a means by which Web services and WS-Resources can be aggregated or grouped together for a domain specific purpose. In order for requestors to form meaningful queries against the contents of the ServiceGroup, membership in the group must be constrained in some fashion. The constraints for membership are expressed by intention using a classification mechanism. Further, the members of each intention must share a common set of information over which queries can be expressed.

Finally, there are also some specifications not included but closely related to WSRF. These are WS – Notification and WS – Addressing.

WS – Notification allows a Web service to be configured as a notification producer, and certain clients to be notification consumers (or subscribers). This means that if a change occurs in the Web service (or, more specifically, in one of the WS-Resources), that change is notified to all the subscribers. (Not all changes are notified, only those chosen by the Web services programmer).

The **WS-Addressing** specification provides a mechanism for addressing Web services which is more versatile than plain URIs. In particular, we can use WS-Addressing to address a Web service + resource pair (a WS-Resource).

2.2.2. Implementations

2.2.2.1. The Globus Toolkit

The open source software Globus Toolkit [3], developed by *The Globus Alliance*, is a fundamental enabling technology for the Grid, letting people share computing power, databases, and other tools securely online across corporate, institutional, and geographic boundaries without sacrificing local autonomy. The toolkit includes software services and libraries for resource monitoring, discovery, and management, plus security and file management. In addition to being a central part of science and engineering projects the Globus Toolkit is a substrate on which leading IT companies are building significant commercial Grid products.

The toolkit includes software for security, information infrastructure, resource management, data management, communication, fault detection, and portability. It is packaged as a set of components that can be used either independently or together to develop applications. Every organization has unique modes of operation, and collaboration between multiple organizations is hindered by incompatibility of resources such as data archives, computers, and networks. The Globus Toolkit was conceived to remove obstacles that prevent seamless collaboration. Its core services, interfaces and protocols allow users to access remote resources as if they were located within their own machine room while simultaneously preserving local control over who can use resources and when.

The Globus Toolkit has grown through an open-source strategy in a similar manner to the Linux operating system, and distinct from proprietary attempts at resource-sharing software. This

encourages broader, more rapid adoption and leads to greater technical innovation, as the open-source community provides continual enhancements to the product.

This toolkit, first and foremost, includes a number of high-level services that we can use to build Grid applications. These services meet most of the abstract requirements set forth in OGSA. In other words, the Globus Toolkit includes a resource monitoring and discovery service, a job submission infrastructure, a security infrastructure, and data management services. The Globus Toolkit is a realization of the OGSA requirements and is almost a *de facto* standard for the Grid. Indeed, the Globus Toolkit 4 includes a complete implementation of the WSRF specification.

2.2.2.2. The WSRF.NET

WSRF.NET [4] is an implementation of the WSRF specifications on the Microsoft .NET platform. WSRF.NET consists of a set of libraries and tools that allows web services to be transformed into WSRF-compliant web services. WSRF.NET uses the IIS/ASP.NET architecture for web services; that is WSRF.NET services are web services. To use WSRF.NET, a service author first creates a web service using VS.NET just as they would any other web service. Then the service logic is annotated with attributes that the WSRF.NET tools recognize. WSRF.NET tools transform the author's compiled web service into a WSRF-compliant web service consistent with the meta–data given by the service author in the attributes.

3. Akogrimo Grid Infrastructure Services Layer

3.1. Overview of the WP4.3 layer

3.1.1. Requirements Analysis

The WP4.3 layer involves the Basic Grid Infrastructure Services as they have been indicated in the Description of Work document. Within the framework of Akogrimo this is the layer that focuses on the basic Grid functionality of the overall platform. For this reason we address its objectives in the broader scope of the project. In order to achieve this, an analysis on the requirements that this layer aims to consider has to be elaborated.

The ***Grid*** can be viewed as a *distributed*, *high performance computing* and *data handling infrastructure*, that incorporates geographically- and organizationally-dispersed, heterogeneous resources (computing systems, storage systems, instruments and other real-time data sources, human collaborators, communication systems) and provides common interfaces for all these resources, using standard, open, general-purpose protocols and interfaces. It is also the basis and the enabling technology for pervasive and utility computing due to its ability to be open, highly heterogeneous and scalable. [5]

The ***Mobile Grid*** is a full inheritor of the Grid with the additional feature of supporting mobile users and resources in a *seamless, transparent, secure and efficient way*. It has the ability to deploy underlying ad-hoc networks and provide a self-configuring Grid system of mobile resources (hosts and users) that are connected by wireless links forming arbitrary and unpredictable topologies. [6]

The following table is a checklist regarding typical Grid characteristics that would clarify the difference between a Grid solution and a custom IT solution. It is based on features that have been extracted from the OGSA specification and should help to identify if Akogrimo application requirements imply (or strongly suggest) the use of an OGSA Grid enabled infrastructure.

Requirements	Description
Dynamic and heterogeneous environments support	Grid systems support *large-scale* distributed computing among different environments and platforms (operating systems, networks, application frameworks…). This is achieved by providing the application layer with standard services for resource virtualization, common management capabilities, resource discovery and query.
Resource sharing across different organization and domains	OGSA compliant grids support for standard protocols and schema, global namespace handling, metadata services, site autonomy management. In this way applications can improve their resource sharing in a wide-scale distributed environment.

Common interfaces using standard, open, general-purpose protocols	Grid systems compliant with the OGSA architecture have component services and interfaces based on standard open and general-purpose protocols. This is for achieving modularity, scalability and interoperability in the context of complex and heterogeneous environments.
Coordinated administrative management	Grid applications can use the automation of common administrative operations to avoid human errors and manage very large-scale systems.
Multiple security infrastructure and policy exchange support	Distributed operation across different, independent domains need the support (and the coordination) of multiple security infrastructures. In a grid environment service requestors and providers can exchange security policy information to establish a negotiated security context between them.

Table 1: OGSA related Grid requirements for the Akogrimo project

Within Akogrimo there is also a basic requirement for mobility which influences the requirements analysis for the overall WP4.3 architecture. The mobility issues do not necessarily stress the requirements in a way that would change the primitive functionality of a Grid infrastructure. Rather, they allow a Grid computing environment to mobilise its resources in a seamless and transparent way to the end-user. Moreover, Akogrimo's view of mobile Grids includes environments where end-to-end quality and security is achieved through network-aware Grid infrastructures.

3.1.2. Akogrimo OGSA-based Layered Architecture

The discussion in the previous sections has indicated a set of objectives that an OGSA compliant architecture aims to fulfil. These include:

• Manage resources across distributed heterogeneous platforms.

• Deliver seamless quality of service (QoS). The topology of Grids is often complex. Interaction of grid resources is usually dynamic. It is important that the grid should provide robust, behind-the-scenes services such as authorization, access control, and delegation.

• Provide a common base for autonomic management solutions. A grid can contain many resources, with numerous combinations of configurations, interactions, and changing state and failure modes. Some form of intelligent self regulation and autonomic management of these resources is necessary.

• Define open, published interfaces. OGSA is an open standard managed by the GGF standards body. For interoperability of diverse resources, grids must be built on standard interfaces and protocols.

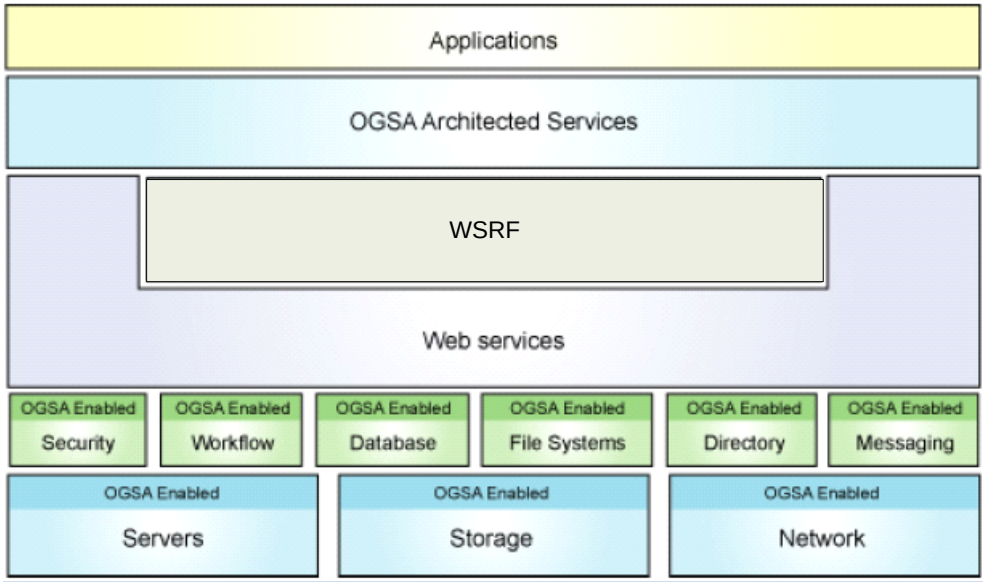

Figure 6: OGSA main architecture

As defined in [7] , the OGSA architecture is comprised of four main layers. They are:

1. Resources: physical resources and logical resources

2.Web services, plus the WSRF that together define grid services

3.OGSA architected services

4.Grid applications

1. Physical and logical resources layer

The concept of resources is central to OGSA and to Grid computing in general. Resources comprise the capabilities of the Grid, and are not limited to processors. Physical resources include servers, storage, and network. Above the physical resources are logical resources. They provide additional function by virtualizing and aggregating the resources in the physical layer. General purpose middleware such as file systems, database managers, directories, and workflow managers provide these abstract services on top of the physical Grid.

2. Web services layer

The second layer in the OGSA architecture is Web services. All Grid resources both logical and physical, are modelled as services. The Web Services Resource Framework (WSRF) defines a family of specifications for accessing stateful resources using Web services. It includes the WS-ResourceProperties, WS-ResourceLifetime, WS-BaseFaults, and WS-ServiceGroup specifications. The motivation for these new specifications is that while Web service implementations typically do not maintain state information during their interactions, their interfaces must frequently allow for the manipulation of state, that is, data values that persist across and evolve as a result of Web service interactions.

3. OGSA architected grid services layer

The Web services layer, together with the WSRF, provide a base infrastructure for the next layer OGSA architected services. The Global Grid Forum is currently working to define many of these architected services in areas like program execution, data services, and core services. Some are already defined, and some implementations have already appeared.

4. Grid applications layer

Over time, as a rich set of grid-architected services continues to be developed, new grid applications that use one or more grid architected services are beginning to appear. These applications comprise the fourth main layer of the OGSA architecture. These applications come from various scientific and business domains and include among others biomedics, construction, engineering, finance, etc.

3.1.3. Positioning in the Akogrimo Architecture – Interaction with the other Layers

The WP4.3 Grid infrastructure layer is placed in the middle of the Akogrimo architecture participating as the "Grid glue" to the functionality of the Akogrimo system.

The basic motivation of this layer is to provide OGSA specific services implemented through the framework of the WSRF. However, since the OGSA architecture is still in a draft specification and moreover aims to address a broad set of issues (sometimes not necessarily in all architectures) the Akogrimo consortium has identified a concrete set of capabilities that it would like to incorporate in the architecture of the system and the Workpackage in particular.

The following figure (Figure 7) gives a conceptual overview of the WP4.3 Grid Infrastructure services layer with respect to the specific services of OGSA that comprise the services considered in the project and its positioning in the layered approach of Akogrimo system.

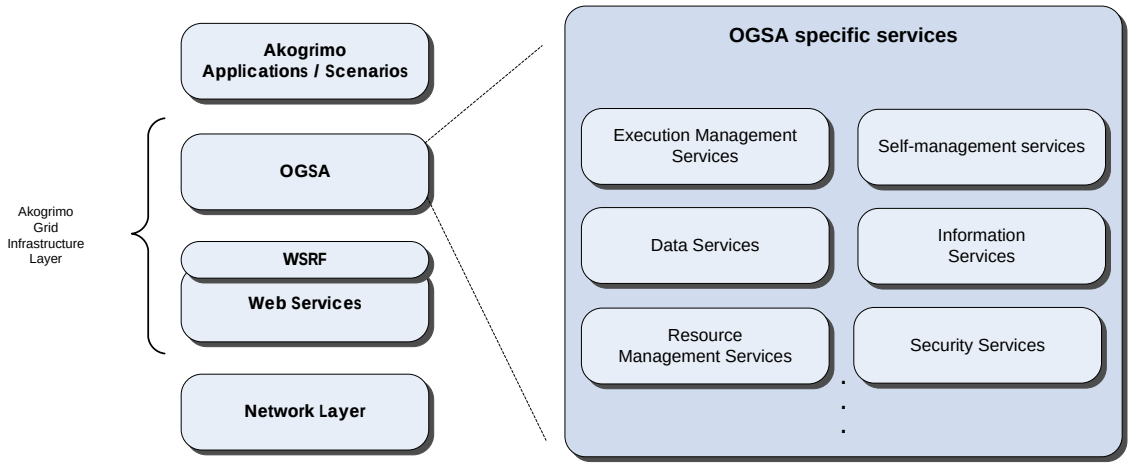

Figure 7: The focus of the Akogrimo Grid Infrastructure layer with respect to OGSA

As far as it concerns the specific objectives of the Akogrimo project and its layers, the WP4.3 Grid infrastructure services layer is positioned as depicted in the Figure 8, interacting with all the other layers.

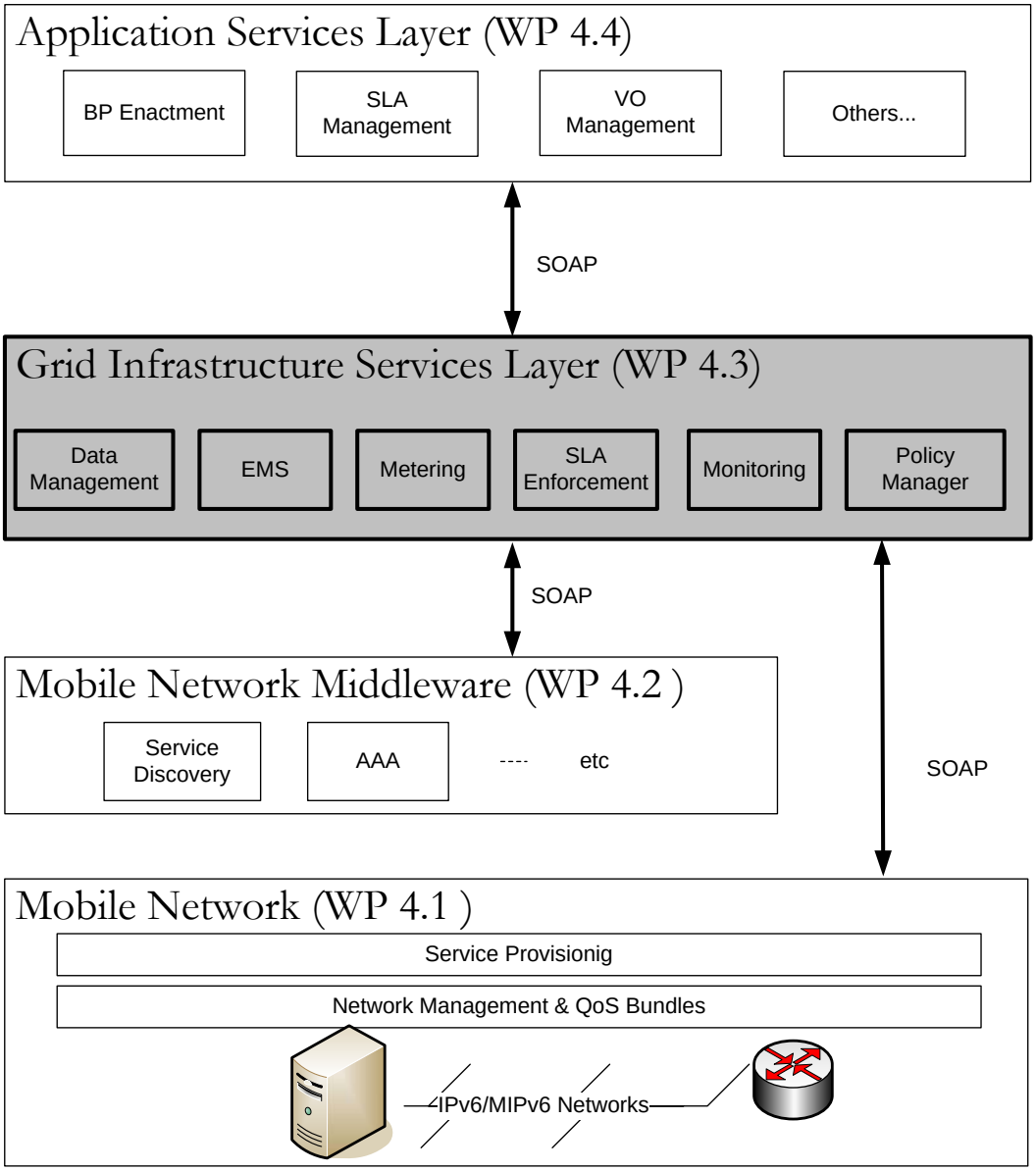

Figure 8: Positioning of WP4.3 components in the overall Akogrimo architecture

The basic role of the WP4.3 layer is to manage the execution of the services on request of the Application Service Layer, aiming to fulfil the Grid requirements, addressing the performance issues (job throughput, efficiency, maximum utilization of the resources) in a transparent way to the user and conforming to the determined Service Level Agreement (SLA). This layer aims to provide services to manage the execution of jobs coming from the Application Service Layer.

More precisely, the Grid Infrastructure Services layer interacts with:

• Application Services layer, to receive the jobs to be executed and to identify the corresponding SLAs and Policies that will regulate and influence this execution.

• Mobile Network Middleware layer (WP4.2), mainly in order to discover services and have access to AAA.

• Mobile Network layer (WP4.1) in order to perform various actions that deal with the network (for instance QoS bundles of services for having advanced reservation of network resources).

The basic functionality that must be supported from the Grid Infrastructure Services layer, as it has been identified by the consortium and is in conjunction with the OGSA draft specification, can be categorized in the following:

Execution Management Services: This category of services comprises all the functionality that is concerned with the problems of instantiating and managing tasks, such as assigning jobs to resources, creating an execution plan, balancing the workload, optimizing the performance, and replicating jobs to provide fault tolerance.

Data Management: This category comprises all the functionality that are concerned with the access to and movement of large data sets, as well as data sharing, replicating and archiving of data etc…

Monitoring: This category comprises the services that are focusing on monitoring and managing of the web services within the layer.

SLA: Services related to the enforcement of the Service Level Agreement contractual terms that especially influence the execution of jobs within the layer.

Metering: Services that are supplementary to the monitoring and accounting services and deal especially with the measurement of resource usage.

Policy management: This category of services comprises the functionality concerned with the management of rules and the policies which apply in the execution of services within the Akogrimo architecture.

Security: This category comprises the services that are concerned with the security issues of the specific layer. It comprises the services that will deal with the confidentiality of the communications and the authorization for execution within the system.

3.2. Execution Management Services

3.2.1. Overview

Execution Management Services (OGSA-EMS) [1] are concerned with the problems of instantiating and managing tasks.

3.2.1.1. Objectives

- Where can a task execute? What are the locations at which it can execute because they satisfy resource restrictions such as memory, CPU and binary type, available libraries, and available licenses? Given the above, what policy restrictions are in place that may further limit the candidate set of execution locations?
- Where should the task execute? Once it is known where the task can execute, the question is where it should execute. Answering this question may involve different selection algorithms that optimize different objective functions or attempt to enforce different policies or service level agreements.
- Prepare the task to execute. Just because a task can execute somewhere does not necessarily mean it can execute there without some setup. Setup could include deployment and configuration of binaries and libraries, staging data, or other operations to prepare the local execution environment to execute the service.

- Manage (monitor, restart, move, etc.). Once the task is started, it must be managed and monitored. What if it fails? Or what if it fails to meet its agreements? Should it be restarted in another location? What about state? Should the state be "checkpointed" periodically? Is the task participating in some sort of fault-detection and recovery scheme?

3.2.1.2. Functional capabilities

- Selects the set of resources that can be used to execute a submitted job.
- Assigns jobs to resources and creates an execution plan, trying to balance the workload, optimize the performance and provide QoS.
- Handles job queues and priorities to meet SLAs or handle crisis situations.
- Replicates jobs to provide fault tolerance.
- Provides advanced resource reservation.
- Manages the job execution (deploy, start, suspend, terminate).

3.2.1.3. Non-functional capabilities

- Provides an API that can be used for the interaction with other components.
- Keeps close co-operation with other critical components of other Workpackages (for instance the Business Process Enactment), since it acts as the heartbeat of the WP4.3 components.
- Performance: the algorithms used for the functionalities mentioned above (scheduling, prioritization, replication, etc.) can be run in real time and do not cause a significant overhead to the overall system performance.
- Scalability: the system can function under heavy workload.
- Modularity: the system is composed of several components and can be easily maintained and/or modified with new algorithms.
- Security: the system interacts only with authenticated and authorized entities.

3.2.1.4. Interaction with other components of the Layer

Deployment Services

EMS should interact with a deployment service if we assume the WSRF model where WS-Resources or Web Services are present. In fact, if the EMS detects that a machine could be the most suitable host for a specific service and this service is not deployed on that machine, then the EMS should be able to deploy on the fly the executable necessary to publish the Web Service on the host. Furthermore, any other configuration action should be executed on the host in order to allow the proper invocation of the deployed service.

Data Management

The EMS interacts with the Data Management component in cases where data services are required. If during the execution of a service there is a need for query on data bases or management of data sets related to execution itself (e.g. storing logging info, transferring information to accounting subsystems, etc.) the EMS interacts with this component for carrying out this job.

Monitoring

In Akogrimo, we can have different kinds of monitoring to be managed:

- Quality of Service to be achieved and to be compliant with the SLA terms
- Faults detection to guarantee some level of fault tolerance
- Monitoring in order to optimize resource usage (e.g. load balancing, …)
- Having knowledge of the execution procedure and the status of the resources

The EMS will be strictly related to the monitoring subsystem at least in two phases:

1. reservation/execution phase: when a new service is going to be executed the EMS has to contact the monitoring subsystem in the reservation phase and in order to start the monitoring of the new service when it is executing.
2. problem detection: in case of problems (QoS violation, fault detections,…) during the service execution

Metering

The EMS should interact with the metering service in different phases. First, the EMS interacts with the metering service when it makes a reservation of resources because there could be a charge for making a reservation. In the second phase EMS interacts with metering service to inform it about the execution start and the parameters (configuration) that are of interest in the specific execution. Since there are many different kinds of services ranked with respect to their complexity and cost efficiency it necessary to identify what kind of parameters are going to be metered (for computationally intensive or data intensive applications etc.)

SLA Enforcement Service

The EMS interacts with the SLA component to receive as input the SLA terms that have to be applied in the execution phase. These terms are the results of the negotiation between a service client (SC), which would like to use a service, and the service provider (SP) who provides it. Furthermore, in order to satisfy a request for services and in order to provide QoS, the EMS makes a "pre-reservation" of the resources for a specific period of time. When the EMS runs a job, it interacts with SLA Enforcement service and Monitoring service to constantly control the job execution, so that possible violations of the agreement described in SLA can be discovered. In the case in which a violation is discovered, the SLA Enforcement service communicates to the EMS the corrective actions to be taken.

3.2.2. Architectural Description

3.2.2.1. Requirements and Constraints

Job Manager (JM)

The Job manager should be able to handle a broad set of diverse jobs in a transparent and efficient way. These jobs should also include legacy applications and Web service requests.

Candidate Set Generator (CSG)

The CSG is in charge of finding information about "where the job *can* be executed". The EPS (see below) uses the information found by the CSG to choose the resources "where the job *is* executed". As a consequence the CSG should not use any particular algorithm to find the "best" set of resources where to run a job. This duty is accomplished by the EPS.

Execution Planning Services (EPS)

An important requirement is that mobility issues have to be considered while elaborating plans for execution, as well as the reliability of resources to meet the desired fault tolerance level of the Grid infrastructure. New scheduling algorithms should be able to be integrated so as to enable the flexibility of the Grid provider.

Advanced reservation services (ARS)

The ARS service should only be invoked by other services belonging to the EMS component. The ARS should be transparent to the majority of the other services; in particular it should be transparent to the SLA services. The ARS service receives requests from the JM and is not responsible for verifying the SLA between the SC and the resources. The reservation requests are the result of the cooperation between the SLA service, CSG service and EPS service. The reservation request to the ARS should be atomic., that is, the ARS should not be responsible for analysing a complex request and breaking it into atomic pieces. The EPS is responsible for that.

3.2.2.2. Functionality

Job Manager

The Job Manager (JM) encapsulates all aspects of executing a job, or a set of jobs, from start to finish. A set of job instances (a complex job) may be structured (e.g., a workflow or dependence graph) or unstructured (e.g., an array of non-interacting jobs). It may schedule them to resources and it may collect agreements and reservations.

Candidate Set Generator (CSG)

The CSG is in charge of finding the computational resources where the job can be executed. In order to find the possible resources where to execute the job, it is necessary to have some form of match-making mechanism. The match-making permits finding of the resources that correspond to the requirements expressed by the service consumer.

The CSG should take into account all the static requirements but not those that are dynamic. The EPS should start from the match-making results provided by the CSG and combine them with the dynamic attributes, and run an algorithm to find the best resources for execution. The distinction between static and dynamic attributes is strongly related to the Grid concept. To be more precise we give an example of these attributes.

Static resources attributes could be:

- Operating system
- Number of processors
- Software available (libraries, binaries, …)
- Memory
- Disk space
- Bandwidth, etc…

Dynamic resources attributes could be:

- Free disk space
- Available CPU
- File transfer rate, etc…

We have listed the attributes without taking into account any distinction of the possible types of resource.

Execution Planning Services

The Execution Planning Service (EPS) is a high level scheduler that creates mappings called "schedules" between jobs and resources – the resources that have been selected by the CSG. An EPS will typically attempt to optimize some objective function such as execution time, cost, reliability, etc (i.e. answering "where should the job execute?"), in order to improve system performance, provide Quality of Service and meet the Service Level Agreements. For this reason the EPS may implement a job priority system. This can be achieved by having several job queues, each with a different priority. Additionally the EPS may implement a checkpointing or replication scheme for fault tolerance, depending on job characteristics, SLAs and the generated schedule.

Advanced reservation services (ARS)

The ARS is in charge of reserve resources for a specific period of time or permanently, depending on the type of reservation. Different types of reservation are possible:

- Computational resource reservation
- Storage resource reservation
- Network resource reservation
- Service reservation

The computational reservation guarantees that the specified resource is available at the time that a job is executed. The service consumer (SC) can reserve a certain amount of disk space, a specific percentage of CPU, etc.

The storage resource reservation put aside a certain amount of disk storage. In this way the SC is guaranteed to have the requested amount of disk space. It is important to note that, in this case, we are dealing with the *storage disk space*. This is different from the amount of disk space that could be used at runtime during the job execution (handled by the *computational resource reservation*).

The network resource reservation is perhaps the most difficult to achieve. It permits the SC to benefit from the specified network efficiency (bandwidth, etc.) during the entire lifetime of the job. The network resource reservation guarantees that, for example, the job execution will not exceed a specified time in terms of network problems.

The resources are allocated at reservation start time and released when the reservation terminates.

In addition to the ability to reserve resources, the ARS has another important ability. It should be responsible for warning the JM when a reservation time slot occurs. Suppose that a job is running on a resource and that the reservation period is reached. The ARS service should inform (send an alert to) the JM. The JM should take a decision to suspend the job, migrate the job to another resource, or abort the job, etc. Whenever a reservation is made, no matter if the reservation is time-limited (CPU reservation, bandwidth reservation, etc.) or is permanent (storage allocation), the ARS should tell the responsible service to track changes to dynamic resource attributes.

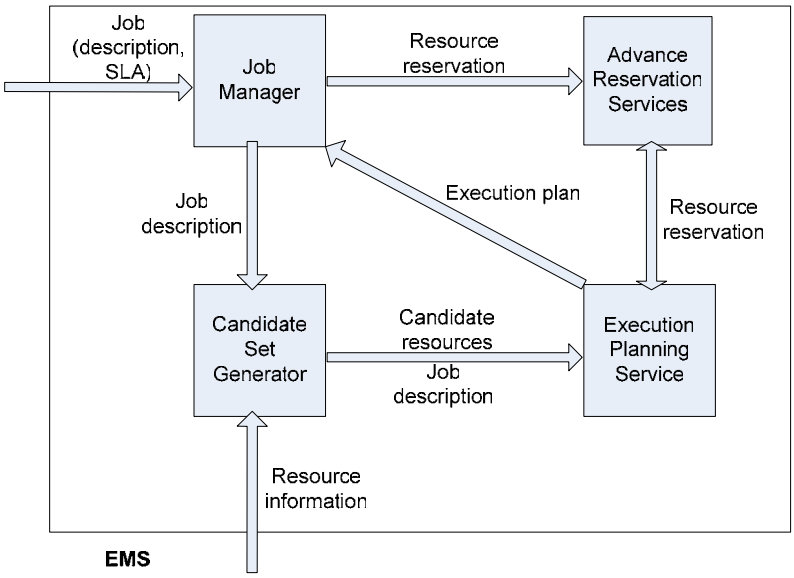

Figure 9: The EMS components and their interactions

3.2.2.3. Interfaces

WP4.3 Subcomponent	Interfaces with Subcomponent, WPs	Interfaces / protocols
EMS	1) Data Management To handle data related jobs (consume / produce data)	SOAP
EMS	2) Monitoring To update the status of the execution and for the manageability of services	SOAP
EMS	3) Metering To receive updated metering information of the execution	SOAP
EMS	4) SLA Enforcement To receive the SLA parameters that have to be met	SOAP
EMS	5) SLA High Level Services (WP4.4) - To check service availability,	SOAP

	- To confirm reservations with SLA-Negotiator.	
EMS	6) Policy Manager To retrieve the policies that will be applied in the execution procedure	SOAP
EMS	7) BP Enactment (WP4.4) To initiate an execution and retrieve the results of it (if any)	SOAP
EMS	8) QoS Broker (WP4.1) To use the network services of WP4.1. To request network resources availability and reservation	SOAP
EMS	9) Service Discovery (WP4.2) To discover the services that need to be used in the various execution phases	UDDI

Table 2: Interfaces for the EMS component

Job Manager

a) Interfaces to components within the EMS

The JM interfaces with all the components of the EMS, namely the Candidate Set Generator (CSG), the Execution Planning Service (EPS) and the Advance Reservation Services (ARS). The JM provides to the CSG the description of the submitted job(s), so that the CSG can identify the appropriate resource(s) for execution. It receives from the EPS the execution plan. Finally it issues requests for resource reservations to the ARS. All these components will be available as services so all the interactions will be carried out using SOAP messages.

b) Interfaces to components outside EMS

The JM will provide a service interface through which a job and corresponding information concerning SLAs may be submitted.

Candidate Set Generator (CSG)

Based on the model that will be used the CSG can have a central repository (like an information index) where all information about the available resources is pushed directly by the resources. The CSG queries the repository to retrieve the information about the resources available that match the specified requirements. Another possibility is that the CSG service has a pool of

resources associated with it. The information index does not exist and the resources talk directly to the CSG. We believe the model should be decided before further design of the interface. We are still investigating the already available models to suggest the one most suited to the Akogrimo scenario.

Execution Planning Services

a) Interfaces to components within EMS

   The EPS will receive from the CSG the job to be executed and the set of appropriate resources. It will also query the ARS for possible reservations made, in order to plan the execution. The generated plan will be provided to the JM. As mentioned above, all these interactions are interactions between services.

b) Interfaces to components outside EMS

   The EPS will not provide any interfaces to components outside the EMS.

Advanced reservation services (ARS)

The ARS interfaces can be of any type and they are not necessarily web service interfaces.

The ARS interfaces are resource-specific. This means that for each type of reservation a specific interface is exposed.

We can define four interfaces (one for each type of resource reservation):

- ARS_computing
- ARS_storage
- ARS_network
- ARS_service

We can divide the attributes of the interfaces in common and specific.

Common attributes could be:

- Time attributes
  - startTime (string dd/mm/yy#hh:mm)
  - endTime (string dd/mm/yy#hh:mm)
  - duration (string d:h:m);
- SC attribute (those attributes that are needed if a check of the Service Consumer identity is required)
  - SC_id
  - SC_credential

Specific attributes.

- Computing attributes
  - dest_IP (string, IP address of the resource)
  - CPU (string, % of required CPU)
  - disk space (string, % of required space)

- Storage attributes
  - dest_IP (string, IP address of the resource)
  - disk quota (string, % of required space)
- Network attributes
  - bandwidth (integer, requested amount of network capacity)
  - fileTransferRate (integer, speed rate)
- Service attributes
  - service name
  - service address

3.2.2.4. Involved technologies

The involved technologies for the set of the EMS services are mainly met in the Globus Toolkit. This includes a set of service components collectively referred to as the Grid Resource Allocation and Management (GRAM). GRAM simplifies the use of remote systems by providing a single standard interface for requesting and using remote system resources for the execution of "jobs". The most common use (and the best supported use) of GRAM is remote job submission and control. This is typically used to support distributed computing applications. In the context of GT4 the involved technology is the Web Service GRAM (WS-GRAM). [8]

Concerning the Candidate Set Generator functionality, various technologies such as Condor-G can be potentially used.

Finally, EGEE (gLite) is one possible candidate for the advanced reservations service.

3.3. Data Management

3.3.1. Overview

The Akogrimo Data Management service is based on the OGSA specification. The OGSA vision offers a broadly applicable and adopted framework for distributed system integration, virtualization, and management. Successful realization of the OGSA requires the definition of a core set of interfaces, behaviours, resource models, and bindings.

This section focuses on requirements, capabilities required to support data management in Grid systems and applications and, finally, on the interactions with other components of the Grid Infrastructure Services Layer.

3.3.1.1. Objectives

Efficient access to and movement of huge quantities of data, data sharing, archiving of data and data management are essential requirements in science, technology and business areas.

Data services are used to move data to where it is needed, manage replicated copies, run queries and updates, and transform data into new formats.

Data services requirements include:

- *Data access*. Easy and efficient access to various types of data (such as database, files, and streams), independent of its physical location or platform, by abstracting underlying data sources is required. Mechanisms are also required for controlling access rights at different levels of granularity.
- *Data consistency*. Consistency should be maintained when cached or replicated data is modified.
- *Data persistency*. Data and its association with its metadata should be maintained for their entire lifetime. It should be possible to use multiple persistency models.
- *Data integration*. Mechanisms for integrating heterogeneous, federated and distributed data are required. Many different types of data must be supported (flat files, streams, DBMS, catalogues, derivations from other data, data services as data resources, etc.). It is also required to be able to search data available in various formats in a uniform way.
- *Data location management*. The required data should be made available at the requested location. It should be possible to allow for selection in various ways, such as transfer, copying, and caching, according to the nature of data.

The study of requirements introduced above results in specific functional and non-functional capabilities regarding Data Management.

3.3.1.2. Functional capabilities

A data resource is any entity that can act as a source or sink of data. This sub-section describes briefly the functional capabilities in line with those provided by the OGSA data services. Not all the functional capabilities provided by the OGSA analysis are taken into account. The only capabilities analysed in this document are those that make sense in the Akogrimo framework.

- **Transparency and Virtualization**

Virtualizations are abstract views that hide the specific characteristics of a huge variety of data resources and allow these data resources to be manipulated without regard to them. These distinctions may be: use of different models to structure the data; different physical media to store it; different software systems to manage it; different schema to describe it; different protocols and interfaces to access it; local or remote storage; unique or replicated data;;materialized or derived data; etc.

- **Client APIs**

This capability allows use of the OGSA data services with legacy applications with existing APIs such as NFS, CIFS, JDBC, ODBC, ADO, POSIX IO or XQuery, by mapping the operations of the existing APIs to the corresponding messages to send to the OGSA data services. OGSA functionality available will depend on the scope of the API concerned.

- **Data Location Management**

Data Location Management allows users to upload, manage and control the access permissions of other users to their own data. DLM offers individual users services such us reliable data transfer from one location to another (by creating a copy of the original or migrating the original completely), data caching at a given location and data replication.

- **Simple Access**

Simple access services provide operations for reading and writing (logically) consecutive bytes from a local or remote data resource (the virtualization interface hides the details of the data location

- **Transformation**

Data services provide data transformation (conversion from one format to another or filtering), explicitly or automatically, in response to certain conditions.

- **Data Update**

Data updating resources depends on the semantics of the data resource, i.e. catalogue updating includes creation, renaming and deletion; structured file and database updating include the update of entries; and for streams and other files the operations are largely limited to appending new data.

Data Services provides support for maintaining the consistency of updates if there are replicated or derived versions of the data resources.

- **Metadata and provenance**

Metadata services are data services that store metadata about Akogrimo data services or other data (i.e. information about the structure of the data, or about the provenance and quality of the data at the level of the whole resource or of its component parts) and provide support for maintaining the consistency of the metadata.

3.3.1.3. Non-functional capabilities

Data Services handle the following non-functional capabilities:

- **Scalability:** in relation to size of data sets, number of data sets, size of data flows, and number of sites.
- **Quality of Service:** implementation of various levels of QoS, such as guaranteed delivery and referential integrity.
- **Coherency:** support for maintaining the consistency of data and metadata in replication, cache and derivation operations.
- **Performance:** minimize the copying and movement of data, a key factor in the overall performance of the Grid.
- **Availability:** features for graceful degradation in the event of network or other failures, i.e. query services may be configured to return partial results when only a subset of sources is available.
- **Legal and Ethical Restrictions:** the Akogrimo data services may have to operate within an environment where a variety of legal and ethical policies affect their operation (confidentiality, privacy, rights, etc.). The security mechanisms provided by Akogrimo allow the specification of policies that apply at the level of groups (e.g. tables) or elements within a resource.

3.3.1.4. Interaction with other components of the Layer

This section summarizes the interactions between the data related services and the other components of the Akogrimo framework.

- **EMS**
  - Should find data requested by the client (physical location)
  - Should replicate data if requested by the client
  - Choose best replica to be accessed (during the scheduling phase)
  - Reserve space for data (advance reservation)
- **Monitoring**
  - No interaction foreseen.
- **Policy Management**
  - Data access policies should be negotiated with the Policy Management service
- **Security**
  - Data should be protected

3.3.2. Architectural Description

In this paragraph we introduce the basilar components of the Data Management service. We can group the components into three main categories:

1. *Data movement*
2. *Data replication*
3. *Higher level data service*

The components of the Data Management services are the following:

- *Data movement*
  - ***Reliable File Transfer***
- *Data Replication*
  - ***Data Replication Service***
- *Higher level data service*
  - ***Replica Location Service***
  - ***Storage Element***
  - ***Metadata Service***

Data transfer is based on:

- File transfer protocol

In the following paragraphs a description of each component and its role is given.

3.3.2.1. Functionality

3.3.2.1.1. Data movement

Reliable File Transfer

The RFT is in charge of moving data from one location to another. It provides a "job scheduler"-like functionality for data movement. Provided with a list of source and destination URLs the service writes a job description into a database and then moves the files on behalf of the requestor.

3.3.2.1.2. Data replication

Replica Location Service

The Replica Location Service (RLS) allows the registration and discovery of replicas. An RLS maintains and provides access to mapping information from logical names for data items to target names. These target names may represent physical locations of data items or an entry in the RLS may map to another level of logical naming for the data item.

3.3.2.1.3. Higher level data services

Data Replication Service

The primary functionality of this component is to allow users to identify a set of desired files existing in their Grid environment, to make local replicas of those data files by transferring files from one or more source locations, and to register the new replicas in a Replica Location Service.

Metadata Service

This component relates to the metadata, e.g. the data about data. The metadata service is in charge of maintaining information about the data stored, to set, get and query the metadata.

Storage Element

The local storage system could be simply a file system or even a high-capacity mass storage device. The Storage Element should be placed on top of the local storage system and provide functionalities to upload data, retrieve data, replicate data, etc. The existing of a Storage Element is very important to standardise the way data are stored and accessed.

3.3.2.2. Interfaces

Table 3 lists the relationships between other component of the WP4.3 layer (other than Data Management) and sub-components of Data Management.

WP4.3 Subcomponent	Interfaces with Subcomponent, WPs	Interfaces / protocols
EMS	Reliable File Transfer	EMS request for files transfer (SOAP)
EMS	Replica Location Service	EMS discover the best replica based on the

		information provided by the RLS (SOAP)
EMS	Data Replication Service	Ask to replicate a file (SOAP)
EMS	Storage Element Service	Reserve disk space
Policy Manager	Storage Element Service	Negotiate local data access (SOAP)

Table 3: Interfaces for the Data Management component

A more detailed description of the interfaces provided by each sub-component and about its features is described below.

Reliable File Transfer

- file transfer request

  This interface is responsible to transfer a file, as requested, from one location to another.

Replica Location Service

- Discover replicas

  As mentioned before each data file has a unique identifier and could be replicated in different locations. The *discover replicas* interface discovers the physical location of the data file.

- Register replicas

  Each replicated file can be registered in the Replica Location Service

Data Replication Service

- Add data files

  This interface permits addition of a data file to the Data Replication Service. This is not the same functionality as adding the file to the local disk storage or to the Storage Element.

- Register data files

  The Data Replication Service should trace all the data files that pertains to the DRS. Each data file should be identified by a unique identifier and registered to the DRS.

- Replicate files

  Data files can be copied from one location to another. The replication of the file should be registered. In this way a unique identifier can be associated with more than one physical copy of the data file.

- Delete replicas

  It should be possible to delete a replica from the DRS.. This does not mean that the data file is physically removed from the storage but that it is simply unregistered from the DRS.

Metadata Service

- set metadata

  It should be possible to add metadata related to a replicated file.

- get metadata

  It should be possible to retrieve metadata information related to a replicated file

- query metadata

  It should be possible to query a metadata catalogue to find out information related to replicated files. The queries should be based on keywords.

Storage Element

- copy file

  This interface permits upload of a file to one or more Storage Elements

- retrieve file

  This interface permits retrieval of a file from a specified Storage Element

- move file

  This interface allows the movement of a data file to a different location on the same Storage Element

- delete file

  This interface enables deletion of a data file from one Storage Element

The interfaces listed above are those exposed by the Data Management services to other services/users.

3.3.2.3. Involved technologies

In this paragraph we analyse two possible technologies that fulfil the requirements and architecture described in the previous paragraphs.

The technologies that we consider in this overview are:

- GT4 Data Management
- gLite Data Management

3.3.2.3.1. GT4 Data Management

The GT4 Data Management is composed of several components. These components are divided into two categories: those that are WS-Component (Web service based) and those that are not.

The non WS-Components are:

- Replica Location Service (RLS)
- GridFTP

The WS-Components are:

- Reliable File Transfer (RFT)
- OGSA-DAI
- Data Replication Service (RLS)

The RLS and GridFTP components are compliant with the description given in paragraph 3.3.2

The WS-Components are briefly described hereafter.

The ***Reliable File Transfer (RFT)*** Service implementation in GT4.0 uses standard SOAP messages over HTTP to submit and manage a set of 3rd party GridFTP transfers and to delete files using GridFTP. The user creates a RFT resource by submitting a list of URL pairs of files that need to be transferred/deleted to RFT Factory service. The RFT service implementation exposes operations to control and manage the transfers (the resource).

OGSA-DAI provides a pure Java data service framework for accessing and integrating data resources - such as files, relational and XML databases - into Grids. Towards this end, OGSA-DAI:

- Exposes intrinsic data resource capabilities - such as the ability to perform SQL queries on relational resources or evaluate XPath statements on XML collections - through web service based interfaces, thus allowing data resources to be easily incorporated as first class citizens in grids.
- Allows additional functionality to be implemented at the service - such as transformation of data coming out of a data resource - so as to avoid unnecessary data movement.
- Provides a compact way of handling multiple potential interactions with a service within a single request via an XML document, called a perform document, where data is pipelined between different set of activities that operate on a data stream coming out of, or going into, a data resource.
- Allows developers to easily add or extend functionality within OGSA-DAI. The perform document, and underlying framework, are extensible allowing additional functionality to be added, or existing functionality to be customised, and still operate within the same framework.
- Allows metadata about data and the data resources in which it is found to be queried via a service interface.
- Facilitates the provision of data integration capabilities from various sources to obtain the required information.

The ***Data Replication Service (DRS)*** is a technical preview provided with the Globus Toolkit 4.0 and first appears in the GT 3.9.5 Beta release. The primary functionality of the component allows users to identify a set of desired files existing in their Grid environment, to make local replicas of those data files by transferring files from one or more source locations, and to register the new replicas in a Replica Location Service. The DRS conforms to the WS-RF specification and exposes a WS-Resource (called a "Replicator" resource) which represents the transactional state of the requested replication activity and allows users to query or subscribe to various Resource Properties in order to monitor the state of the resource. The DRS is built on the GT 4.0 Java WS Core and uses the Globus RLS to locate and register replicas and the Globus RFT to transfer files. The service also depends on use of a standard database server via JDBC interfaces. A client tool is included that allows users to submit a request file as a command line parameter that contains a list of data files to be replicated at the site. The client tool creates a Replicator resource to maintain state for the replication operation.

3.3.2.3.2. gLite Data Management

The gLite project represents an initiative to develop lightweight middleware for Grid computing. This approach is based on the assumption of dealing with files instead of data sets, data works or tables regarding relational databases. In the gLite, the data management functionality is provided by means of the following three services or sub-components:

- Storage Element

- Data Scheduler
- Catalogs

The ***Storage Element*** (SE) is the component in charge of managing every process related to upload a certain file into the GRID storage infrastructure. The SE manages disk space form permanent files and maintains the cache for temporary files. The control and management of the storage process require the existence of the following main services or components:

- Storage Resource Management (RSM) guarantees a common interface to every type of individual storage (from a simple file system to a high-capacity mass storage device).
- File Transfer Service (FTP). The functionality of this service is simply to carry out the transfer of certain files when required.
- GLite File I/O service. This service or interface allows the access of an end-user to files in the GRID through some supported protocol.

The ***Data Scheduler*** (DS) assures that a given file is available at the chosen site where the task will be executed. In addition, the Data Scheduling services expose non trivial interfaces to the user for data placement in a distributed environment. The DS is also responsible of organising and keeping track of all transfers occurred within the system.

Finally, the ***Catalogs*** component is in charge of storing the location of data. Data is univocally determined by a logical name and, due to the replication of data, each logical name might correspond to several files in different locations. This correspondence between logical name of certain data and the physical counterpart is stored in Catalogs. Several types of Catalogs might be encountered depending on the type of data managed. File Catalog, Metadata catalog and Replica Catalog are different examples.

3.3.2.4. Use case view

In this paragraph we describe a general use case that shows the relationships between the internal components of the data management.

Before describing the use case we want to introduce some general definitions that are used in the use case and that are general terms that will be used also during the development phase of the Data Management component.

- A *logical file name* (**lfn**) is a unique identifier for the contents of a file.
- A *physical file name* (**pfn**) is the location of a copy of the file on a storage system.
- A *unique identifier* (**uid**) is the unique identification of a registered file.
- A *storage uniform resource locator* (**surl**) is the address of the storage system.
- A file is called *replica* if it is already uploaded into a SE and registered into the RLS.

Having in mind the e-health scenario, we propose the following general use case focused on the Data Management services.

A patient required the intervention of a doctor in an emergency case. The doctor is abroad and should retrieve some data related to the patient. The data are stored in two different *SE* located in different cities. One place is where the patient lives and works during the winter, the other

place is where the patient spends his holiday. Depending on the context (e.g. where the doctor is) some data could be retrieved in a faster way (depending on the network availability). In addition the doctor does not know which files to retrieve but he knows just some information about the patient and about the type of analysis he is looking for. For this reason the *metadata service* should also be used. Once the ***lfn*** of the files is known, the *replica location service* is in charge of finding the ***pfn*** corresponding to the ***lfn***. The EMS will calculate the best files to be retrieved. The SE service will return a list (if more then one) of available files providing a ***surl***. The *Reliable File Transfer* service will be in charge of moving the patient data from the SE location to the doctor's location. (The *Data Replication* service does not enter this use case scenario because we assume that the files are already registered.)

This general use case is sketched in Figure 10: Data Management Use Case. This use case in not exhaustive, it only gives a general idea of the possible use of the Data Management component. It is a qualitative use case that should help in understanding the roles of the different sub-components. In Annex more specific use case examples are provided to explain the relationships between sub-components of the Data Management service from an implementation viewpoint.

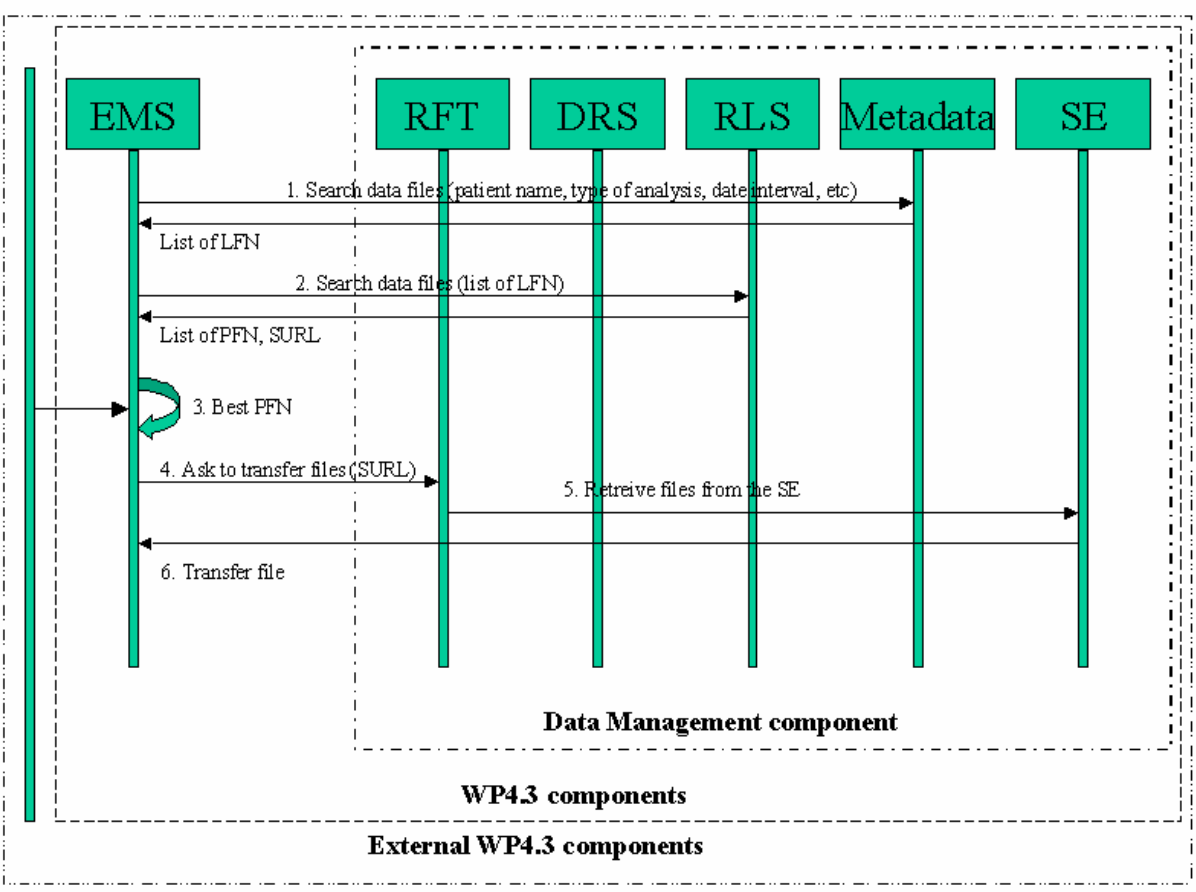

Figure 10: Data Management Use Case

3.4. Monitoring

3.4.1. Overview

This section describes the Monitoring subsystem of the Grid Infrastructure Services Layer. WSDM [10][11] and WSRF [2] are considered.

3.4.1.1. Objectives

Monitoring tasks are fundamental in every distributed computational system. Monitoring data represents an operational snapshot of the system behaviour along the time axis. Such information is fundamental to determining the origin of the problems or for tuning different system components. For instance, fault detection and recovery mechanisms need a monitoring component to decide whether a particular subsystem or server should be restarted due to the information collected by the monitoring system.

3.4.1.2. Functional capabilities

In the particular case of Akogrimo grid computing system, the following functionalities have been identified for the Monitoring component:

- To monitor quality of services.
- To monitor service level agreements.
- To check resources status.
- To provide monitoring information used for controlling tasks.

In order to carry out these tasks and taking into account the nature of the Akogrimo system, a monitoring system based on web services appears to be the more adequate option. In that sense, the OASIS Web Service Distributed Management (WSDM) Technical Committee has defined a set of specifications, Management Using Web Services (MUWS) that leads with the requirement to manage (not just to monitor) a distributed system by using web services. From the implementation point of view, this specification is mainly based on the work realized by the Web Service Resource Framework developing group (WSRF). In that sense, MUWS capabilities could be suitable to fulfil the Akogrimo monitoring requirements.

3.4.1.3. Non-functional capabilities

The monitoring data has the following particular characteristics:

- Short lifetime of utility: If a particular server has a deficient performance over a period of time, such information is mainly important at that time in order to correct such behaviour. Once the problem has been overcome this monitoring data becomes valid only for performance statistics.
- Data monitoring is very dynamic: Since the monitoring data reflects the online system performance, the information updates are very frequent.
- Statistical aspect of monitoring data: A single value of a certain parameter such as CPU load is not as important as the collected values over a certain period of time. In that sense, statistics represents a different information system point of view.
- Monitoring data must be managed in a distributed fashion.

As a direct consequence of the characteristics of the monitoring data expressed above, every monitoring system should provide mechanisms in order to keep the following capabilities:

- Low latency: Monitoring information should be transmitted as quickly as possible since the lifetime of utility of certain data is very short. For example, the failure of a particular service should be transmitted very quickly to the system in charge of restarting the service.
- High data rate generation: The production rate of relevant monitoring data is, in general, very high, consequently the monitoring data system should handle adequately such amount of information.
- Minimal measurement overhead: Monitoring performance should be carried out without affecting the available system resources. For example, a very complete monitoring system will not be useful if it requires a considerable amount of system memory in order to operate adequately.
- Secure: Information generated by the monitoring system is sometimes very sensitive. In that sense, the monitoring system should ensure the data integrity and access permissions.
- Scalable: The monitoring architecture should provide flexibility to overcome and handle correctly a possible growth of systems to be monitored.

3.4.1.4. Interaction with other components of the Layer

The different interactions between monitoring components and the remaining WP4.3 components will be analysed:

3.4.1.4.1. EMS

Execution Management Services (OGSA-EMS) are concerned with the problems of instantiating and managing tasks. Within OGSA-EMS a task refers to a single unit of work to be managed – for example a legacy batch job, a database server, a servlet running in a Java application server container, etc.

The monitoring component interacts with the following two EMS subsystems:

Job Manager

Job Manager as a producer

The activity of the Job Manager needs to be monitored. A web service should be implemented that can monitor several Job Manager properties such as Identity, Availability, Status (State capability) and a description of the manageable resource (for the sake of clarity).

Job Manager as a consumer: provisioning stage

One of the responsibilities of the Job Manager is the orchestration of the set of services to start a job or set of jobs. In that sense, JM needs availability service information as well as the knowledge of other resource-specific features. It seems to be essential to define a Web Service for each resource that may be invoked by JM that implements the following manageability capabilities: Identity, availability, status, in addition to other resource-specific capabilities.

Candidate Set Generator

Candidate Set Generator (CSG) determines the set of resources on which a unit of work can be executed. CSG determines or monitors the availability, capacity and other characteristics. Therefore, CSG receives notifications about the availability of all resources susceptible to be used, about other resource-specific capabilities to be determined. In addition, information about

the network status becomes basic. A web service should be implemented able to notify different network properties. Within the general monitoring architecture, CSG represents a Consumer and will query different resources to determine whether the resource is able to carry out the execution required.

3.4.1.4.2. Service Level Agreement

A Service Level Agreement (SLA) is concerned with the aspects related to the quality of service of all services offered in Akogrimo.

The monitoring component interacts with the following SLA subsystem:

SLA-Controller

After the negotiation phase, once instances of services have been created, it is necessary to ensure that the service provider observes the contractual terms. The Execution Manager System notifies the Monitoring that a new service is being used. At this moment the Monitoring component is in charge of measuring and probably sending the QoS value of this service to the SLA-Controller component.

At this point, the Monitoring system and the SLA-Controller should establish a communication. This communication can be established in two different ways: notification (CurrentParameterStatus operation defined in SLA section) or request/response (GetParameterStatus operation). The basic information interchanged is the QoS of the service. This information is fundamental for the SLA-Controller in order to verify the degree of fulfilment of the agreement.

3.4.1.4.3. Metering

The Metering component is concerned with the production of well-defined metric data that could be necessary for the operation of the other components of the layer. The Monitoring component will be in charge of to communicate the Metering component with the rest of the components asking for this kind of data.

3.4.2. Architectural Description

Next, a general architecture for a monitoring system is presented, where the main actors and processes will be briefly described. As shown in the Figure 11, the monitoring system consists of four main components:

- Producer: A component that produces the monitoring information.
- Consumer: A component interested in the producer monitoring information.
- Registry: A repository of information about consumers and producers.
- Discovery component: A component in charge of supplying to a consumer the adequate producer.

A producer can be, for instance, a printer that reports every ten minutes about its toner level. In this case, a consumer can be considered any system in charge of controlling the printer's toner level. In this particular case, the discovery component looks for the appropriate producer within the directory service to supply it to the consumer.

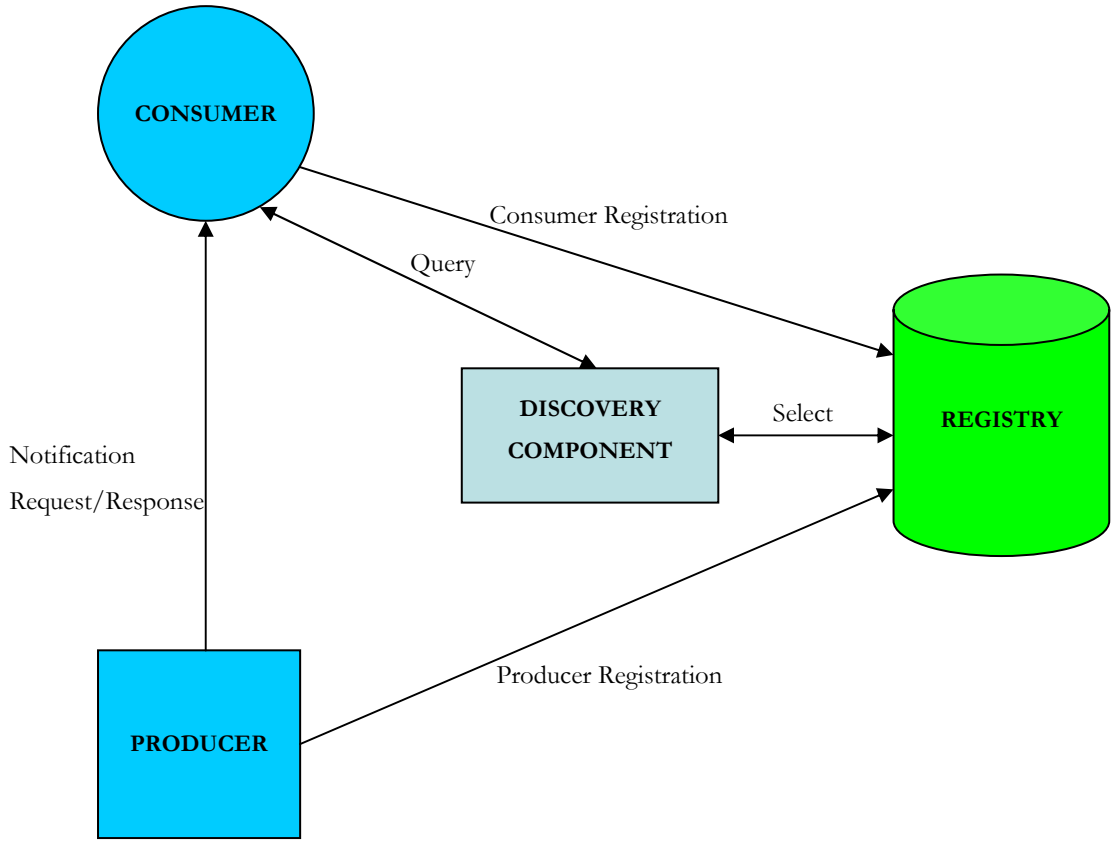

Figure 11: Architectural description of the Monitoring component

The Registry is a component where both consumers and producers might publish information about the type of information they are interested in (consumer case) or the type of information that they generate (producer case). The Discovery component provides publishing and discovery mechanisms. The consumers can ask to the Discovery component if there exists any component (producer) providing a specific device information. Then, the Discovery component will carry out a "select" to the Registry. Finally the Discovery component facilitates the "contact" information in order to allow both components to communicate directly each other or to allow the consumer to subscribe to producer notification procedures. The Registry and the Discovery components provide with mechanisms to add/update/remove entries (consumers or producers) and with searching mechanisms, respectively.

The communication between consumers and producers can be of three types: publish/subscribe, query/response and notification. The publish/subscribe interaction is carried out in three different stages. The first one is the subscription where the initiator (consumer or producer) shows interest in some events supplied by the server (producer or consumer). At this stage some parameters such as buffers size, timeout and period of time between communications are exchanged. The second stage constitutes the "functional" communication between consumer and producer. The final stage represents the unsubscription.

The query/response interaction consists in two stages. In this case the initiator should be the consumer. The first stage establishes the transfer condition in a similar way as the first publish/subscribe stage. Finally, in the second stage the producer send all the information required to the consumer without an unsubscription communication. This procedure resembles very much to the well-known http request/response.

The notification interaction is a single stage procedure always initiated by the producer. The producer communicates all the performance events in a single communication to the consumer.

Next, we introduce the general lines of management carried out by means of web services. The specification shown now is not only valid for monitoring systems but for a general management system so that a specific fit should be done in order to apply it to the monitoring case.

3.4.2.1. Requirements and Constraints

It is supposed that the network status will be provided from the Metering component in the network layer, but if this condition will not be fulfilled, the Monitoring component should arrange a manner of obtaining certain parameters about the network status.

In addition, each entity providing monitoring information must implement a web service offering the values of the parameters and, in the same way, each entity that consumes information must implement another web service.

3.4.2.2. Functionality

3.4.2.2.1. Management using Web Services

The MUWS specification of WSDM defines how the ability to manage, or, how the *manageability of*, an arbitrary *resource* can be made accessible via *Web Services*. In order to achieve this goal, MUWS is based on a number of web services specifications, mainly for messaging, description, discovery, accessing properties, and notifications, all of them included within the so-called Web Service Resource Framework (WSRF). At this point, it is very important to note that monitoring tasks must be considered as a subset of management.

The MUWS specification defines the following basic entities/actors:

- Manageability endpoint (Web service endpoint): The access to the resource to be managed.
- Manageability consumer: The entity carrying out the management of the resource.
- Manageability resource: The entity being managed.

The basic concepts of management using Web services can be illustrated by the Figure 12:

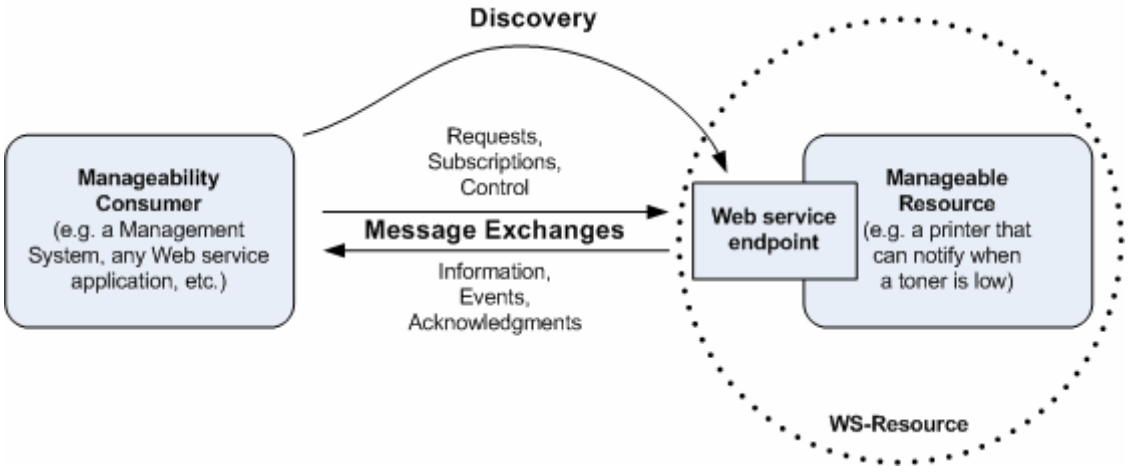

Figure 12: WSDM Concepts

The basic process of resource management can be summarized as follows:

1. Manageability consumer discovers the manageability endpoint by means of a certain discovery mechanism that is out of the scope of the monitoring component (discovery mechanisms are the same as those used for standard web services).
2. Manageability consumer and the web service endpoint exchange some messages in order to request information, subscribe to events, or control the manageable resource associated with the endpoint.

A manageability consumer first obtains from the web service endpoint an Endpoint Reference (EPR), as defined by the WS-Addressing [9] specification, and afterwards obtains any other required descriptions, including, but not limited to, a WSDL [15] document, an XML Schema, or a policy document. MUWS uses the same mechanisms, for obtaining EPRs and their associated descriptions, as used by regular Web Service implementations, and their applications.

The central entity in WSDM specifications is the resource and its management. WSDM specifications involve the way a manageability entity (manageability consumer) accesses and manages certain resources. There are two different aspects related to resources: functionality and manageability. For instance, the functional aspect for printers is printing whereas the manageability aspect would be everything related to the control and management of the printer: level of the toner, whether the printer is online/offline. Every resource that is susceptible to being managed is called a manageability resource. A manageable resource may support a number of capabilities. In particular, an implementation of a manageable resource should provide a number of manageability capabilities via Web Services endpoints. Consequently, the manageability consumer will access and manage manageability resources via manageability endpoints that from am implementation point of view are Web services endpoints.

The manageability capabilities implemented in the manageability endpoint can be classified in two different groups:

- General manageability capabilities: They represent those capabilities that are common to every manageability resource such as: identity or availability capability.
- Resource-specific manageability capabilities: Those capabilities are properties of the particular manageability resource.

WSDM-MUWS specifications define the manageability capability as being composed of the following elements: properties, operations, events and metadata.

Properties of a manageability resources are exposed via a XML document called the resource properties document. From an implementation point of view, the resource properties document is included within the *type* label of WSDL document.

The operations over a manageability capability are indicated via the label portType in the WSDL document of the corresponding web service (manageability) endpoint.

Events are offered by the manageability endpoint whenever a certain property changes. Their implementations are based on a WS-Notification specification[14]. Their properties will be included within the resource property document whereas the operations, such as subscription, can be found within the *portType* label.

Finally metadata can be specifically defined for properties, operations and events. MUWS supplies three metadata items: mutability, modifiability and capability, but also allows further items to be defined.

3.4.2.2.2. General manageability capabilities for Akogrimo resources

First, we will focus on the general manageability capabilities suggested by the MUWS specifications. In case any other general capability should be provided, they will be included later. All the manageability capabilities here included are explained in detail in the following references [10][11]**Error! Reference source not found.**.

- Identity: The aim of the identity capability is to determine whether two different manageable resources are the same. This is the only manageability capability that MUWS states to be compulsory to define for every manageable resource. The uniqueness of the resource is established through the only Identity property: *ResourceId* [13].
- Manageability characteristics: The manageability characteristics show the list of different capabilities that the manageable resource supports.
- Correlateable Properties: The correlateable properties capability supplies an alternative way to the Identity capability for determining whether two manageable resources are the same or not. In this case, the identity of the resource is established by means of a set of particular conditions based on specific-resource properties (e.g. for a printer this could be ip, host,...).
- Description: The goal of the description capability is to supply a human-readable description of the managed resource. This is accomplished via the following properties: *Caption*, *description* and *version* that represent the name, description and version of the resource being managed respectively.
- State: The state capability represents a way to allow the representation of very different state models applied to different manageable resources.
- Operational status: The operational status capability represents the availability of the resource. The availability is modeled through the property *OperationalStatus* and the values such a property can take: *Available, partially available, unavailable* and *unknown.*
- Metrics: Metrics capability represents a collected value over a period of time. More precisely, it represents a measurement of a particular property (such as CPU load) over a certain period of time.
- Configuration: The configuration capability represents to any resource property such that the change of its value implies some operational behavior.

3.4.2.3. Interfaces

WP4.3 Subcomponent	Interfaces with Subcomponent, WPs	Interfaces / protocols
Monitoring	EMS (Job Manager), WP4.3	SOAP
Monitoring	EMS (Candidate Set Generator), WP4.3	SOAP
Monitoring	SLA (SLA-Controller), WP4.3	SOAP
Monitoring	Metering, WP4.3	SOAP
Monitoring	Metering, WP4.1	SOAP

Monitoring	Accounting, WP4.2	SOAP

Table 4: Interfaces for the Monitoring component

3.4.2.4. Involved technologies

The protocols used in the component interactions will be XML Schema, SOAP and WS-Notification.

A MUWS and/or WS-* platform implementation is required in order to develop a monitoring component. The new Globus Toolkit 4.0 supplies a first approach to such an implementation using Java as the web service programming language and Tomcat-Axis as the web and soap container respectively.

3.4.2.5. Configuration and Deployment

Each time a new web service is developed, a Tomcat deployment process has to be carried out.

3.5. Service Level Agreement Enforcement

3.5.1. Overview

Within the WP4.3 context, the SLA subsystem handles those aspects related to the quality of service (QoS) of all offered services in Akogrimo. When a Service Costumer (SC) wants to use a service published by a Service Provider (SP), they must negotiate the conditions of use which are sealed in the SLA contract. Thus, during the use of any kind of service, the agreed QoS level must be assured. SLA subsystem is in charge of watching over the fulfilment of this QoS, and to know by means of the Policy Manager what recovering actions should be applied in case that the QoS described in the contract is not respected throughout the execution phase. At this point the figure of a trusted third party appears; part of SLA functionalities can be optionally located on an independent third party trusted by SC and SP, who will look after the fulfilment of the agreed conditions.

While a service is being used, some particular or abnormal situations could happen, especially in Akogrimo's mobile context, for example temporary unavailability of mobile resources etc. In this case, it is necessary to overtake some actions at different levels (from network level to the application level) in order to guarantee correct service provision to the SC.

3.5.1.1. Objectives

SLA is present in several places inside the Akogrimo infrastructure. The negotiation phase between a SC and a SP is thoroughly tackled in WP4.4. This negotiation culminates with the creation of a contract that reflects the QoS agreed. The contract must also contemplate all the possible situations that could happen during the use of the service (part of this information is split in the policies managed by the Policy Manager). Once a service is being used, parts of the SLA Management have to oversee the execution phase, to assure that the service is running as is required. In this case the SLA Enforcement integrated inside WP4.3 appears. One component of

this is an agent (SLA-Controller) that is responsible for the verification of the contract conditions (QoS thresholds). Whenever a service does not meet these conditions this agent notifies it to the SLA-Decisor module (the other component of SLA Enforcement), which is in charge of starting the appropriate actions. Both components only act as supervisors of the service for all the time it is running. Their role is to be alert for any abnormal situation and to take quick and effective decisions (assisted by the Policy Manager) each time the system does not offer the agreed conditions. They can be seen as "communicators" of the running services' status to inform other subsystems of anything wrong, but they do not perform direct actions over any component, as this is the responsibility of other subsystems.

3.5.1.2. Functional capabilities

The following functional capabilities have been identified for SLA Enforcement:

- **To receive measurements**:
  - To gather all measurements performed by monitoring subsystem.
  - To ask directly for a service status when is necessary.
- **To check filters:**
  - To evaluate thresholds for the verification of fulfilment of the QoS detailed in SLA-Contract.
- **To propagate violations:**
  - To generate the necessary notifications / alarms in case that QoS is not met.
- **To find policies:**
  - To request the analysis of associated SLA policies described in the Policy Manager.
- **Recovery actions:**
  - To send information about bad status situations during the execution phase.
  - To request the execution of enforcement action (recovery actions described in corresponding policies).

3.5.1.3. Non-functional capabilities

These are identified several non-functional capabilities:

- **Interoperability:** to support open standards to guarantee the interoperability.
- **Scalability:** a general enough SLA management to a distributed environment.
- **Quality of Service:** management of all aspects related to provision of specific QoS.
- **Performance:** efficient management of communications related to SLA management, avoiding the introduction of excessive overhead in making decision process.
- **Availability:** assure a continuous control of the service status to hide any unexpected situation from the user.
- **Auditing:** to assure independence in QoS evaluation, some functionalities of SLA Management (verification of violation) could be potentially located in an external trusted third party.

3.5.1.4. Interaction with other components of the Layer

Figure13 outlines the interaction of SLA Management modules in WP4.3 with the rest of components.

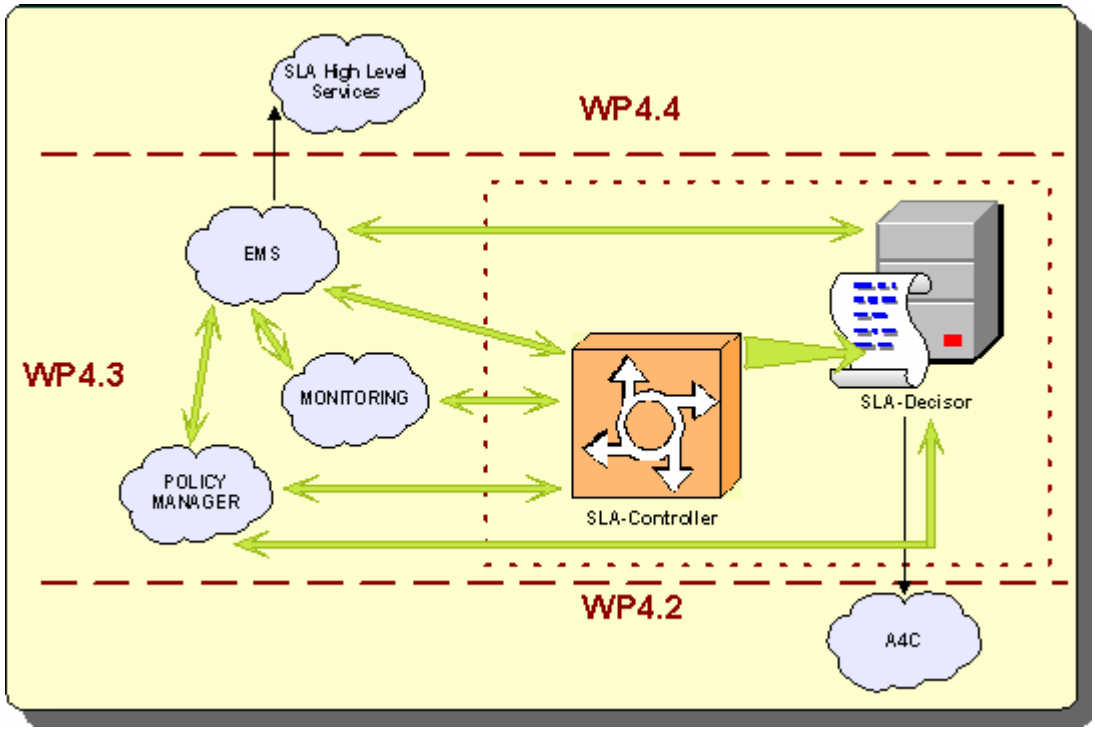

Figure 13: SLA architecture in WP4.3

Regarding WP4.3, SLA Enforcement (marked between the dashed lines in Fig. 13) is in charge of controlling the correct evolution of on-going service execution from a QoS point of view, processing the information received from the monitoring and requesting the application of necessary corrective actions described in the policies.

As we have seen, SLA needs to interact with other subsystems to be able to manage SLA information. These are the subsystems that interact directly with the SLA Enforcement (intra and inter WP modules):

EMS: The Execution Management Service will enable the SLA-Controller for evaluating the status of the service. EMS will also be in charge of applying corrective actions requested that are derived from permanent QoS metric violations.

Monitoring: This sends all measurements in order to enable the SLA-Controller to perform verification of events/violations/etc produced with regard to predefined filters and quality metrics during service execution.

A4C: SLA-Decisor when tracing violations of QoS it informs A4C for its accounting.

Policy Manager: The SLA interacts with Policy Manager in two well-defined cases: on one hand to get the required filters for QoS thresholds that are applied in a concrete domain and on the other hand to obtain a suitable policy that must be applied in case of violation.

3.5.2. Architectural Description

3.5.2.1. Requirements and Constraints

SLA Management subsystem is split into two layers corresponding to WP4.3 and WP4.4. In this section, we justify the reasons why it is necessary to do a distribution of the SLA Management subsystem in these two layers.

Concerning the participation of the SLA subsystem in the overall operation of Akogrimo platform, we clearly distinguish two phases; the "negotiation" and the "runtime". This distribution has constrained SLA design and therefore we have split the responsibilities throughout these two layers; SLA High level services for WP4.4 and SLA Enforcement for WP4.3.

The SLA is involved in the negotiation phase when the OpVOBroker is searching for specific services; this is covered in more detail in the WP4.4 component design document. This section focuses on the runtime phase, in which the SLA Enforcement controls the status of the services and manages any violations produced. SLA Enforcement is composed of the SLA-Controller and the SLA-Decisor. Both sub-modules follow the execution of the running services in a VO.

All services deployed in the Akogrimo infrastructure must consider the SLA aspect, so that it is possible to control them and to guarantee the best service provision. However, there can be particular cases in which services are not under the control of SLA Management. Concretely we refer to "generic" services offered for free with non-economic purposes (a QoS cannot be requested) or simply to services whose nature is not liable to be monitored (for instance the "establish a phone call" service has a binary result and no QoS parameters).

3.5.2.2. Functionality

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

SLA-Controller

This module is in charge of supervising that all the agreements reached in a contract are respected. It receives and processes all measurements performed by Monitoring subsystem and therefore it shall be deployed together with the Monitoring subsystem in order to avoid network communication overheads. It checks whether the measurements are within the thresholds established in SLA contract for QoS metrics. This module also applies the domain filter extracted from Policy Manager before sending eventual notifications to SLA Decisor (in cases when the QoS is not fulfilled).

SLA-Decisor

This receives and manages notifications from the SLA-Controller. In turn, it requests the Policy Manager to retrieve the suitable policy associated according to the execution context. This policy contains the actions that must be undertaken (to show informational messages, to increase processes priorities, to apply discounts, to destroy the service, to re-instantiate the service, etc). Depending on the seriousness of the violation, context of 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 apply corrective actions, whereas in the second case,it may be necessary to update higher layers (Workflow manager or other subsystem) to recover from the situation by applying global corrective actions. It is the responsibility of the designer to

decide which performed actions will be managed directly by the Policy Manager or the SLA-Decisor. After selecting the best policy, the Policy Manager can ask directly for a concrete action on another component, or on the other hand it can communicate the corrective action to the SLA-Decisor.

3.5.2.3. Interfaces

Next table shows all the interactions between SLA components and other modules and between themselves in the Akogrimo infrastructure. These interactions are described in detail in the Annex section.

WP4.3 Subcomponent	Interfaces with Subcomponent, WPs	Interfaces / (protocols)
EMS	SLA-Controller	To create an "agent" to control the status of the new service created / SOAP
SLA-Controller	Policy Manager	To obtain the mapping defined between the High and Low level parameters from a service / SOAP
SLA-Controller	SLA-Decisor	To create a subscription of SLA-Controller into the SLA-Decisor for posterior communications between these two modules/ SOAP
Monitoring	SLA-Controller	To notify SLA-Controller about the value of the QoS dimensions related to the service / SOAP
SLA-Controller	Monitoring	To ask directly Monitoring about the value of the QoS dimensions related to the service / SOAP
SLA-Controller	SLA-Decisor	To inform to SLA-Decisor about a violation produced on the QoS fulfilment of the service/ SOAP
SLA-Decisor	Policy Manager	To ask for the best policy to apply in case of violation / SOAP
SLA-Decisor	EMS / Workflow Manager (according to associated policy)	To perform action when something has going wrong during the service execution / SOAP

Table 5: Interfaces for the SLA enforcement component

- **ActiveServiceController:** After EMS creates a new service instance it also enables a SLA-Controller that will be the responsible of the fulfilment of the SLA-Contract for this service. The result of this action is a new SLA-Controller agent associated to the service instance.
- **GetPolicyFilters:** Once SLA-Controller is created it accesses to the Policy Manager and searches the policy metric for mapping parameters. This action is done only one time and before starting to receive measurements, so this module is ready to analyse the information of the service use.
- **RegisterToServiceDecisor:** SLA-Controller looks for a SLA-Decisor instance to be subscribed and then be able to notify all violations occurred during the service execution. When SLA-Decisor has received the maximum number of SLA-Controller instances allowed, is necessary to create a new instance of it.
- **CurrentParametersStatus:** Once EMS has notified to Monitoring that a new service is being used, Monitoring is in charge of measuring and sending the QoS value of this service. This asynchronous function is called periodically on the Monitoring subsystem and the result is the QoS measurements of the service that SLA-Controller receives. SLA-Controller receives the status of the QoS related to a service. This action will be repeated automatically after reaching a determinate period of time until the service is destroyed.
- **GetParametersStatus:** Once EMS has notified to Monitoring that a new service is being used, SLA-Controller can interact with the Monitoring subsystem and requests the status of the QoS related to this service. The use of this function depends on the workflow implementation, but generally it is not necessary due to CurrentParameterStatus function will send periodically the QoS measured values to the SLA-Controller.
- **CheckFilters:** SLA-Controller applies the filters that it knows related to the service over the measurements that Monitoring send to it. Due to the application of these filters is possible to avoid some possible violations and don't give rise to perform some unnecessary recovery actions.
- **SetViolation:** After checking all the received measurements and applying the existent filters, SLA-Controller knows that something is going wrong in the service execution and that a violation has been produced, so it sends a notification to SLA-Decisor.
- **BestPolicy:** SLA-Decisor looks into the Policy Manager about recovery policies in relation with the violation it has received. Policy Manager must return the best action to perform to correct this undesired situation. Instead of returning the control to the SLA-Decisor. The Policy Manager can also order the execution of concrete actions to other components.
- **SetAction:** If the control is returned to the SLA-Decisor, it notifies the violation and what is the action to be carried out to the correspondent component. This component will know what are the necessary steps to apply in each case.

3.5.2.4. Involved technologies

The SLA implementation can be based on two specifications that are WS-Agreement [16] and WSLA [17]. There are not full implementations of SLA standard specifications but we will show the possible alternatives that can be considered.

On one hand, WS-Agreement implementation can be found in the Cremona framework included in IBM Emerging Technologies Toolkit (ETTK) [19]. Cremona is the acronym of "Creation, Monitoring and Management of WS-Agreement" and it 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 IBM Web Services Toolkit [18] 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 the 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. Even 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.

3.5.2.5. Configuration and Deployment

There can be several configurations for deploying SLA Enforcement submodules (SLA-Controller and SLA-Decisor). This will depend on the strategy to configure 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 SLA-Controller within Akogrimo environment on each machine containing the Monitoring module, whereas SLA-Decisor will be centralized in one host of the HE (per administrative domain):

- SLA-Controller will be deployed on every machine as the Monitoring subsystem will do. This component must appear besides the monitoring to receive all measurements provided by a specific machine which is hosting the service. SLA-Controller cannot be centralized due to the fact that for each service there will be associated one of these components. To have one SLA-Controller in the same machine as Monitoring will allow avoiding unnecessary network communications. The permanent interaction between these two components does necessary to deploy them together. After the analysis of the measurements received, SLA-Controller will communicate with SLA-Decisor only when it decides is necessary.
- SLA-Decisor will be deployed once for administrative 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 EMS should do. It's not necessary to have one SLA-Decisor component for each machine because it will only receive direct communications by the SLA-Controller module, and after it must contact with 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 generation of violations.

3.5.2.6. Use case view

Next picture shows the interaction among SLA Enforcement modules and other subsystems. There are two differentiated phases as the Activation and the Service Use. Detailed use cases are provided in the Annex.

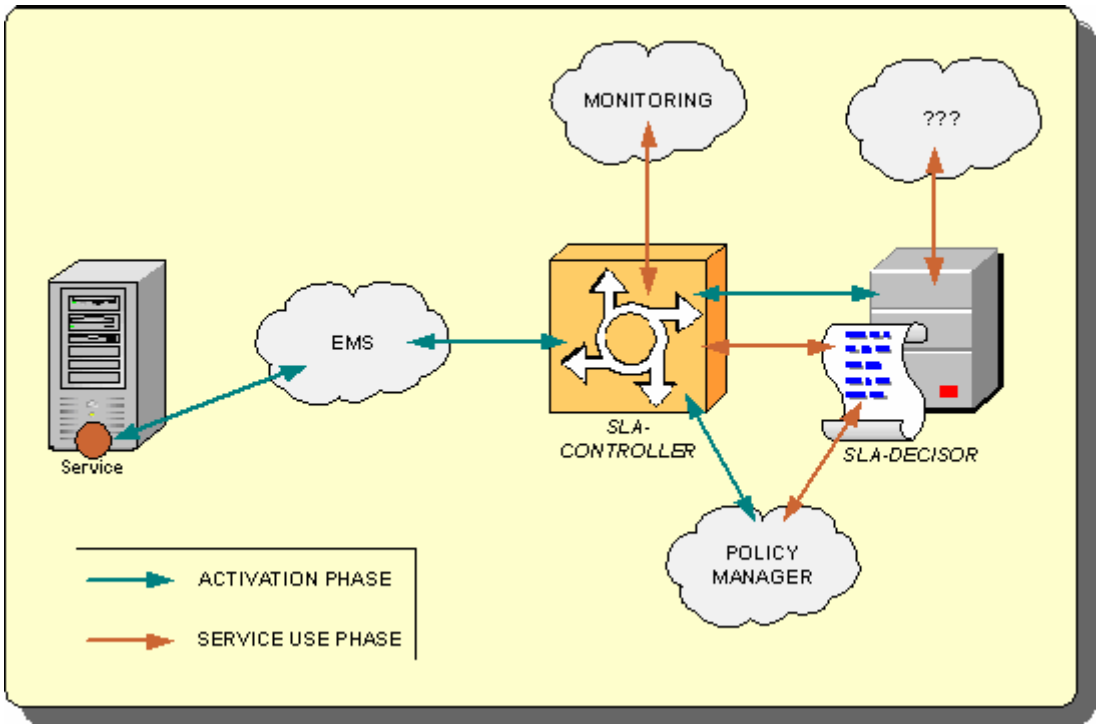

Figure 14: SLA interfaces

Next figures show the sequence diagrams within WP4.3 layer. These figures gather the interfaces described in section 3.5.2.3.

During the instantiation phase of a service the EMS must create an agent to allow controlling the status of the service. Afterwards the SLA-Controller subscribes in SLA-Decisor for sending to it the necessary notifications.

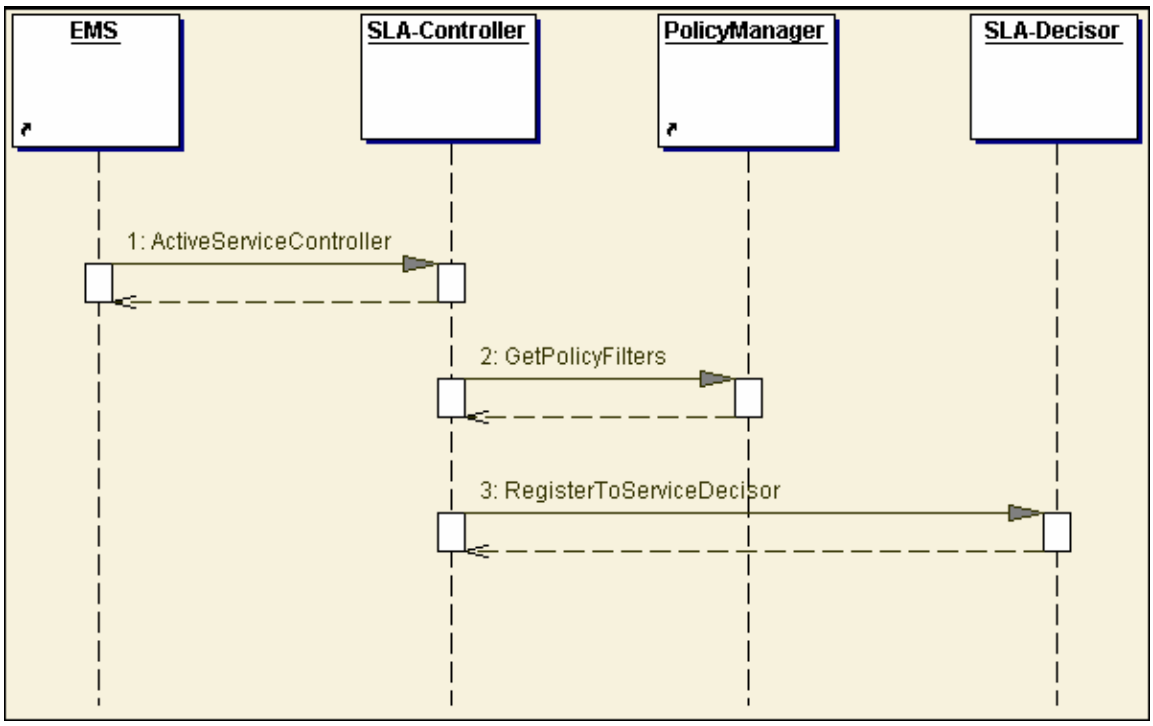

Figure 15: Sequence Instantiation

During the service use the monitoring sends periodically the measurements to the SLA-Controller that will decide if the service is running in the expected conditions (QoS parameters within thresholds). If a violation occurs, this is managed by the SLA-Decisor which makes use of Policy Manager to take appropriate recovery actions.

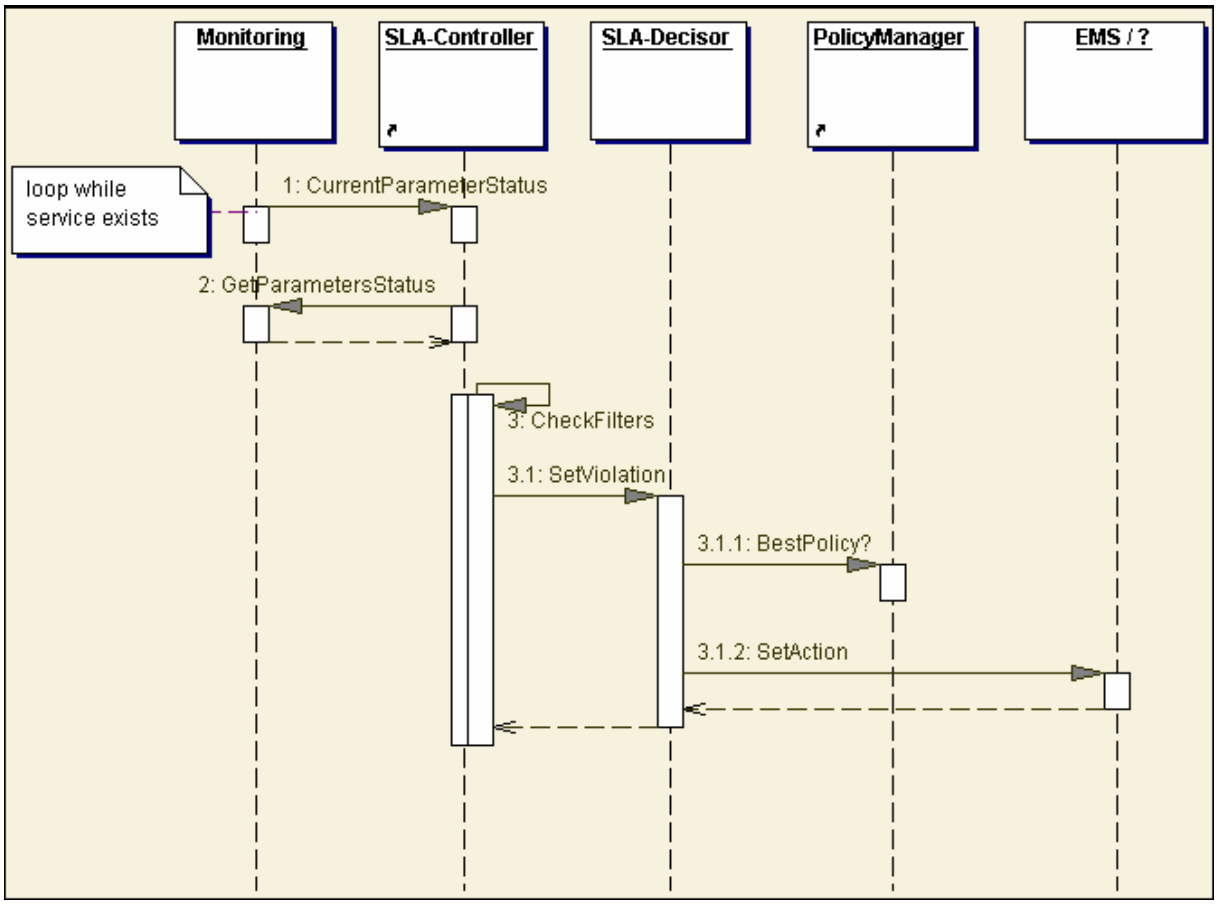

Figure 16: Sequence ServiceUse

3.6. Metering

3.6.1. Overview

3.6.1.1. Objectives

The main objective of the metering component in WP4.3 is to measure the usage of resources that belong in the Akogrimo Grid infrastructure and to communicate them to the AAA sub-system of WP4.2, so that an aggregated accounting record for the usage of resources can be made for the customers of the Akogrimo platform.

User applications have different resource requirements depending on computations performed and algorithms used in the specific application services that will be used in Akogrimo. Some applications can be CPU intensive while others can be I/O intensive, or a combination of both. For example, in CPU intensive applications it may be sufficient to charge only for CPU time whilst offering free I/O operations. This scheme is not sufficient for I/O intensive applications. Finally, we need to clarify also a wall clock metering of resource and services consumption which is applicable as a general measurement of utilization of resources which is applicable in cases of having idle resources (case of advance reservation which might have been unused).

3.6.1.2. Functional capabilities

Within the functional capabilities of a metering component we see the measurement of usage on specific resource attributes and the storage of them in an efficient way.

The metering can be applied on the "Job level" where each job is assigned a unique id and the measurements for that job are reported for each id, or "User level" when this measurement is applied for specific users based on their identities. This relation however is related to the status that will be used for the Accounting module of WP4.2. Moreover there is the "Aggregate metering" which reports the data in aggregate (summarized) form.

The Metering component interacts with the Monitoring to exchange information with respect to the execution of the various services. It feeds the monitoring component with data so that this can continue with the manageability of services when specific conditions are met (constraints levels are met, alerts are triggered).

3.6.1.3. Non-functional capabilities

The Metering component needs to provide an API through which it interacts with other components, and also for "customizing" itself with respect to what parameters it should measure.

It should be lightweight enough so that it does not provide significant overhead on the execution phase.

It should be "repeatable" and consequently reliable meaning that for the same operations it should provide as similar as possible measurements.

It should be able to address also the intermediate "middleware" services which are transparent to the end users.

Security: the system interacts only with authenticated and authorized entities.

3.6.1.4. Interaction with other components of the Layer

EMS

In order to start measuring various resources consumption and service usage it has to be asked to do so by the EMS component which is the heartbeat of the execution. The EMS manages the execution and for this reason it might need to get also feedback on specific resource consumption.

Monitoring

The Monitoring component is the overall supervisor of the execution. For this reason it has to collaborate with the Metering component for having knowledge of the execution phase and the respective consumption of resources and services usage.

WP4.2 AAA

The information that is measured by the Metering component is submitted to the AAA module which keeps the "vertical" accounting of the Akogrimo resources utilization.

3.6.2. Architectural Description

3.6.2.1. Requirements and Constraints

There are many issues that that have to be considered for the metering component.

Resource Heterogeneity: Computational resources are usually heterogeneous in that these resources may have different hardware, such as instruction set, computer architectures, number of processor, physical memory size, CPU speed, and so on, and also different software, such as different operating systems, file systems, cluster management software, and so on. The heterogeneity results in differing capability of processing jobs. An adequate metering component should address the heterogeneity and further leverage different computing power of diverse resources.

Preserving Site autonomy: Typically a Grid may comprise multiple administrative domains and each one of them shares a common security and management policy. This autonomy should not be compromised for the case of having metering information for the resource utilization.

Lightweight: When measuring resource usage, the overall performance of the system should not be influenced (at least with a significant overhead).

Ability for auditing: Measurements have to be transparent in their metrics and of course able to provide tracking logging of their behaviour. Moreover when executing twice a specific experiment, the metering component should provide under the same conditions the same measurements.

Application diversity: The problem arises because the Grid applications are from a wide range of users, each having its own special requirements. For example, some applications may require sequential execution, some applications may consist of a set of independent jobs, and others may consist of a set of dependent jobs. An adequate scheduling system should be able to handle a variety of applications.

Dynamic behaviour: In a Grid environment, dynamics exists in both the networks and computational resources. On one hand new resources may join the Grid, and on the other hand, some resources may become unavailable due do problems such as network failure. This has to be taken into consideration when metering resource utilization.

3.6.2.2. Functionality

The Metering component of WP4.3 is positioned in the Grid middleware layer. It aims to make the appropriate measurement of resource usage so that it updates the accounting module of Akogrimo AAA to proceed with the pricing (and consequently charging) of the services that have been used.

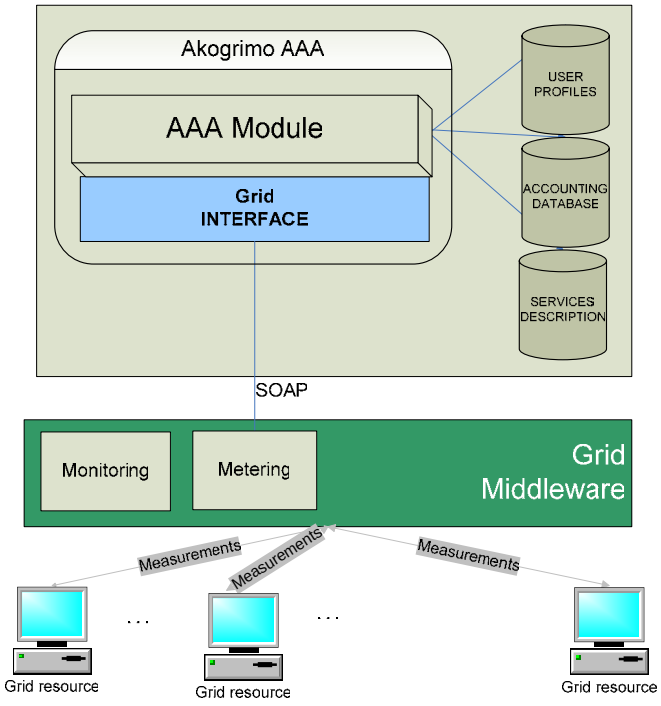

Figure 17: Metering component and AAA in Akogrimo – The vertical approach

The consumption of the following resources needs to be metered and submitted to the AAA:

• CPU - User time (consumed by user application)

• Wall clock time (time that elapsed while the job was running)

• Number of CPU nodes used

• Memory used (mainly virtual)

• Storage used

• Network bandwidth consumption

• Signals received, context switches

• Premium rate(s) for special handling (advanced reservation of resources, Higher job queue priority within a job class)

• Special Application Charges (fault tolerance, checkpointing, failure recovery, mobility of results)

• Software and Libraries accessed (particularly required for the ASP world).

Access to each of these entities can be metered individually or in combination. It does not mean that any resource provider must charge for all these list points. For instance, a site may decide that it will only charge for CPU utilization.

The resource usage metered has to be presented to the "consumer" in an understandable and decomposable fashion – the user needs to know what the measures are for using a site's resources so that an informed decision can be made before submitting a job.

3.6.2.3. Interfaces

WP4.3 Subcomponent	Interfaces with Subcomponent, WPs	Interfaces / protocols
Metering	EMS/WP4.3	SOAP
Metering	Monitoring/WP4.3	SOAP
Metering	AAA/WP4.2	SOAP

Table 6: Interfaces for the Metering component

3.6.2.4. Involved technologies

The involved technologies will be detailed in a later phase after having identified what exactly will be measured for the metering purpose of the Akogrimo services.

It is considered that the technologies used will be based on the GGF drafts:

Usage Record (UR-WG): In order for resources to be shared, sites must be able to exchange basic accounting and usage data in a common format. This working group proposes to define a common usage record based on those in current practice.

OGSA Resource Usage Service (RUS-WG): To define a Resource Usage Service (RUS) for deployment within an OGSA hosting environment that will track resource usage (accounting in the traditional UNIX sense) and will not concern itself with payment for the use of the resource.

Grid Economic Services Architecture (GESA-WG): The purpose of this working group is to define the protocols and service interfaces needed to extensibility support a variety of economic models for the charging of Grid Services in the OGSA.

Moreover, the technologies that will be used will involve operating system based libraries and functions (or even third party software packages) for the various measurements that have to be taken. Benchmarking technologies will be also considered for the identification of the capabilities and the potentialities of the resources involved.

3.7. Policy Manager

3.7.1. Overview

The Akogrimo project foresees different kinds of communications with different kinds of terminals. In this scenario there are many participants involved in the communication and some rules are necessary in order to manage them. These rules represent policies, fixed on the base of participant roles and participant services offered. The first main distinction between participants could be made on the basis of actions that they want to take within the VO. So, some participants may want to offer a service (SP) and some others may want to use/consume them (SC). The interaction between SC and SP are managed by a specific agent (e.g. EMS). This agent has to have the possibility to retrieve policy information about a particular context (i.e. action requested by a SC on a resource deployed by a SP). Agents can change their own behaviour on the basis of the policies received.

There is a need for distributed, automated management agents whose behaviour also has to change dynamically to reflect the evolution of the system being managed. Policies are a means of specifying and influencing management behaviour within a distributed system, without coding the behaviour into the manager agents. Furthermore in a large distributed system there will be multiple human administrators specifying policies which are stored on distributed policy servers. Policy conflicts can arise due to omissions, errors or conflicting requirements of the administrators specifying the policies.

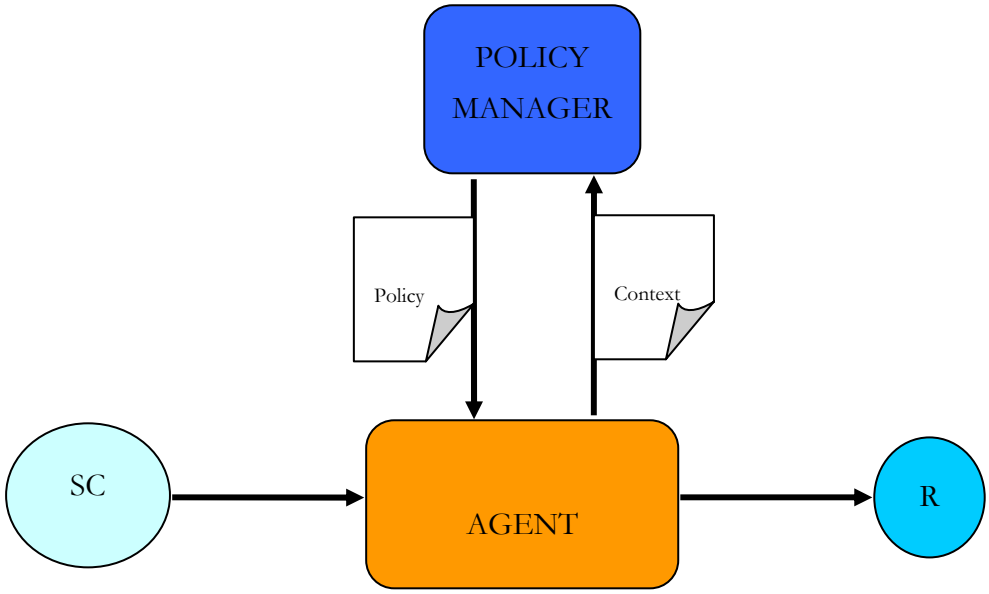

Figure 18: Policy manger overview

3.7.1.1. Policy

Policy is a very general term, meaning only a mechanism to achieve flexibility. In this context, the policy describes "*how use available resources*" and not things like "*permit/deny access to available resource*" (authorization or access control problem). For our purpose, the available resource has to be virtualized and accessed (directly or indirectly) through a grid service interface.

So, these policies provide flexibility in the resource management context, at several levels. Before the description of a Policy Model suitable for the Akogrimo objectives we briefly introduce the approach used in WS-Policy Specification.

3.7.1.1.1. WS-Policy Specification

The **WS-Policy** (draft) specification is a general purpose model to describe the policies of a Web Service. WS-Policy provides a grammar for expressing capabilities, requirements and general characteristics of ***entities*** in a Web Services based system. A policy is *basically* a ***collection of assertions*** that specifies requirements about, for instance, authentication scheme, transport protocol selection and other issues related to the wire interaction style or, in other situations, to the proper service selection and usage. Assertions are ***domain-specific semantics*** and *are*

expected to be defined in separate domain-specific specifications. In a Web Services based system a policy is used to convey conditions on an interaction between two different *WS endpoints*. A Service Provider uses a policy to exposes the conditions under which it provides the service. The Service Consumer uses the policy to decide whether or not to use the service. The policy assertions are grouped in *policy alternatives,* a way to give to the policy consumer different alternatives in the using of the service. The way used by a Service Provider to expose the policy is described in the **WS-PolicyAttachment** specification. It describes the mechanism used for associating policies to *Policy Subjects*. Four different policy subjects have been identified (in WSDL 1.1): services, endpoints, operations and messages.

In the specification two different attachment mechanisms are considered:

- the first allows to associate policies as part of the intrinsic definition of a resource, attaching the policy reference in the XML elements of a WSDL description. This mechanism leverages on the "wsp:PolicyURIs" attribute (or the equivalent <wsp:PolicyRefence> element) and the policy identification mechanism defined in WS-Policy ("wsu:ID" attribute). In the following example two policies are associated respectively to:

  A.1. the *endpoint subject* generally described collectively by the <wsdl:port>, <wsdl:portType> and <wsdl:binding> element of the WSDL description.

  A.2. the *message subject* described in this example by the "wsdl:portType/wsdl:operation/wsdl:input" element.

  To ensure that consumers of a policy-annotated WSDL document are capable of processing such Policy attachments the <wsp:UsingPolicy> element must appear in the <wsdl:definitions> element of the WSDL.

```
<definitions>
...
<wsp:UsingPolicy wsdl:rRequired="true" />
...
<portType name="myPortType" wsp:PolicyURIs="http://myHost/policies#Policy1" >
  <opearation name="aOperation" >
    <input message="aMessage" wsp:PolicyURIs="http://myHost/policies#Policy2"/>
    <output message="aResponse"/>
  </operation>
</portType>
...
</definitions>
```

Wsdl document using WS-Policy attachments

```
<wsp:policy xmlns="..." xml:base="http://myHost/policies" wsu:ID="Policy1">
  ...
  {alternatives}
  ...
</wsp:policy>
```

Policy file located at URL: http://myHost/policies#Policy1

- The second mechanism allows policies to be associated independently from their definition. In this case the subjects are explicitly indicated by means of the <wsp:AppliesTo> element using a domain expression syntax. In the specification the ApplyTo/{Any} is an extendible point of the schema. To date only the EndpointRefence domain expression is defined. Others domain expression syntax may be defined by subsequent specifications

```
<wsp:PolicyAttachement>
 <wsp:AppliesTo >
  <wsa:EndpointReference>
    ...
  < wsa:EndpointReference
 </wsp:AppliesTo >
 <wsp:PolicyReference URI=http://myHost/policies#Policy1 />
</wsp:Policyattachment>
```

Finally the PolicyAttachment specification describes how to associate a Policy through the use of UDDI registry. There are two approaches for registering policies. One used to associate policy for a specific Web Service referencing policies in UDDI entities, the second one is to register policies as tModel and referencing them in a modular way.

We can recognize some limits in the adoption of WS-Policy in the Akogrimo project:

- Firstly the policy framework mainly addresses the issues of typical B2B applications in which the Web Services components agree on common communication protocols and workflows. In this situation the policy is not used by enforcement agents but primarily by the service consumer to decide whether or not to use the service and how to use it.
- The policy attachment mechanisms is used to make clear the constraints that need to satisfy in using every service element as message exchange end so on primarily in a Web Services based system. In Akogrimo project is needed to associate policies to abstract entities such as users, groups, and resources. We need a more general purpose way to define Policy Subjects, not necessarily tied to WSDL constituents. Lastly the way of associating a Policy, attaching a reference within the WSDL constituents, could limit the modularity and on fly configuration requirements of Akogrimo system components.

3.7.1.1.2. Other policy models

In the "**Automated Policy-Based Resource Construction in Utility Computing Environment**" [32] A. Sahai et Al. pose the problem of how to construct automatically computing environment using resources restricted by policies. In this scenario we have a group of resources available to construct a computing environment; each resource has a policy attached; the policy defines rules for the usage of the resource. From these resources and related policies, the paper proposes a way to compose resource constructing a higher-level resource and to solve automatically all policy requirements at all levels.

The interesting part of this document is the language used to describe resource policies. This language allows the construction of complex expressions using operators (e.g. boolean, relation), control constructs (like if-then-else, implies), and allow the use of inheritance between policies (we can inherit a previous policy definition, avoiding to rewrite the same rules).

3.7.1.2. Objectives

Actions inside the VO have to be controlled, in a way that each participant will be subordinated to some rules in order to access resources (i.e. services) provided by other participants. In this scenario each SP can provide policies for own resources and the VO manager can provide policies for all involved VO participants. The aim of Policy Manager (PM) is to provide mechanism to hold and manage policies, sending them to consumer (in some particular cases policy requestor and policy consumer could be the same entity).

3.7.1.3. Functional capabilities

- Policies DB management. This functionality maintains the policies uploaded by a SP and in such way it should maintain some reference to high level policies, managed by VO Manager
- Policies request Service. This Service is responsible to reply/forward to policy requestor/consumer in order to provide right policy about the action scenario described by the request provided.

3.7.1.4. Non-functional capabilities

- Policies Description Flexibility. In order to satisfy the needs of various VOs. (i.e. possibility to insert, into DB, rules that involve group of resource or SC)
- Privacy and Security. DB can contain reserved information. PM should avoid providing this information to unauthorized participant.
- Grid features. In this context PM should have typical characteristic of a grid system (scalability, performance…).

3.7.1.5. Interaction with other components of the Layer

- EMS. The policy manager will communicate with the Execution Management Service in order to apply policies related to the job/service that is running. These policies are customized for each job/service and are stored in the PM repository of the local HE. A typical example of policies applied/checked at this level could be: "do not transfer data between two specific databases if are greater than xGB", "do not meter information for specific urgency issues" etc.
- EMS contacts PM through an internal component, "Query Service". EMS wants to know all the policies associated to a context and queries PM to retrieve them. To make this, EMS has to supply to PM all information related to the context (i.e. user credential and information like certificates and/or the belonging group…) to retrieve policies. PM returns to EMS all policy matching request context.
- SLA. The policy Manager will be asked by SLA to obtain the policy containing metrics mapping parameters to guarantee QoS of the service. Furthermore SLA can ask for PM to obtain policy containing action to be performed when a violation has arised.
- DataManagement contacts PM to obtain policy that limit data access.
- Security: The PM could be asked by the security component to obtain the policies related to the authorization and security level of operations and communications..

3.7.1.6. Interactions with other Layers

- VOManager. In some cases the policy manager could be informed by high level entity in order to check low level policies. In these particular cases we foresee an interaction with VO Manager, following a hierarchical architecture described in next paragraphs.

3.7.2. Architectural Description

In a large dynamically and heterogeneous environment it is better to distribute Policy Manager Systems. In the previous analysis we have to focus on two authors of policies, Service Provider and VO Manager. The architecture showed below is layered in two levels. The first one (local respect to Host environment), managed by SP, and the second one (global), managed by the VO Manager and aimed to hold VO rules.

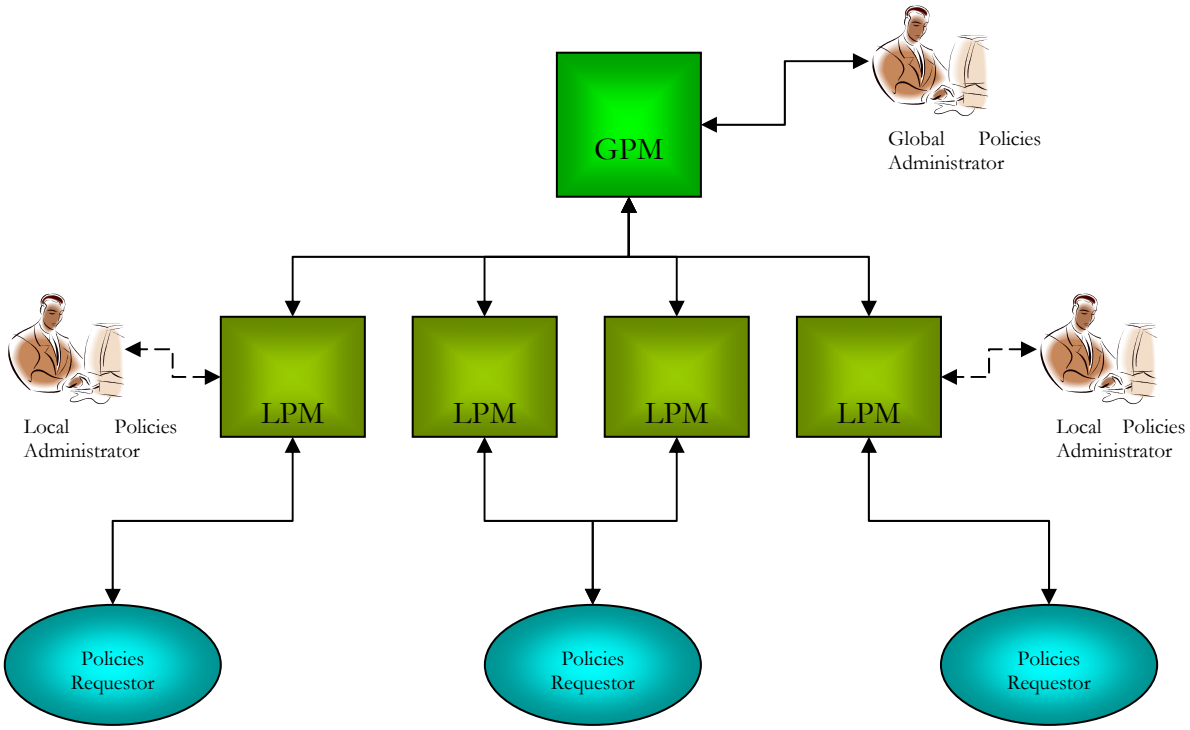

Figure 19: Hierarchical vision

The Local PM (LPM) service is delegated to serve the policy requests coming from the Policy Requestor (PR). To do this the LPM could query policies to GPM in order to respect global rules (in this situation LPM take the role of the PR implementing a hierarchical architecture).

In a VO, LPM and GPM could hold policies applying to distinct Management Domains. In this way we would avoid conflicts between LPM policies and GPM policies. For example, the LPM could maintain technical policies and GPM could maintain administrative policies.

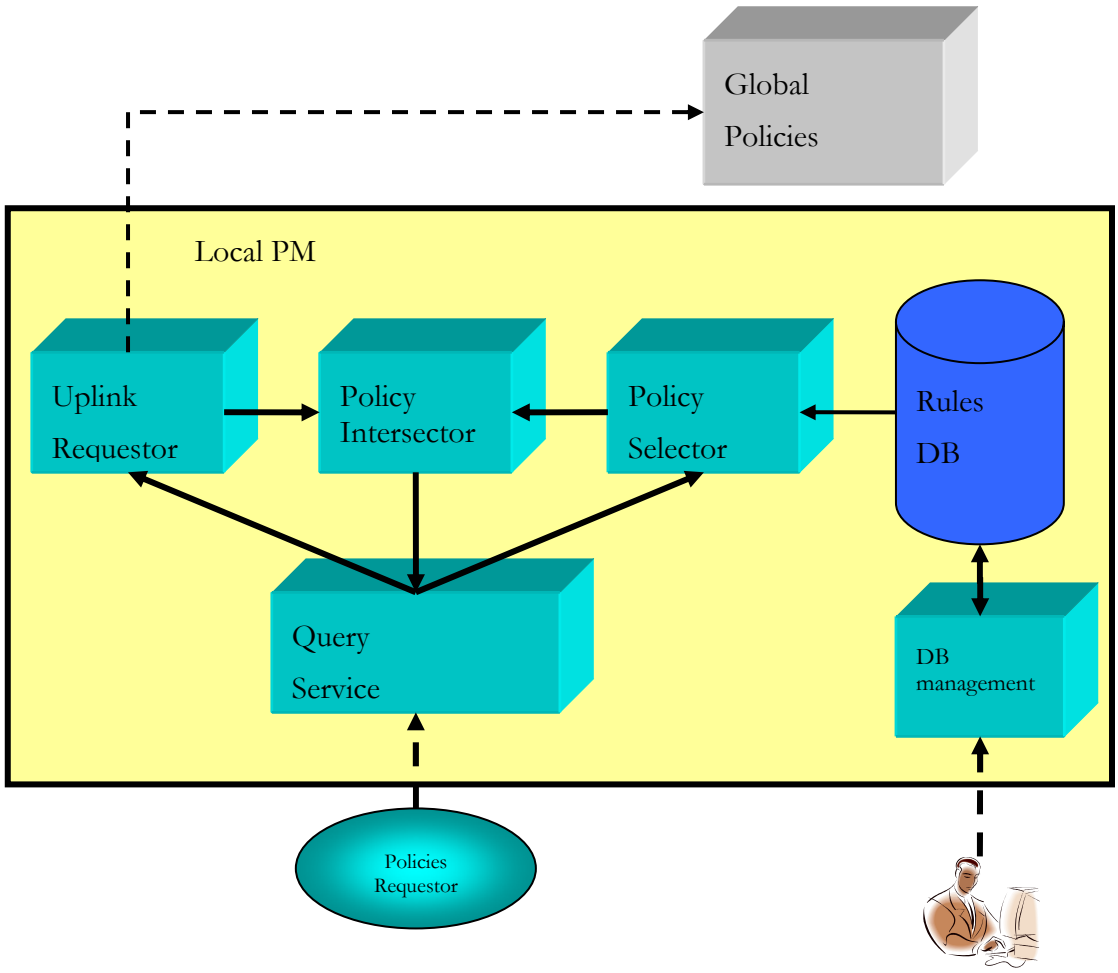

Figure 20: Entities involved during policy request

In the Figure 21, starting from a generic vision of the PM framework, we distinguish components involved in our policy management process (grouped by the red line).

- *Policy Store* is required to hold and make available the various policies. The store must be accessible to the editor and decision agent. In our architecture we have a policy store distributed on two levels, local and global. In the local policy store we maintain the technical policy. In the global policy store we maintain the administrative policy concerning the whole VO.
- *Policy Owner* is the owner of an entity, whose interests are being protected by the policy. The Owner, with suitable credentials, can write a policy and store it in the policy store.
- *Policy Editor* is the user interface that permits to write a policy and manage all policies in the store.
- *Policy Language* will be used by the editor to create a policy. This language has to be understood by all components that interact with PM. Furthermore as such language has to be extensible to satisfy the need of dynamic grid environment.
- *Decision Agent* is an entity that, taken the context, makes a query request to the PM, it receives the policy related to this context. Then the decision agent processes the policy and takes a decision about the action to be performed.

- *Enforcement Agent* is an entity that handles all operations to runs the action specified by Decision Agent. In our case will be the policy consumer.

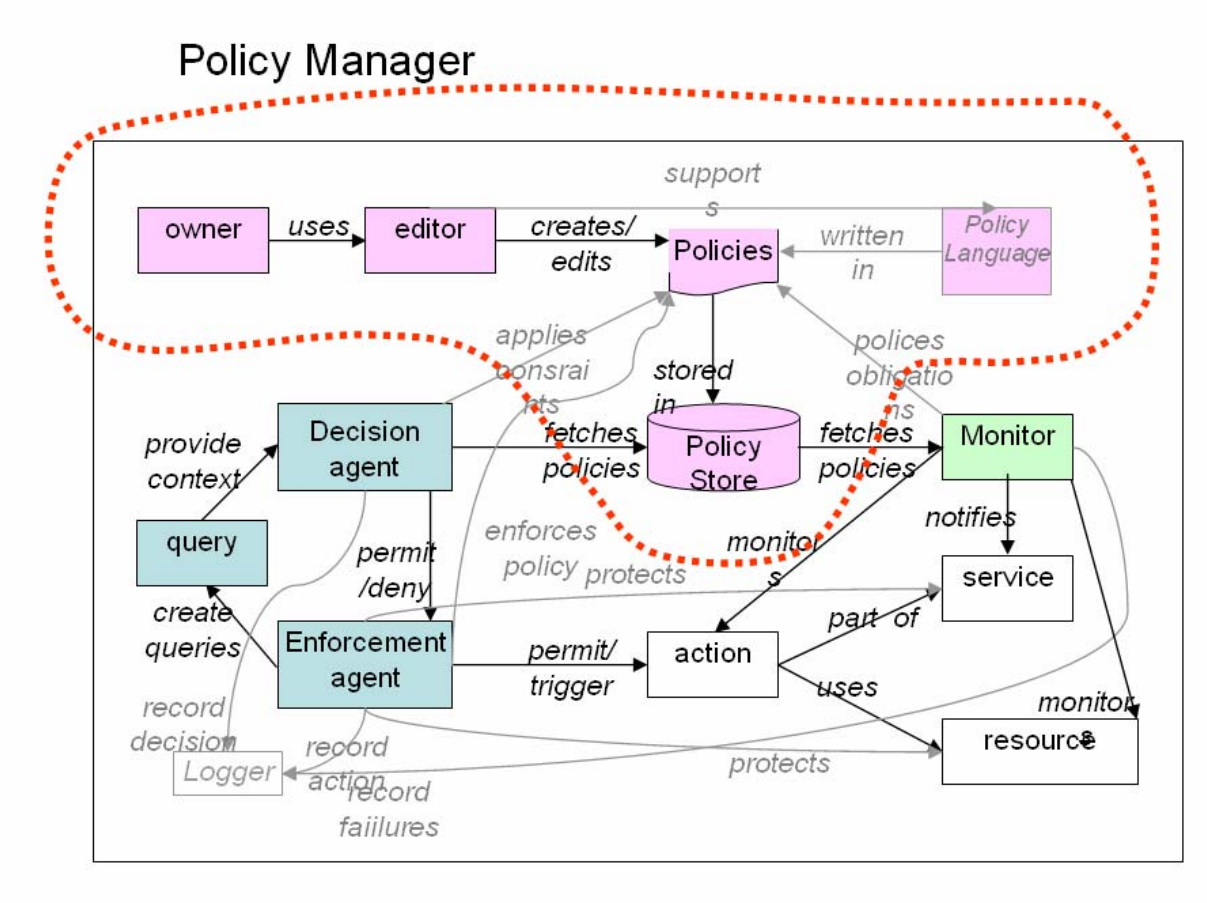

Figure 21: Mapping between our PM and a general PM framework

The use of the policy has the purpose to supply "information" to others components which have to take a decision, namely how to perform an action required by a user on a resource.

In the policy definition process two different sets of information have to be provided, *Context Pattern* and *Statement Information* as shown in the figure below (Figure 22).

1. Context Pattern is a set of information that defines the context to which a policy has to be applied. In terms of WS-Policy the context pattern determines a set of policy subject to which the policy is associated. Different languages have been proposed to describe the subject. In the section A.6 some examples are provided based on a LDAP Filter grammar. Basically, every abstract entities have described by means of a pair of attribute name and attribute value combined with logical AND, OR, NOT operator.
2. Statement Information is a set of information returned to the agent by the PM. This information is provided when the context described by the agent matches the Context Pattern. The meaning of this information is known just to the agent that uses it and not to the PM. The set of statements returned has obtained matching the context provided by the agent against the context pattern used to attach the policy. A match occurs when the policiy subject identified by the current context belongs to the set of subjects defined in a context pattern.

As agents, calling PM, can be several and of different type and each need particular information, then the Statement Information can be divided in *Statement Domains*. In this manner, an agent can retrieve from the matching policy just specific information for its domain.

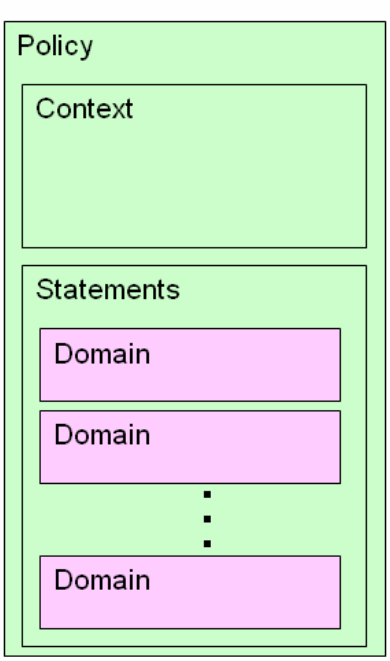

Figure 22: Context and statements

3.7.2.1. Requirements and Constraints

Since policies influence the "behaviour" of the VO, the security of policy management is an important aspect to be considered. Also the policies itself are responsible for the security management. In the design and implementation of the Policy Framework, need to avoid unintentional and/or malicious actions made by VO actors that could compromise the functioning of the VO.

Topics that need attention are:

- Privacy: policy information passing through the network should be ciphered to avoid sniffing tampering; also data access to DB (Policy Store) should be made accurately.
- Authentication: different policies are applied to different subject, therefore it's important to identify correctly the entities to which policies must be applied.

Another requirement to be considered is the "extensibility" of the policy framework: a very dynamic Grid VO environment requires a dynamic policy management. The policy management system should be able to adapt itself to changes of VO administrator requirement, without the need to be redesigned. To reach this goal, the choice suitable to modern technologies can be the usage of language based on XML. These allow a clear and formal definition of the syntax maintaining also extensibility.

3.7.2.2. Functionality

The policy manager architecture foresees the following functionality:

- DB management

Allows to add/remove DB policies information. This functionality provides an interface used to manage policies stored in DB.

- Policy Selection
  - Searches in DB the policies that are involved in a context, described by request performed by PR.
- Uplink Request
  - Retrieves the Global policies from GPM. The request is based on the request performed by PR.
- Policy Intersector
  - Collects the policies returned from the policy selection process (local and global) and merges them in a single policy response.
- Query Service
  - This functionality exposes an interface used by PR to submit query to PM System.

3.7.2.3. Subcomponents Interaction

In Figure 23 we show an interaction in the policy query process. This interaction involves a Policy Requestor, a Local Policy manager and a Global Policy Manager. When the requestor asks for a policy to LPM, start the policy selection process inside the PM. If the LPM has configured with an uplink to a Global Policy Manager, the query has forwarded to it. While the LPM waits the policy from the GPM runs the policy selection in the local DB. When all policies have retrieved, will be merged in a single policy and returned it to the requestor (in this case the policy requestor and the policy consumer are the same entity).

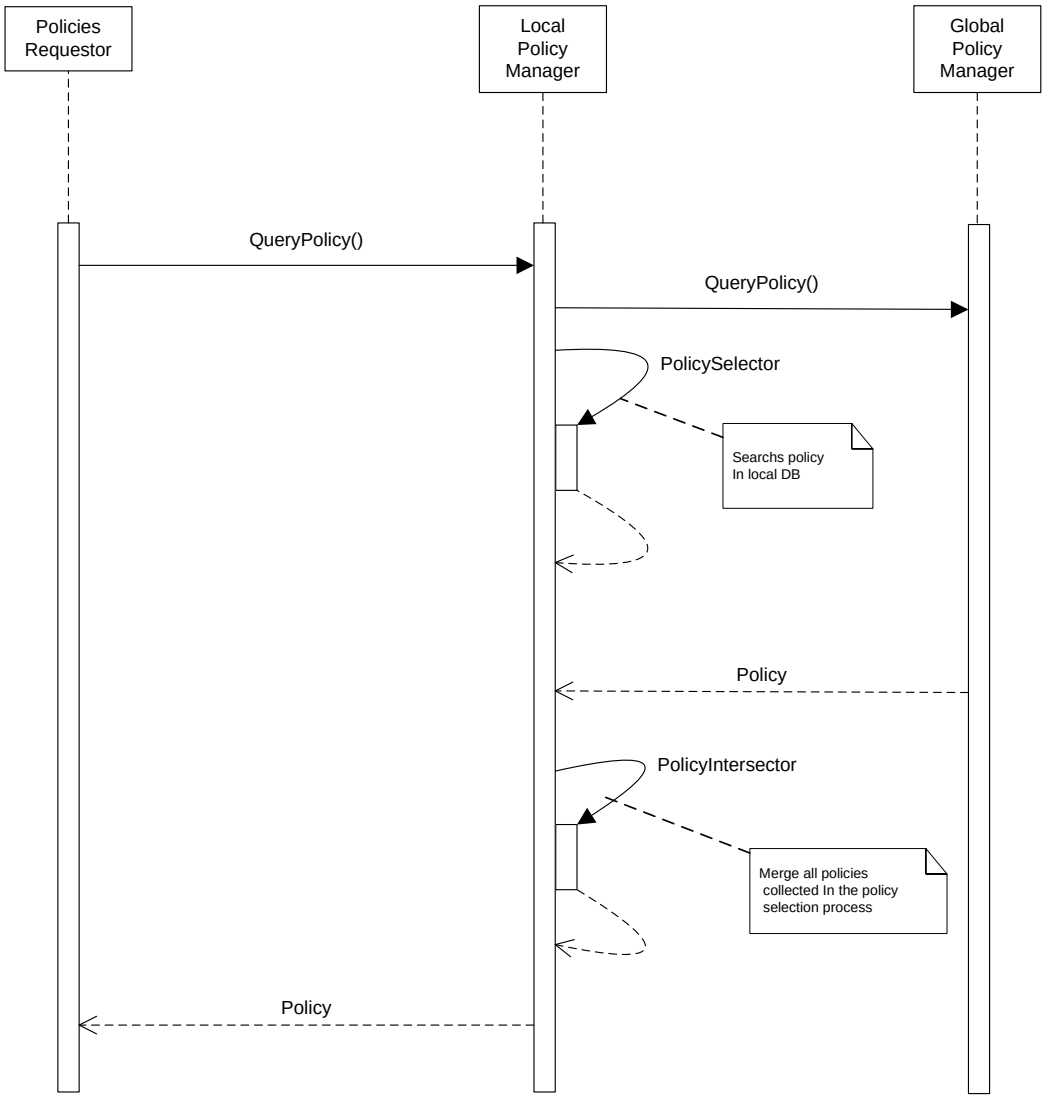

Figure 23: Policy request sequence

3.7.2.4. Interfaces

WP4.3 Subcomponent	Interfaces with Subcomponent, WPs	Interfaces / protocols
LPM	GPM	SOAP over HTTP Interface: • *QueryPolicy*: Retrieve policy related to a specific context
GPM	VO Manager	SOAP over HTTP Interface: • *AddPolicy*: Add a policy • *RemovePolicy*: Remove a

		policy • *GetPolicy*: Read a policy • *SetUplink*: Configure a reference to a policy manager at a higher level.
LPM	Policy Requestor (e.g. EMS, SLA-Controller…)	SOAP over HTTP Interface: • *QueryPolicy*: Retrieve policy related to a specific context

Table 7: Interfaces for the Policy Manager

3.7.2.5. Involved technologies

The interactions between elements collected inside the policy manager architecture involve web services and repository to maintain data. So, the main technologies that could be involved are:

- Standard driver to connect databases (ODBC, JDBC…).
- Transport technologies (HTTP, SOAP over HTTP, SOAP over UDP...).
- Policy Description Language (XML, LDAP filter…).
- GUI framework to define Policy Editor

3.7.2.6. Configuration and Deployment

The elements involved in the policy manager architecture are represented by services. So we need some containers that will contain them and some frameworks that manage these resources (i.e. WSRF resources). On the base of underlying WSRF framework (like WSRF.Net, GT4…) there will be specific steps to configure and deploy services.

Particularly the PM has composed by a modular component, LPM that can be configured in hierarchical structure. In order to setup a two level hierarchy (LPM and GPM) described above, the uplink reference in each LPM has to be configured.

3.8. Security

In a dynamic environment such as in the Virtual Organization each entity is bound to its own administration domain where it is identified. In fact, the VO isn't in charge of maintaining information about all entities involved within the VO, its effort is reduced to deal with the representative entities for administration domain (i.e. services provider, network provider, etc.). The VO thus identified will provide the dynamicity and mobility required by enabling security at local level as well as at VO level.

In Akogrimo we want to define a common mechanism to ensure the co-existence of several independent infrastructures within the VO and to also address the nomadicity and mobility of users. This mechanism must provide a transparent and complexity-reduced way for identifying each entity (e.g. user or resource) intra/inter-domain by relying on functionality provided by

lower layers. At this aim, the SAML identification mechanism provided at network layer can be helpful to Akogrimo's entities for allowing the authentication of participants at grid level.

3.8.1. Overview

Security is an issue crossing all layers (from network to application support layer) in the Akogrimo architecture. At this level security is concerned with the right service use assurance by reliving each improper access and undertaking adjustments actions. The federation of services belonging to several administration domains leads to a shared mechanism for enacting security policies on user's requests. These policies are established by service developer before the service is made available for use (i.e. service publishing).

In Figure 24, a general sight of security services stack, crossing network to grid application support layer, is visible.

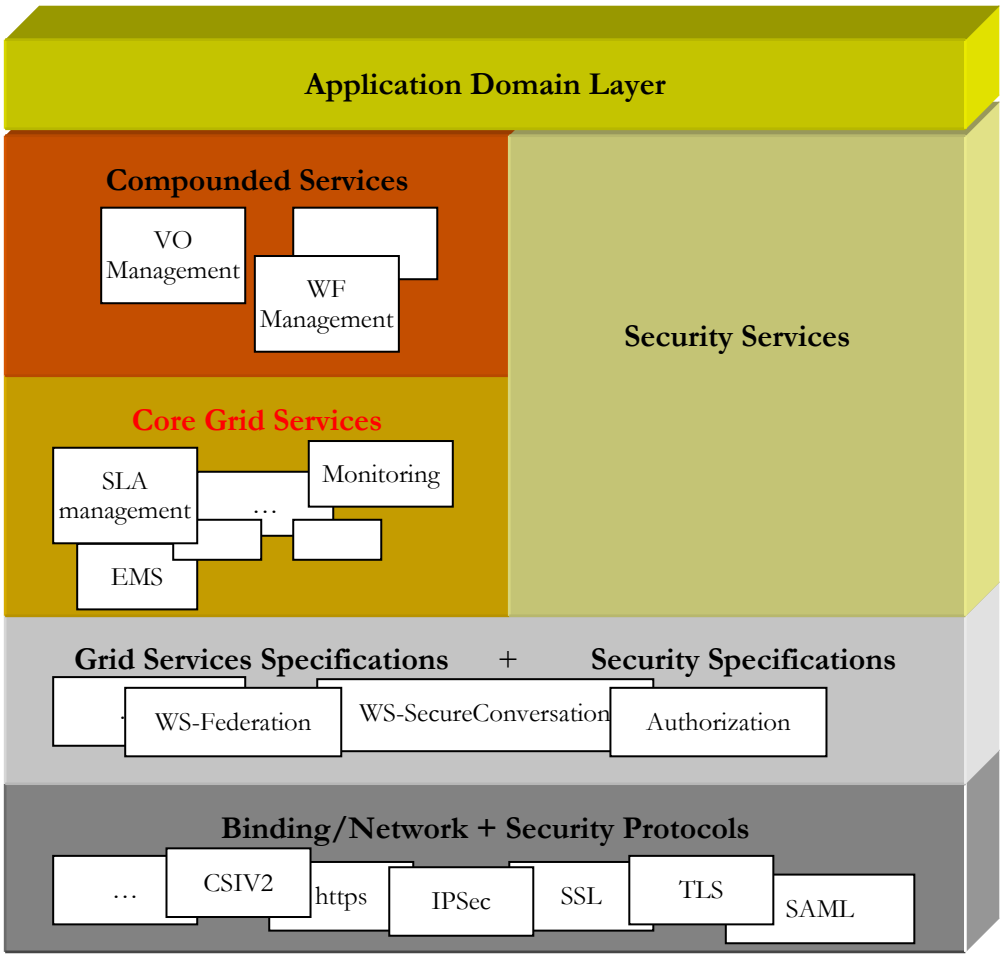

Figure 24: Akogrimo security stack

In order to support security, binding/network level must be extended with Security Protocol like IPSec, SSL and TLS for the network level, and https CSVIV2; XML Digital Signature, XML Encryption, etc., for the binding layer. This layer provides a secure mean of communication by providing secure communication protocol that should be used by upper layer (unless we won't a secure communication).

The security specifications, standards and mechanisms hinted so far provide only security between intermediaries and only for the communication channel, but no end-to-end. The upper layer provides a common infrastructure that can be useful for defining services. In that layer Grid services specifications (e.g. WSRF) should be joined to Security Specifications or Models (e.g. WS-Security [22], WS-Policy [23], WS-Federation [24], WS-SecureConversation [27], WS-Privacy, WS-Trust [25] or Liberty Alliance model [29], WS-Authorization , etc.) in order to provide a basic secure behaviour for grid services. In particular the WS-SecureConversation specification is mainly taken into account in order to support end-to-end security. WS-SecureConversation defines extensions built on WS-Security and WS-Trust to provide secure communication across one or more messages. This specification describes how to manage and authenticate messages exchanges between parties including security context exchange and establishing and deriving session keys.

Instead, WS-Security provides end-to-end message security and it is used in Akogrimo for secure communication among Grid services when using SOAP messages. WS-Security defines a SOAP Security Header to contain security elements. It includes:

- Security tokens: Akogrimo uses SAML tokens. Akogrimo components may insert SAML tokens for user authentication and authorization at grid services.
- Signature elements: XML-Signature is used to protect SOAP messages. XML-Signature provides message integrity, message authentication and non-repudiation.
- Encryption elements: XML-Encryption is used to encrypt the SOAP message. This provides message confidentiality.

Furthermore in our grid infrastructure we should take into account the concept of WS-Federation and WS-Trust; because they enable other use cases where indirect trust relations are established via a trusted third-party, or a resource needs to access another resource on behalf of the initial requestor and will require delegation from the initial requestor. Delegation is an important mechanism for Grid applications and for enabling user initiated dynamic job/task execution. In our case we should have a direct interaction between services or using a third part to communicate with them (e.g. EMS).

3.8.1.1. Objectives

Security services are to facilitate the enforcement of the security-related policy within a (virtual) organization. In the OGSA model, security architectural components have to support, integrate and unify popular security models, mechanism, protocols, platforms, and technologies in a way that enables a variety of systems to interoperate securely. In Akogrimo the focus on mobile and nomadic aspect of the Virtual Organization leads to a security infrastructure layered on network, channel and message levels and to make uniform the authentication process based on emerging technologies and mechanisms. This architecture at this layer must be able to support integration with security capabilities and constraints provided at lower layer. The identity management envisaged at network level creates the framework of agreements, standards and technologies necessary to make identity portable across different administrative domains. This identity management is taken into account for supporting user/service authentication and authorization at Grid infrastructural level as well.

This means that the architecture should be *implementation-agnostic*, so that it can be instantiated in term of any existing security mechanism; *extendible*, so that it can incorporate new security services as become available; and *integrable* with existing security services. Furthermore in order to free as much as possible the user from any implementation detail it should leverage a high level virtualization.

3.8.1.2. Functional capabilities

The basic Akogrimo security model must address the following security disciplines [36]:

- Authentication. Authentication means the capability of identifying entities. Both users and services require authentication in a secure environment. In order to achieve this target security services, at this layer, exploit the authentication mechanism provided by underlying layers.
- Delegation. Provide facilities to allow for delegation of access rights from requestors to services, as well as to allow for delegation policies to be specified. When dealing with delegation of authority from an entity to another, care should be taken so that the authority transferred through delegation is scoped only to the task(s) intended to be performed and within a limited lifetime to minimize the misuse of delegated authority.
- Credential Lifespan and Renewal. In many scenarios, a job initiated by a user may take longer than the life span of the user's initially delegated credential. In those cases, the user needs the ability to be notified prior to expiration of the credentials, or the ability to refresh those credentials such that the job can be completed.
- Authorization. Allows for controlling access to Akogrimo services based on authorization policies (i.e. who can access a service, under what conditions) attached to services. Also allow for service requestors to specify invocation policies (i.e. who does the client trust to provide the requested service).
- Privacy: allows both a service requester and a service provider to define and enforce privacy policies, for instance taking into account things like personally identifiable information (PII), purpose of invocation, etc. (Privacy policies may be treated as an aspect of authorization policy addressing privacy semantics, such as information usage rather than plain information access.)
- Confidentiality: enables only the intended recipients to be able to determine the contents of the confidential message. It protects the confidentiality of the underlying communication (transport) mechanism and the confidentiality of the messages or documents that flow over the transport mechanism in Akogrimo's network infrastructure.
- Message integrity: ensures that unauthorized changes made to messages or documents may be detected by the recipient. The use of message or document level integrity checking could be determined by policy.
- Policy exchange. Allow service requestors and providers to exchange dynamically security (among other) policy information to establish a negotiated security context between them. Such policy information can contain authentication requirements, supported functionality, constraints, privacy rules etc.
- Non Repudiation: ensures that a message cannot later be denied by one of the entities (sender and receiver) involved in a secure communication

A Grid service must be able to define or publish the Quality of Protection (QoP) it requires and the security attributes of the service. Aspects of the QoP include security bindings supported by the service, the type of credential expected from the service requestor, integrity and confidentiality requirements, etc.

3.8.1.3. Non-functional capabilities

- *Minimized credential lifetime for delegated authorities*. Reducing time to perform task(s) for the *delegation* of access rights from requestor to services in order to minimize the misuse of delegation authority.
- *Manageability*. Explicitly recognize the need for manageability of security functionality within the OGSA security model. For example, identity management, policy management, key management, and so forth. The need for security management also includes higher-level requirements such as anti-virus protection, intrusion detection and protection, which are requirements in their own rights but are typically provided as part of security management
- *Integration with heterogeneous legacy infrastructure*. The virtualization of legacy service definition enables Akogrimo to use a standardized ways of enabling the federation of multiple security mechanism.
- *Trust relationship*. Grid services requests can spam multiple security domains, trust relationships among these domain thus plays an important role in the outcome of such end-to-end traversals. The trust relationship problem is made more difficult in Akogrimo environment by the need to support the dynamicity and mobility of transient services.

3.8.1.4. Interaction with other components of the Layer

Security is a vertical issue not only in this workpackage but also between different workpackages. At the moment we aren't able to fulfil this section, but sure if we want to address security between core services (like EMS, Monitoring…) identified in this layer, we should foresee secure interactions between them, not only between business services.

3.8.2. Architectural Description

At Akogrimo's infrastructure layer we can distinguish several security services. Likewise any other grid service, security services should be exposed as web services (i.e. with a WSDL interface) and should expose functionality while hiding implementation details. No specific security technology should be hard-coded in order to secure these services. How shown in the Figure 25 the principal service is the Enactment Authorization Service. Moreover, for completeness, some services not defied at this layer (e.g. A4C component) or defined as other subsystems at the same layer (e.g. policy service) are depicted in this figure since they provide capabilities (e.g. security policies) or their functionality are utilized at this level to meet all security needs. These predefined security services are coloured in grey. All the other security services at this level depend on the Policy Service being in charge of security decisions. The other services at this level represent the accomplishment of decisions made by the Policy Service. More in detail:

- Policy Service: actions inside every Hosting Environment have to be controlled, in such way, each participant that will be subordinated to some rules in order to access HE resources (i.e. services). This component, indeed, doesn't belong to security services individuated here but its functionalities are used here: this component is nested within a set of security services at infrastructural level since security is dealt with policies and since it is able to take the current context and requested action, returning a decision to (for our purposes) Enforcement Authorization Service and/or Policy Service whether the action cannot (or "partially") be carried out.

- Enforcement Authorization Service: is the agent in charge of actualizing an action issued by the Policy Service. Its tasks are to stop, permit or trigger actions on a given resource or service.
- Enforcement Privacy Service: Privacy Services has functionalities similar to Enforcement Authorization Service. Privacy policies may be treated as an aspect of authorization policy addressing privacy semantics, such as information usage rather than plain information access.
- A4C: is the component enabling the user (and service) identification by coping with the different user's administration domain. This component is provided at network level since provide a framework for supporting the Mobile Grid within security environment as well.
- Security Designer: is the component that allows defining security policies for services. These policies will be applied for regulating service and resource usage within the VHE as well as outer this VHE.

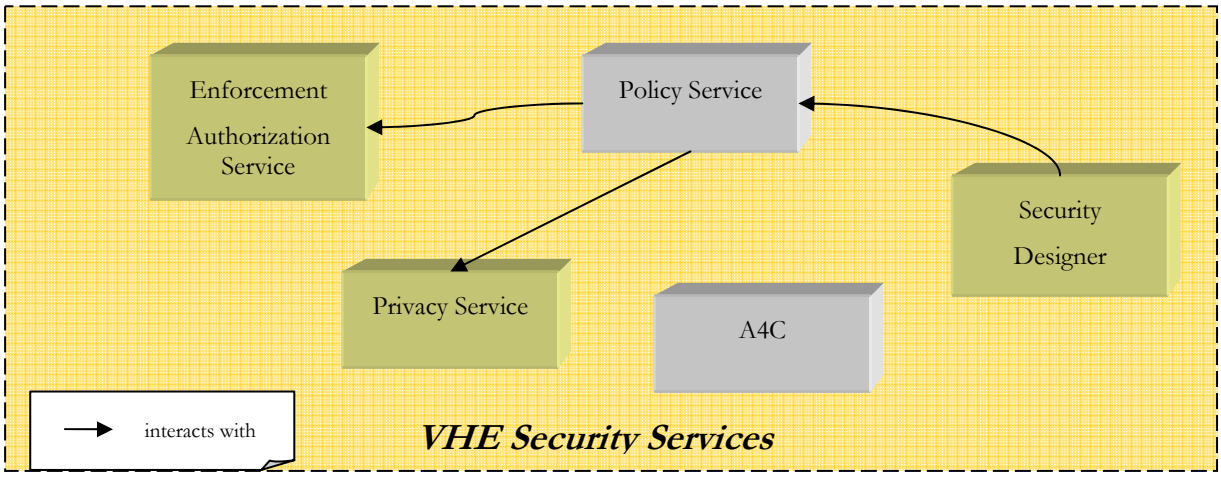

Figure 25: Security Services for Akogrimo's security services

3.8.2.1.1. Scenario: User Authentication and Authorization

The authentication phase is based on authentication mechanisms that have been developed for the network middleware layer of Akogrimo. These mechanisms take advantage of A4C components for authentication of users and services (and device as well) and provide these capabilities to upper layers by interfacing with them.

In Figure 26 the main process for authorization is depicted. The network middleware layer is in charge of user authentication as soon as he requires access to network. After this phase, a SAML token asserting the network session approval is assigned to him so that he is also enabled (obviously if he has the rights) to access Grid services. In the figure, the user hosted in HE1 requires a service S hosted in HE2 by invoking its functionalities in a SOAP request including the SAML token. Before service S checks user's rights, S has to communicate with A4C component to check if user was authenticated by his home network and to guarantee access control for network layer services (communication inter-A4Cs components of Service's and User's domain is needed for this purpose). By following the OGSA purposes, at grid level the communication between A4C and the other security services must occur by means of SOAP protocol. At this aim, the A4C component will be virtualized as a grid service providing a standard interface for accessing (i.e. WSDL). These first steps conclude with SAML assertion returned to service S.

To be noted that in the picture the call-outs are made from within the stub of the required service, and outside of service implementation totally transparent with it. In such a way, the service developer can reduce his efforts mainly to service functionalities instead of taking care of security issues.

Besides those SAML assertions, the service might also request the User's Profile stored in the user's home or temporary domain. This information might also be used for taking authorization decisions by asking to Policy Service if user has the rights to access functionalities of service S. The Policy Service is general service in charge of checking the user rights by means of policies defined on the Grid Service S. In the case of some violation the Enforcement Authorization Service is notified in order to understand if the violation will imply some security actions. Moreover, the interaction with Privacy Service is request for ensuring that incoming requests make claims about the sender's adherence to services' privacy policies issued by the service's organizations.

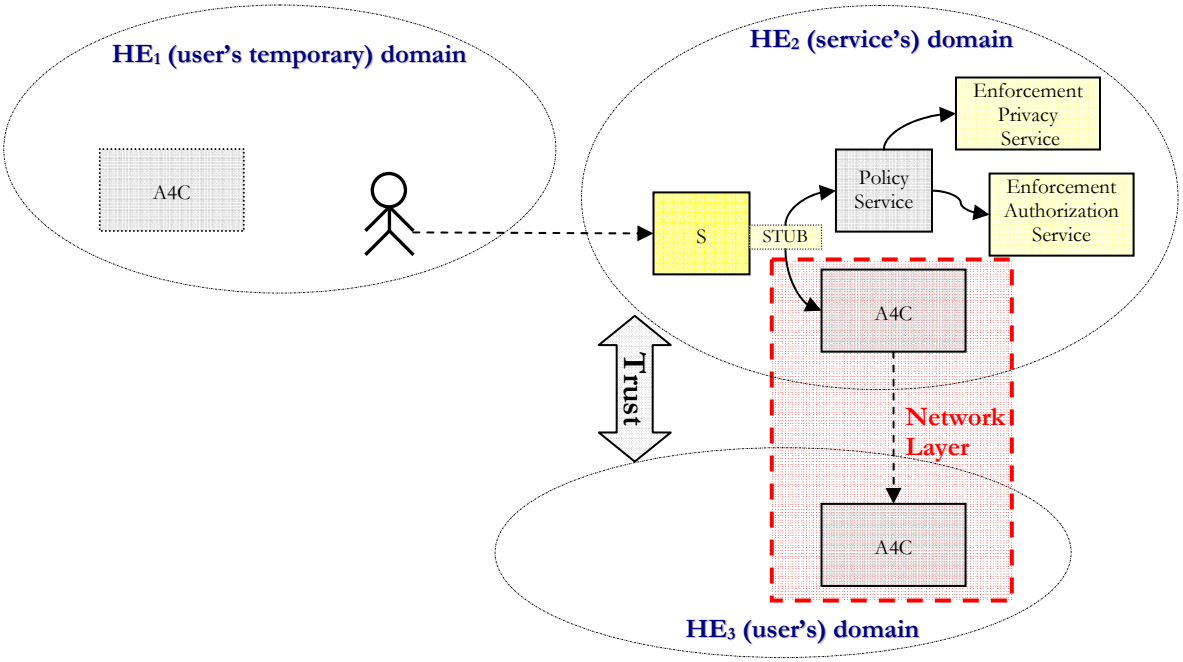

Figure 26: User Authorization

3.8.2.2. Requirements and Constraints

Services that traverse multiple domains and hosting environments need to be able to interact with each other, thus introducing the need for interoperability at multiple levels:

• At the **protocol level**: we require mechanisms that allow domains to exchange messages. This can be achieved via SOAP/HTTP, for example.

• At the **policy level**: secure interoperability requires that each party be able to specify any policy it may wish in order to engage in a secure conversation and that policies expressed by different parties can be made mutually comprehensible. Only then the parties can attempt to establish a secure communication channel and security context upon mutual authentication, trust relationship and adherence to each other's policy.

• At the **identity level**: we require mechanisms for identifying a user from one domain in another domain. This requirement goes beyond the need to define trust relationships and achieve

federation between security mechanisms (e.g. from Kerberos tickets to X.509 certificates). Irrespective of the authentication and authorization model, which can be group-based, role-based or other attribute-based, many models rely on the notion of an identity for reasons including authorization and accountability. It would be nice if a given identity could be (pre)defined across all participating domains.

3.8.2.3. Functionality

Like any other service, security services should be exposed as web services (i.e. with a WSDL definition) and should expose functionality while hiding implementation details. No specific security technology should be hard-coded in order to secure these services. Services must be secured using the Grid security model (e.g. a service request must be protected using WS-Security).

The Akogrimo security infrastructure may use a set of primitive security functions (suggested in the frame of OGSA, as well) in the form of service themselves and implement some others to achieve specific purposes.

3.8.2.4. Involved technologies

The security infrastructure is implemented following WSRF specification and then the communication will be made using SOAP and HTTP protocols. Furthermore we will use some specific features of WS-* related to secure management.

3.8.2.5. Configuration and Deployment

The configuration and deployment mechanism is container specific, in the sense that when we will decide what grid service container we need, then we will deploy secure services following specific schema. Probably we should use WSRF.NET in order to exploit WSE features [20].

4. Justification of the selected architecture

In the previous sections we have presented the adopted architecture for the WP4.3 Grid Infrastructure Services Layer and its positioning in the broader scope of the Akogrimo project. This architecture has been based on the OGSA framework since the requirements that it imposed for Grid related functionality fall into the requirements that an OGSA based architecture aims to fulfil.

The key benefits of implementing the OGSA based architecture are among others:

- It provides the Web service framework which is a simple mechanism for applications to become services accessible by anyone, anywhere, and from any device.
- It allows for the interoperability in such complex and mobile environments that the Akogrimo project will focus on.
- Enables dynamic location and invocation of services through service registries.
- Enables collaboration with existing applications that are modeled as services to provide aggregated Web services.
- It handles service-based application connectivity facilitating intra-enterprise and inter-enterprise communication with respect to predetermined Service Level Agreement terms
- Allows for self-manageability, ease of use, vertical QoS and security

5. Conclusions

In this document we have described the architecture of the WP4.3 Grid Infrastructure services layer. This architecture is based on the OGSA draft specification and as such it has been determined to fulfil the specific requirements of a Grid environment. The partners of the Akogrimo consortium that are involved in this WP have identified a set of building blocks that will gather all the requested functionality in order to bring Grid services into the framework of the project.

The document will be used as the basis for the implementation phase which will lead to the deliverable D432 "Prototype" as well as for the Version 2 of the document in the second cycle of the Akogrimo project.

References

[1] Open Grid Services Architecture draft specification. Available at - https://forge.gridforum.org/projects/ogsawg/document/draft-ggf-ogsa-spec/en/13/draft-ggf-ogsa-spec.doc

[2] Web Services Resource Framework, http://www-106.ibm.com/developerworks/library/ws-resource/

[3] The Globus Toolkit developer's guide, http://www.globus.org/toolkit

[4] WSRF.NET developer's guide, http://www.cs.virginia.edu/~gsw2c/WSRFdotNet/WSRFdotNet_programmers_reference.pdf

[5] I. Foster, "What is the Grid? A Three Point Checklist", GRID Today, July 20, 2002

[6] A. Litke, D. Skoutas and T. Varvarigou "Mobile Grid Computing: Changes and Challenges of Resource Management in a Mobile Grid Environment", presented in Workshop: "Access to Knowledge through Grid in a Mobile World", PAKM 2004 Conference, Vienna

[7] Jay Unger, Matt Haynos "A visual tour of Open Grid Services Architecture: Examine the component structure of OGSA" IBM Whitepaper Aug 2003 (available at http://www-128.ibm.com/developerworks/grid/library/gr-visual/)

[8] Web Service – Grid Resource Allocation and Management http://www-unix.globus.org/toolkit/docs/3.2/gram/ws/developer/GT_3_2_GRAM_files/frame.html

[9] Web Service – Addressing. http://www.w3.org/Submission/ws-addressing/

[10] Web Services Distributed Management: Management Using Web Services (MUWS 1.0) Part 1 http://docs.oasis-open.org/wsdm/2004/12/wsdm-muws-part1-1.0.pdf

[11] Web Services Distributed Management: Management Using Web Services (MUWS 1.0) Part 2 http://docs.oasis-open.org/wsdm/2004/12/wsdm-muws-part2-1.0.pdf

[12] A Grid Monitoring Architecture (http://www-didc.lbl.gov/GGF-PERF/GMA-WG/)

[13] Modeling Stateful Resources With Web Services http://www-106.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf

[14] Web Service Base Notification http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-03.pdf

[15] Web Services Description Language (WSDL) 1.1 http://www.w3.org/TR/wsdl

[16] Web Services Agreement Specification (WS-Agreement) https://forge.gridforum.org/projects/graap-wg/document/WS-AgreementSpecificationDraft.doc/en/10

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

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

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

[20] http://www.ggf.org/documents/GFD.30.pdf

[21] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wse/html/1357a513-229b-43ca-b2fb-f85fa3f1e4a0.asp

[22] Web Services Security, http://www-106.ibm.com/developerworks/webservices/library/ws-secure/

[23] Web Services Policy, http://www-128.ibm.com/developerworks/library/specification/ws-polfram/

[24] Web Services Federation Language (WS-Federation) Version 1.0 - http://www-106.ibm.com/developerworks/webservices/library/ws-fed/

[25] Web Services Trust Language (WS-Trust) Version 1.0 - http://www-106.ibm.com/developerworks/library/ws-trust/

[26] Web Services Security Policy Language (WS-SecurityPolicy). Version 1.0. http://www-106.ibm.com/developerworks/library/ws-secpol/

[27] Web Services Secure Conversation Language (WS-SecureConversation). Version 1.0 - http://www-106.ibm.com/developerworks/library/specification/ws-secon/

[28] Web Services Security Framework by OASIS - http://www.oasisopen.org/committees/tc_home.php?wg_abbrev=wss

[29] Liberty Alliance model, http://www.projectliberty.org/

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

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

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

[33] *Web Services Policy Framework (WS-Policy),* ftp://www6.software.ibm.com/software/developer/library/ws-policy.pdf

[34] *End-toEnd Provision of Policy Information for Network QoS*, Volker Sander, Ian Foster, William Adamson, Alain Roy, Proceeding of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC), August 2001;

[35] *Web Service Policy Attachment (WS-Policy Attachmnet),* ftp://www6.software.ibm.com/software/developer/library/ws-polat.pdf

[36] The Security Architecture for Open Grid Services http://www.cs.virginia.edu/~humphrey/ogsa-sec-wg/OGSA-SecArch-v1-07192002.pdf

Annex A. Examples

A.1. Usage Examples for EMS

The EMS component acts as the heart of the WP4.3 modules. For this reason, and in order to show its functionality we present two message sequence diagrams that indicate the "dialogues" and the participation of the various WP4.3 components inside the Akogrimo architecture.

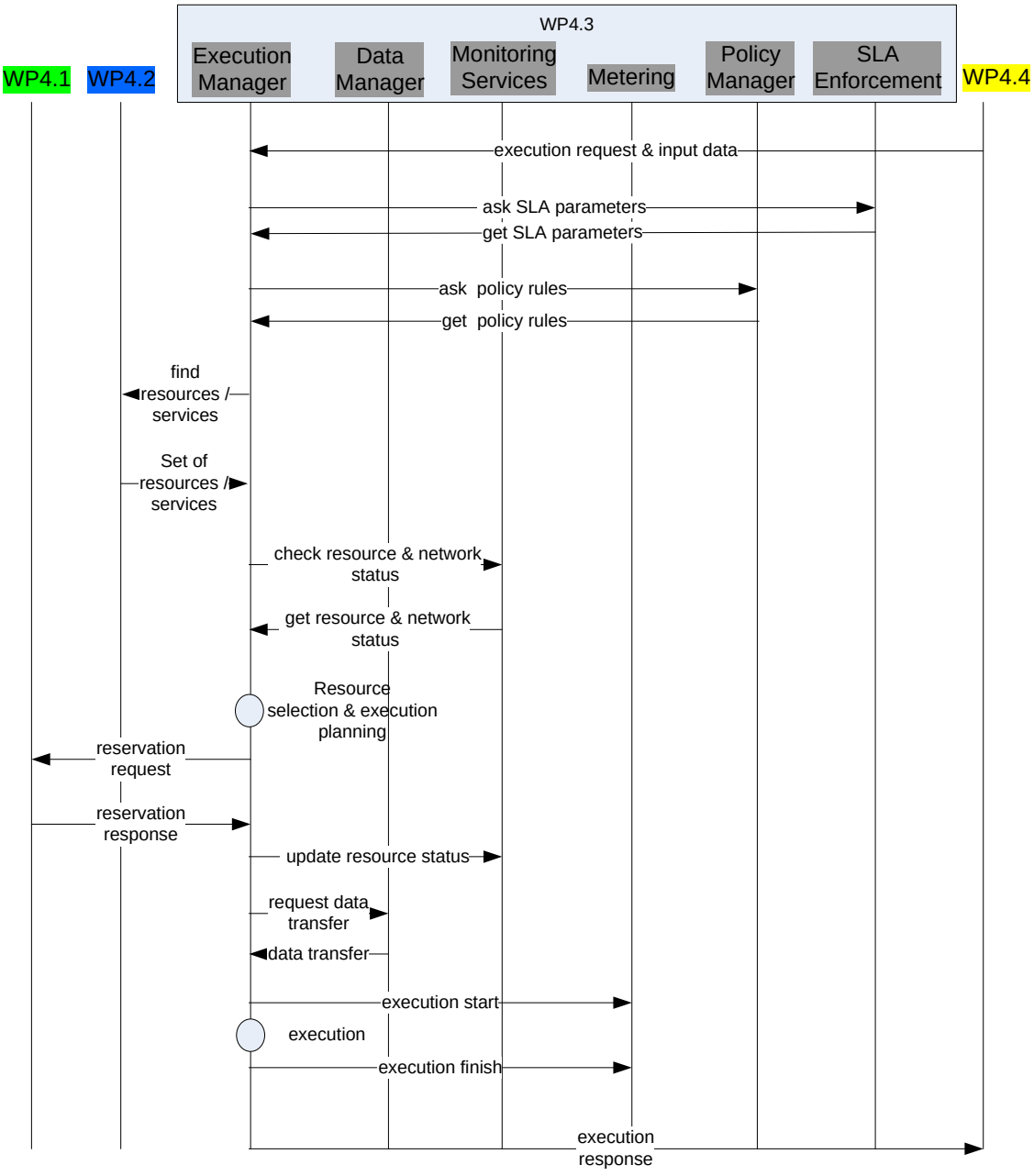

Figure 27: Message sequence diagram of the WP4.3 components and the interaction with other layers

The following message sequence chart shows the internal interaction of the EMS for tha same case as above. It is a typical execution example for a job comprising either a "small workflow" or a single executable process.

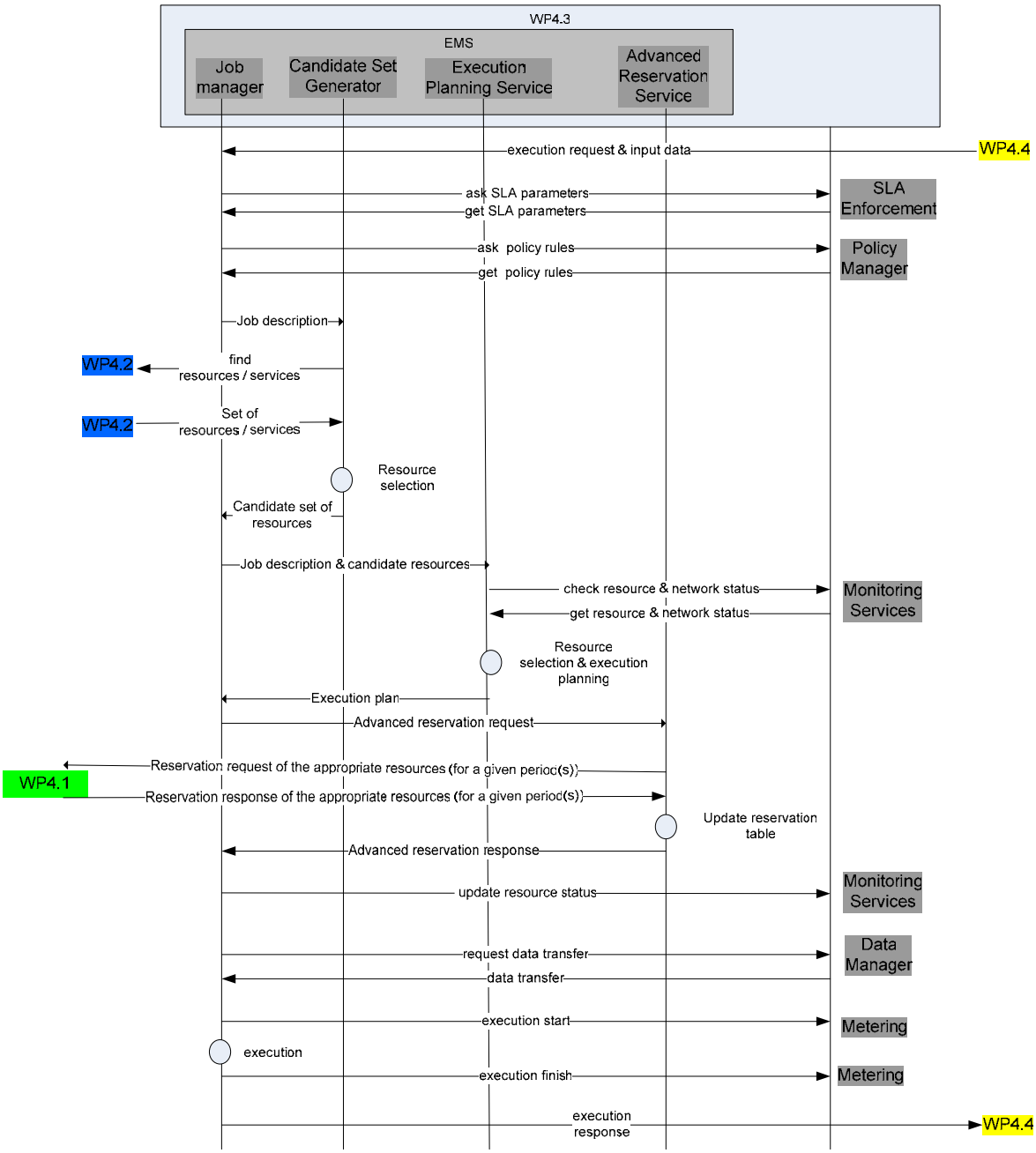

Figure 28: Message sequence diagram within the EMS and the interaction with the other components

A.2. Usage Examples for Data Management

The following use cases are described:

1. Upload a file to a SE and register the file in the RLS
2. Find a replica
3. Replicate and register file

Upload a file to a SE and register the file in the RLS

Figure 27 show the sequence diagram for this use case.

A requestor (that can be a user or a service) ask to upload a file into a specified SE (the *surl* must be specified). If the file is successfully uploaded, the file is then registered to the RLS (the *pfn* is produced by the SE and returned to the RLS). The *lfn* produced by the RLS is the identifier that will be used to find a file in the DRS. The *uid* can be used as well.

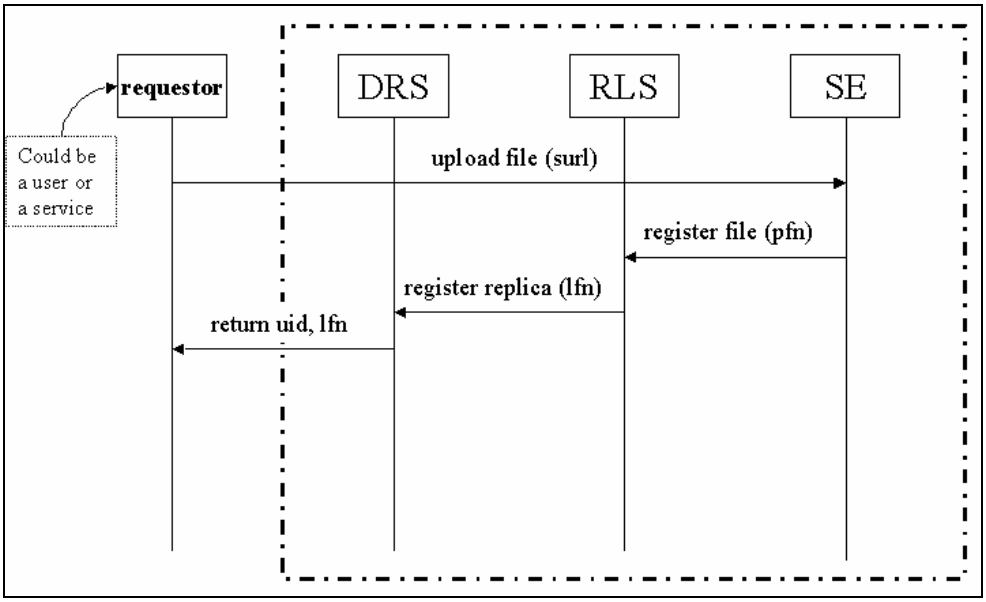

Figure 29: "Uload data to a SE and register in RLS" sequence diagram

Find a replica

This use case show the steps that are necessary to find out a replica (Figure 28).

A requestor asks the DRS to find out a file. The search can be done providing the *lfn* (or *uid*). The DRS asks the RLS to search if and where the file is. If the file is already registered the RLS returns one or more *pfn* (depending on the replicas found).

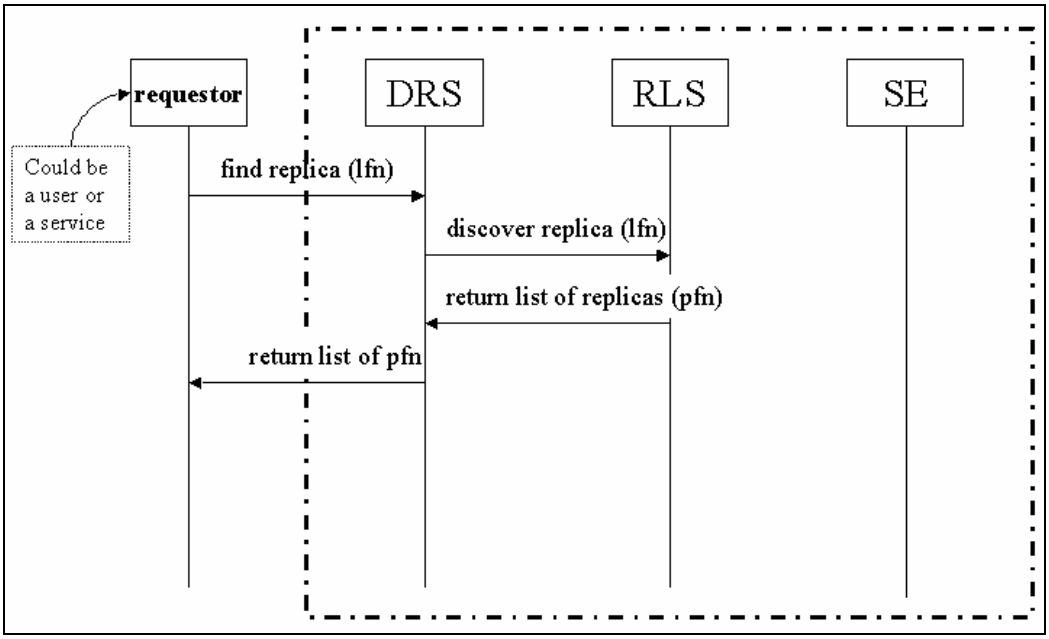

Figure 30: "Find a Replica" sequence diagram.

Replicate and register file

This use case show how to replicate (and register) a file that is already present in the DSR.

The requestor asks to the DRS service to replicate a file. In order to do that it must be given the original *surl* and the destination *surl*. The DRS upload the file to the destination SE. If the file is correctly uploaded then a *pfn* is returned. The DRS asks the RLS to register the new replica. To the existing *lfn* is associated another *pfn* (corresponding to the new replica in a different SE) See Figure 29.

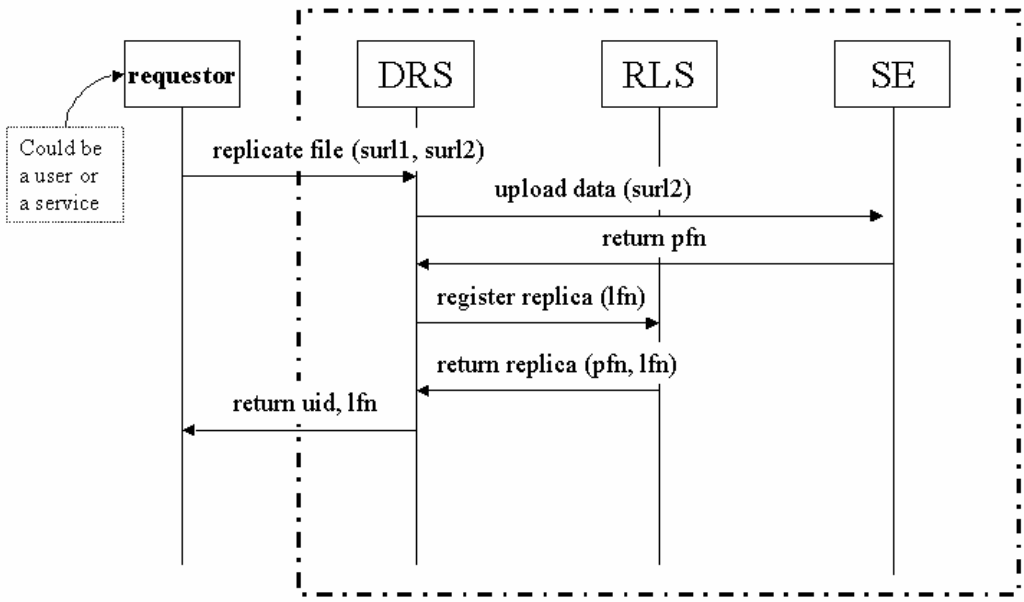

Figure 31: "Replicate & register a file" sequence diagram.

A.3. Usage Examples for Monitoring

A.3.1. Metering Component Registration use case

The Metering component registration is a process established between this component and the registry directly. The registry subcomponent should provide with mechanism in order to carry out this registration. The sequence diagram is next depicted:

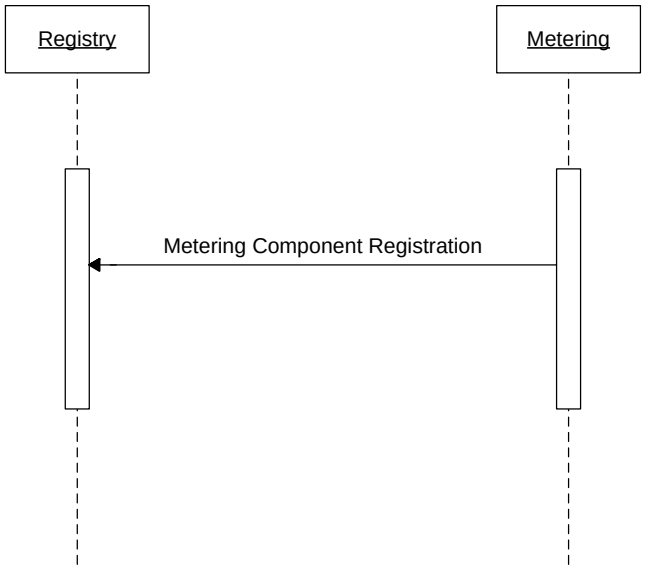

Figure 32: Metering Component Registration use case

A.3.2. Discovery use case performed by the Accounting System querying Metering information

The discovery process is a more complex process established between the consumer (in this case the Accounting System), the discovery and the registry components. First, the consumer should find out all the producer registries that it might be subscribed to. This previous step is carried out by asking the discovery component. Once the discovery component supplies the list of producer registries stores within the registry, the consumer selects those (in this case, the Metering components needed) to whom it desires to subscribe. The sequence diagram is also depicted next:

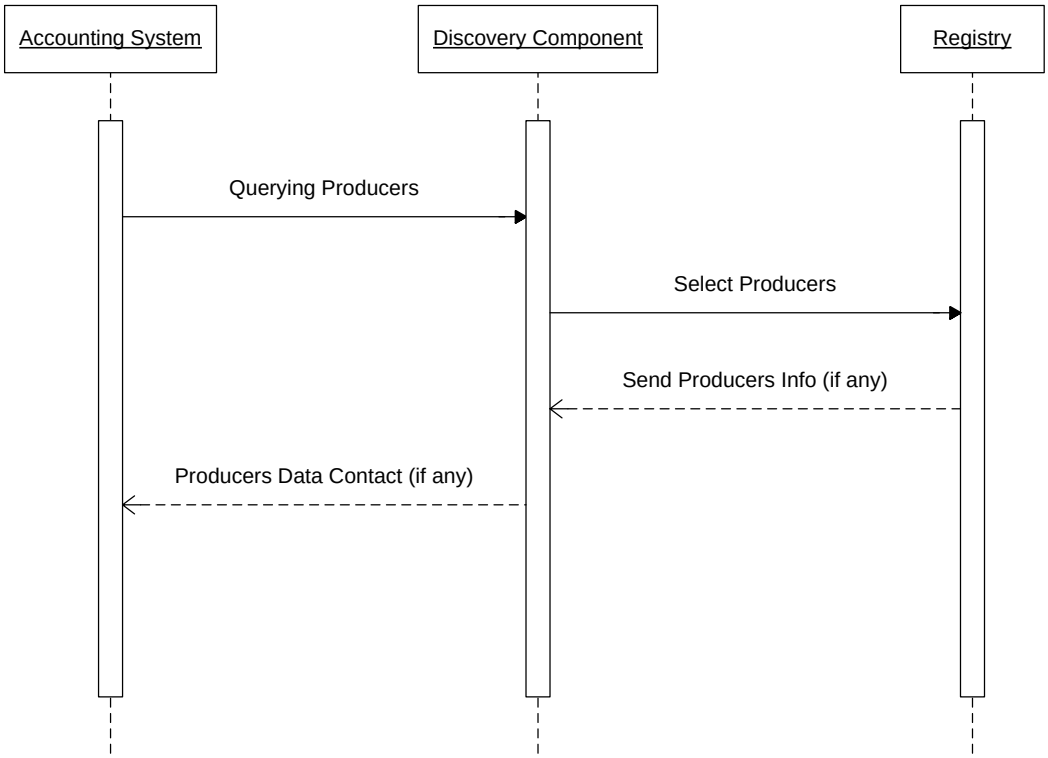

Figure 33: Discovery use case

A.3.3. Notification use case

Once, the consumer component knows the producer contact data, a communicating process between consumer and producer can be established. In this case, we analyse the notification process between the Accounting System (consumer) and the Metering component (producer). This process can be seen as a three-step process. First of all, the Accounting System component shows interest in receiving notification from certain Metering components (it should be taken into account that a discovery process should be carried out before this process in order to know the producers data contact). This process is called subscription. The subscription message includes consumer data contact as well as the notification condition to be fulfilled in order to initiate the second step message: the notification. Once the notification condition fulfils the producer sends a notification message to the consumer. This process will carry out as long as an unsubscription message is not sent from the consumer to the producer. The sequence diagram is simply:

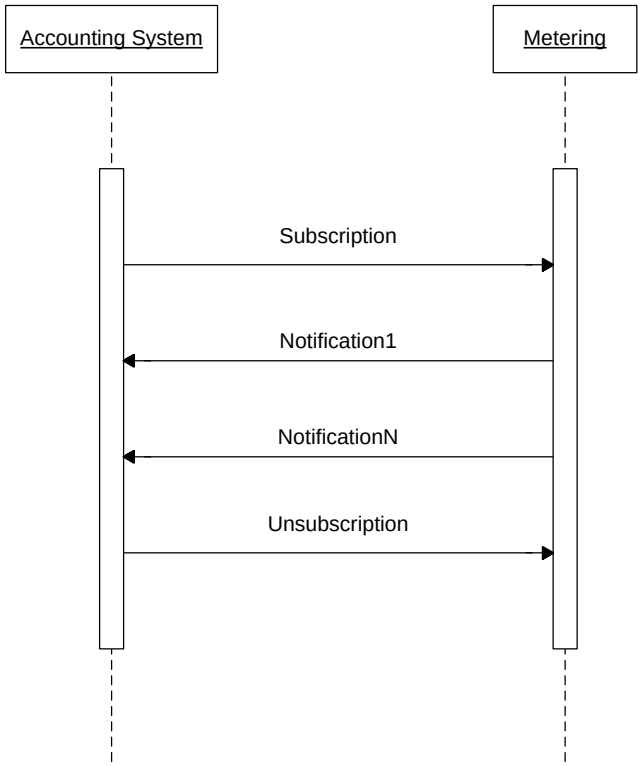

Figure 34: Notification use case

A.3.4. Request/response use case

A different type of communication might be established between a consumer and producer, in this case, between the Accounting System and the Metering component. The request/response is an alternative to the notification. This process is a two-step process. The consumer requests certain information to the producer and then the producer sends a response to a consumer. This is similar to the http request/response. The sequence diagram is next shown:

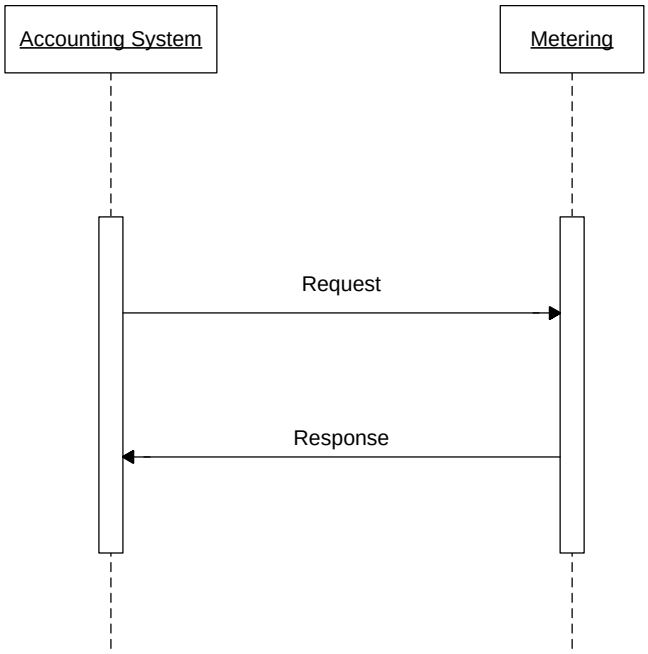

Figure 35: Request / response use case

A.4. Usage Examples for SLA Enforcement

Next figure shows the use case packages that compose all SLA management. Only "Package Instantiation" and "Package ServiceUsage" belong to this layer and will be explained below.

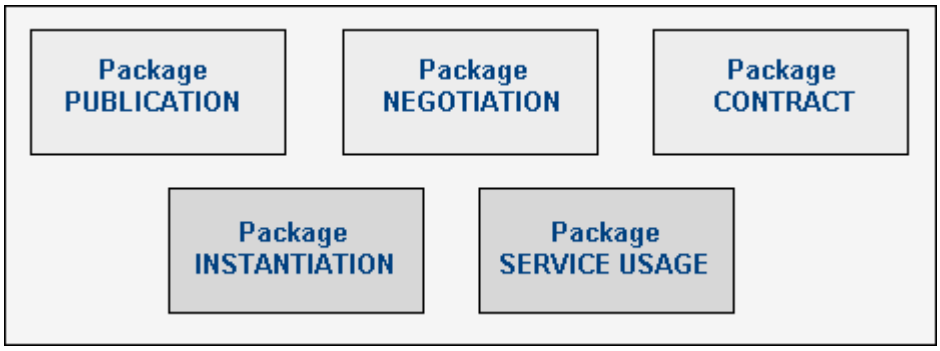

Figure 36: SLA packages

Next are shown both packages that are related to WP4.3 layer. From now on, to make easier the understanding each time the word "service" appears in the text, it will refer to a concrete instance of any kind of service.

The Package Instantiation is composed by a generic use case that is split in two more use cases:

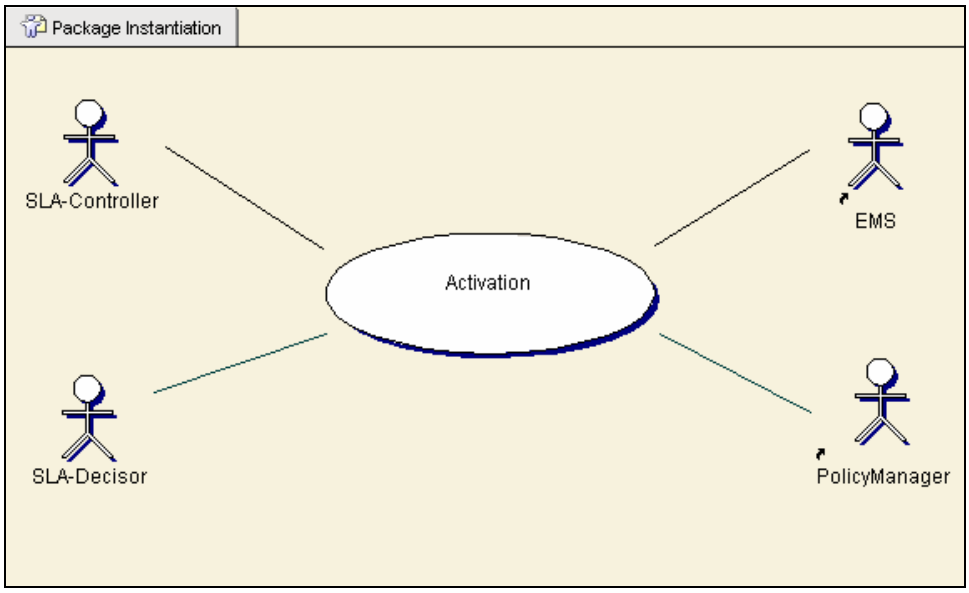

Figure 37: Use case "Activation"

Next figure depicts the breakdown of this generic use case:

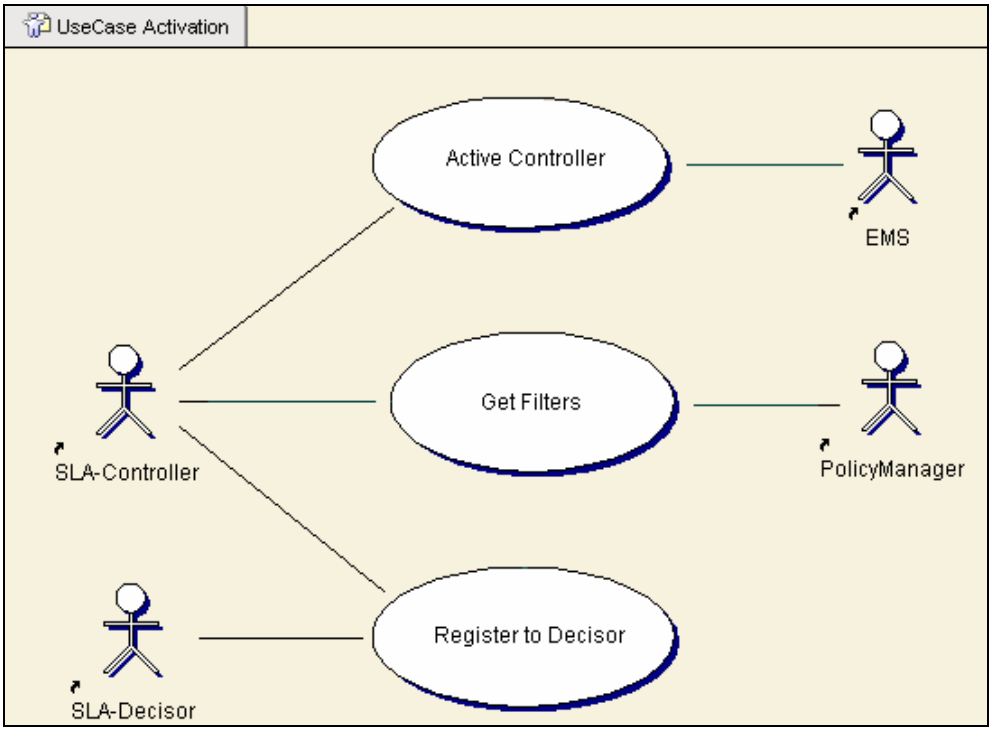

Figure 38: Use case "Active Controller"

Use Case	**Active Controller**
Description	EMS creates a service instance and also enables a SLA-Controller that will be the responsible of the fulfilment of the SLA-Contract for this service

	Flow of events: • EMS creates a new instance of SLA-Controller • EMS associates the Monitoring that is measuring the QoS of this service with the SLA-Controller created
Actor	EMS, SLA-Controller
Preconditions	EMS has created a new service instance
Postconditions	EMS has created an instance of SLA-Controller that is associated to the service execution

Table 8: Use case "Active Controller"

Use Case	**Get Filters**
Description	SLA-Controller asks Policy Manager for internal filters to be applied over the measurements that Monitoring will send every time. SLA-Controller will evaluate the measurement with the filter to decide if the result is considered a violation during the execution. Flow of events: • SLA-Controller looks for filters over the QoS measurements in the Policy Manager
Actor	SLA-Controller, Policy Manager
Preconditions	EMS has created an instance of SLA-Controller that is associated to the service execution
Postconditions	SLA-Controller has obtained the measurement filters

Table 9: Use case "Get Filters"

Use Case	**Register to Decisor**
Description	SLA-Controller is subscribed to SLA-Decisor which will receive its notifications Flow of events: • SLA-Controller communicates to SLA-Decisor its subscription • If SLA-Decisor has already more than X subscriptions of SLA-Controller components is necessary to create a new instance of SLA-Decisor to receive next subscriptions
Actor	SLA-Controller, SLA-Decisor

Preconditions	EMS has created an instance of SLA-Controller that is associated to the service execution
Postconditions	SLA-Controller has been subscribed to the SLA-Decisor

Table 10: Use Case "Register To Decisor"

The Package ServiceUsage is composed by a generic use case that is split in 5 more use cases. There are two main actors (SLA-Controller and SLA-Decisor) that interact with external subsystems.

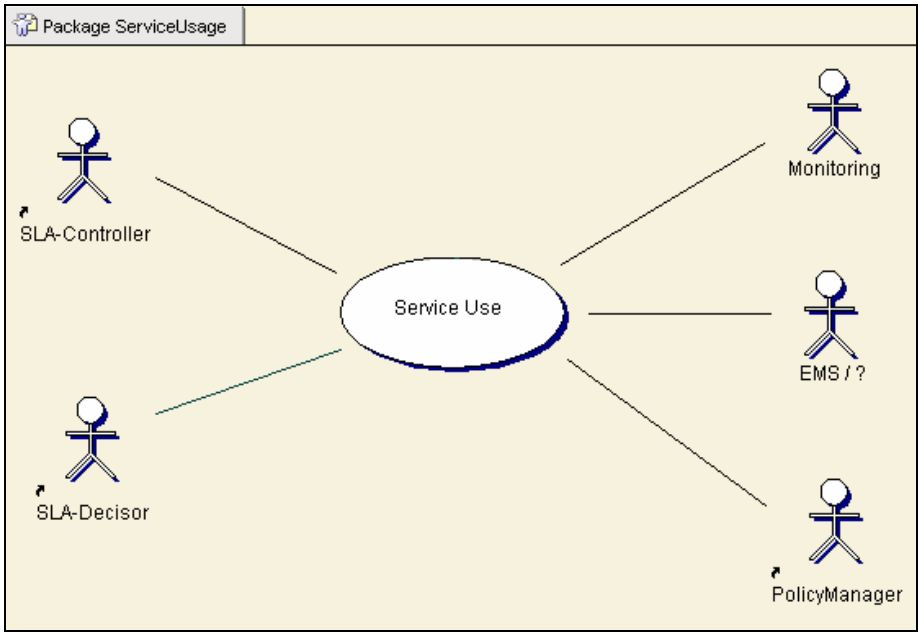

Figure 39: Use case "Service Use"

Next figure depicts the breakdown of this generic use case:

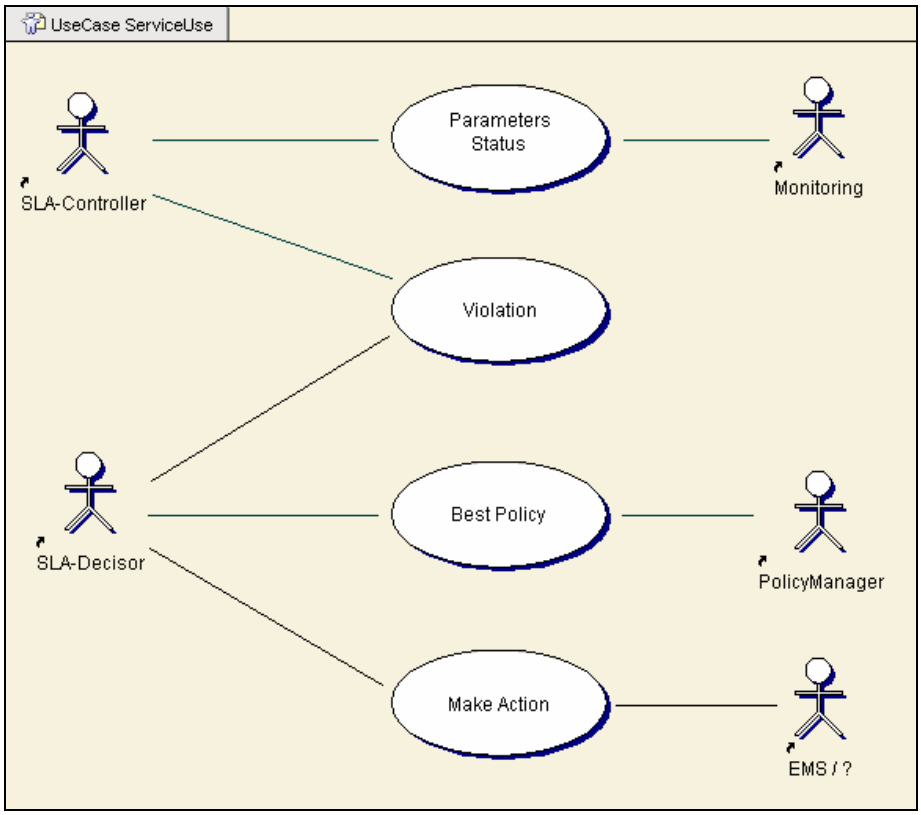

Figure 40: Use case "Service Use" composition

Use Case	**Parameters Status**
Description	Monitoring subsystem sends periodically the QoS measurements of the service to the SLA-Controller Flow of events: • After a fix period of time, Monitoring gets the QoS values of the service • Monitoring sends this information to the SLA-Controller • Also SLA-Controller can interact directly with the Monitoring subsystem to get the QoS measurements of the service Flow of events: • SLA-Controller asks for the current status of the service • Monitoring reads the current QoS • Monitoring sends these values to the SLA-Controller
Actor	Monitoring, SLA-Controller
Preconditions	EMS has notified to Monitoring that a new service is being used and from this moment it has to measure a specific QoS

Postconditions	SLA-Controller receives the status of the QoS related to a service. This action can be invoked directly against the Monitoring or it will be repeated automatically after reaching a determinate period of time until the service is destroyed.

Table 11: Use Case "Parameters Status"

Use Case	**Violation**
Description	SLA-Controller evaluates the received QoS measurements and applies the known filters. If the result is a violation of the contract, it is notified to the SLA-Decisor. Flow of events: • SLA-Controller checks the received QoS values with the filters. • It decides to inform to SLA-Decisor when QoS is not met
Actor	SLA-Controller, SLA-Decisor
Preconditions	SLA-Controller knows which filters are related to this service and now it has received the current QoS
Postconditions	SLA-Decisor will be notified by SLA-Controller of the violations produced by the service

Table 12: Use Case "Violation"

Use Case	**Best Policy**
Description	If a violation has arisen SLA-Decisor looks into the Policy Manager the policy to apply Flow of events: • SLA-Decisor checks the policies established in the Policy Manager • Policy Manager searches for the best policy to apply
Actor	SLA-Decisor, Policy Manager
Preconditions	SLA-Decisor has received a violation produced by a service.
Postconditions	Policy Manager returns to SLA-Decisor the recovery action to do

Table 13: Use Case "Best Policy"

Use Case	Make Action
Description	SLA-Decisor will communicate to EMS or other components which actions should be applied (it is to abort the process, to move the service to another machine, to increase the process priority…) Flow of events: • SLA-Decisor checks the policy received • SLA-Decisor notifies the action to the right component
Actor	SLA-Decisor, EMS/?
Preconditions	SLA-Decisor has obtained the policy that has to be applied
Postconditions	SLA-Decisor has interpreted the policy and communicated the action to be performed to the corresponding component

Table 14: Use Case "Make Action"

A.5. Usage Examples for Metering

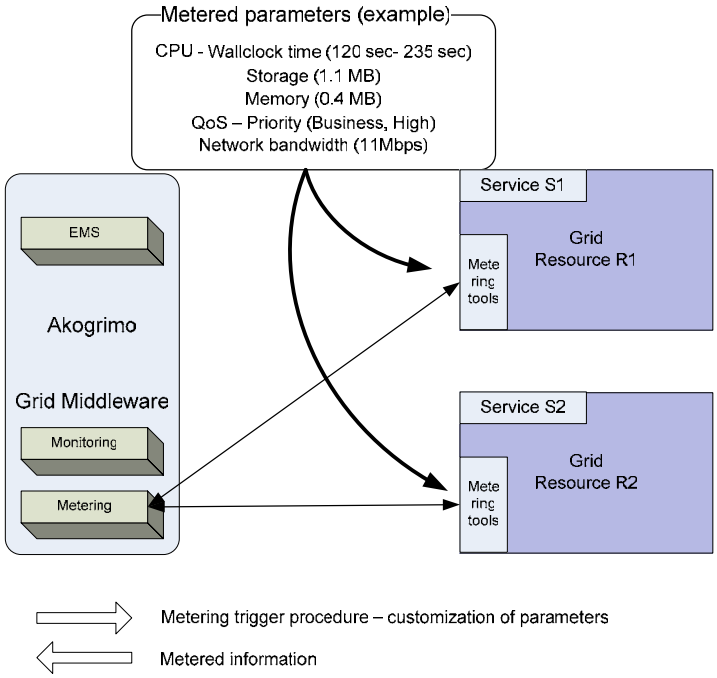

Figure 41: Metering Grid related information

In Figure 39 we see an example of such a metering procedure as it is designed for the Akogrimo project. We can see the flow of the information between the Grid middleware of the basic Grid infrastructure and the metering "agents"(tools) of the resources that are attached to the domain of the middleware. Although the metering procedure is implemented in a more centralised way in the middleware area, there are still some software components necessary on the site of the services so that we can have the site autonomy and the auditing of the metered information.

At the time the EMS starts an execution procedure, the Metering component of the Akogrimo middleware will start the measurements on various parameters that are defined and configured for been measured on the specific execution. This information mainly passes through the middleware itself and is not necessary to be submitted by the metering tools residing on the resource site. Examples are: QoS and prioritization request, wallclock time, data set transfer, network bandwidth, advanced reservation, etc. But there is also a need for some specific parameters measurements from the metering (like I/O bandwidth-messages, memory used, libraries– specific software etc.). The communication between the metering tools on the Grid resources and the Metering component of WP4.3 is through SOAP protocol.

In the following we present an example of a resource usage as it is presented in the Usage Record draft specification of the GGF.

```
<?xml version="1.0" encoding="UTF-8"?>
<UsageRecord xmlns="http://www.gridforum.org/2003/ur-wg"
      xmlns:urwg="http://www.gridforum.org/2003/ur-wg"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.gridforum.org/2003/ur-wg
      file:/Users/bekah/Documents/GGF/URWG/urwg-schema.09.02.xsd">
      <RecordIdentity                urwg:createTime="2003-08-15T14:25:56Z"
      urwg:recordId="urn:nasa:arc:usage:82125.lomax.nas.nasa.gov:0"/>
      <urwg:JobIdentity>
      <urwg:LocalJobId>82125.lomax.nas.nasa.gov</urwg:LocalJobId>
      </urwg:JobIdentity>
      <urwg:UserIdentity>
      <urwg:LocalUserId>foobar</urwg:LocalUserId>
      </urwg:UserIdentity>
      <Status urwg:description="pbs exit status">0</Status>
      <urwg:Memory          urwg:metric="max"          urwg:storageUnit="KB"
      urwg:type="virtual">1060991</urwg:Memory>
      <urwg:Processors urwg:metric="total">32</urwg:Processors>
      <urwg:EndTime>2003-06-16T08:24:32Z</urwg:EndTime>
      <urwg:ProjectName            urwg:description="local            charge
      group">g13563</urwg:ProjectName>
      <urwg:Host urwg:primary="true">lomax.nas.nasa.gov</urwg:Host>
      <urwg:Queue>lomax</urwg:Queue>
      <urwg:WallDuration>PT45M48S</urwg:WallDuration>
      <urwg:CpuDuration>PT15S</urwg:CpuDuration>
      <urwg:Resource                 urwg:description="pbs-jobname">m0.20a-
      7.0b0.0v</urwg:Resource>
</UsageRecord>
```

A.6. Usage Examples for Policy Manager

In this section we show some examples of policy attachment based on some use cases provided by the Akogrimo partners and examples on how a Policy Requestor have to query the Policy Manager to retrieve a policy.

A.6.1. VO Authorization Policy

Description

Inside a VO (both Base VO and OpVOs), there are USERS and SERVICES and the sharing of the services should be regulated. In some way, we would like to define some authorization policies describing access rights of the users and access control list for services.

In the Base VO, these policies should be quite static. In the OpVO, they should be transient (they are created/set up (and destroyed) for each user or service, for each new OpVO) and dynamic (it should be possible to change these policies at run-time: for example, changing right access of some users during the workflow).

Example

A typical usage of these policies inside the VO:

When a new BP (i.e. e-learning application testbed) is instantiated then a new OpVO is created. Some users (with different "roles": student, teacher, expert, etc) and services (i.e.: Field Trip service, Speech-To-Test service, Simulation service, Advanced Meeting service) should be added inside this OpVO. The policies could describe the fact that a student may use only some services, an expert other services and the teacher all services.

Solution

In the following code snippet we have a policy applied to a user, a student, which can use just two services, Simulation service and Speech-To-Text service. In this case an **attribute** named **"role"** is defined to the express all the possible values that represent the individuals that populate the category "user": *student, teacher, expert*. The <context> element is generally used to group all the subjects to which apply the policy. It's equivalent to the <wsp:AppliesTo> element of the WS-PolicyAttachment specification but the set of subjects depends on the combination of attributes we use in the context. In this case only one attribute is required.

```
<akp:policy xmlns:akp="http://akogrimo/policy">
        <akp:context>
                <attribute name="role">student</attribute>
        </akp:context>
        <akp:statements>
          <akp:domain name="ControlAccess" creator="{ManagedURI}"
xmlns:acc="http://policy/ControlAccess">
                <acc:granted>
                  <acc:service name="Simulation">{simulation EPR}</acc:service>
                  <acc:service name="Speech-To-Text">{Speech-To-Text EPR}</acc:service>
                 ...other granted service or reference to Student granted list ...
            </acc:granted>
          </akp:domain>
        </akp:statements>

</akp:policy>
```

The second example specifies the policy for an expert user. In this case the policy could be expressed extending the access granted to the less privileged student, there is some glitch in using hierarchical control access and the kind of model to use is an application level and domain concern. Anyway the needs to reference other policies in a standard way across different domains could be a question to address.

```
<akp:policy xmlns:akp="http://akogrimo/policy">
        <akp:context>
                <attribute name="role">expert</attribute>
        </akp:context>
        <akp:statements>
```

```
                    <akp:domain name="ControlAccess" creator="{ManagedURI}"
xmlns:acc="http://policy/ControlAccess">
                      <acc:granted>
                        <acc:service name="advanced service">{advanced service EPR}</acc:service>
          ...other granted service or reference to Student granted list ...
                      </acc:granted>
                    </akp:domain>
          </akp:statements>
</akp:policy>
```

A.6.2. Filters in Violations (internal to SLA-Controller)

Description

During the execution of a service the monitoring subsystem measures some low level parameters in order to control the service status, and pass them to the SLA-Controller for an after processing. Each parameter has associated a threshold that it should not reach during the service execution. The SP or the Designer has to create some rules/filters to determinate when some threshold excess is considered like a violation. Doesn't have sense that every time a threshold is exceeded a violation has been produced. After receiving each measurement the SLA-Controller will check these rules and will determinate if any violation has arisen.

Example

The service A is running and is established that it has reserved 30% of CPU available and 2 seconds of response time for processing its operations. The monitoring subsystem is controlling periodically these parameters and passes their value to the SLA-Controller. The SLA-Controller knows the rules of service A and then processes the information.

As an example the policies defined should be next filters:

For metric CPUUse:

Filter A (Strict or Transparent): One CPU threshold exceeded implies a violation

Filter B (smooth): The CPU threshold is exceeded 3 times in 1 minute

Filter C (permissive): The CPU threshold is exceeded 5 times in 1 minute

For metric ReponseTime:

Filter X (Strict or Transparent): Time not respected implies violation

Filter Y (smooth): 1 every 2 time the response time is not respected implies a violation

Solution

In the following example of policy we apply the strict filters. When the CPU_usage is less then 30% or Response_Time less then 2 seconds there is a violation and the SLA-Controller can execute the action to solve the violation for instance invoking a web service method. In this case the language to express the subject is trivial. In the *context* we define an *attribute* **"service"** to specify the *endpoint reference* of the service that has to be monitored by the SLA-Controller.

```
<akp:policy xmlns:akp="http://akogrimo/policy">
          <akp:context>
                    <attribute name="service">{Service_A EPR}</attribute>
          </akp:context>
```

```
        <akp:statements>
                <akp:domain name="SLA-Controller" creator="{ManagedURI}" xmlns:sla="http://policy/SLA-
Controller">
                        <sla:assertion>
                                <sla:event>CPU_usage</sla:event>
                                <sla:condition>lessThen(30)</sla:condition>
                                <sla:action>{operation}</sla:action>
                        </sla:assertion>
                        <sla:assertion>
                                <sla:event>Response_Time</sla:event>
                                <sla:condition>lessThen(2000)</sla:condition>
                                <sla:action>{operation}</sla:action>
                        </sla:assertion>
                </akp:domain>
        </akp:statements>
</akp:policy>
```

Notice that in the **statement** part definition we use an **Event-Condition-Action** model (ECA rules) to express the policy. This general model could be used by all the enforcement agents into defining the assertions. All the possible values for the <event> element in defining the policy assertions depend on the specific state variables that is able to monitor the enforcement agent (the SLA-Controller in this example). The **condition** represents the guard that has to be satisfied when the SLA-Controller collects the data about the event; on the contrary the SLA-Controller executes the associated **action**. Generally speaking the action could also trigger another event. This situation could also produce infinite loop but in this way we are capable to express into the policy assertions more sophisticated filters like the smooth filter. In fact we could describe the smooth filter with two combined assertion:

```
<sla:assertion>
  <sla:event> CPU_Usage </sla:event>
  <sla:condition>lessthen(30) <sla:condition>
  <sla:action> inc(counter) ; fire(Smooth_Event)</sla :action>
</sla:assertion>
<sla:assertion>
  <sla:event> Smooth_Event </sla:event>
  <sla:condition> counter < 3 <sla:condition>
  <sla:action> {violation} </sla:action>
</sla:assertion>
```

A.6.3. QoS policy (mapping functions for QoS parameters)

Description

SP has to define the concrete mapping between:

a) the "user-meaning or high level" metrics (i.e. ResponseTime)
b) the system parameters of its administrative domain that will be monitored

Example

A typical mapping of this "policy" becomes when the SP and SC agreed on an SLA contract.

Metric in SLA: ResponseTime. => Low level parameters: CPU.

If the SP and SC agree on a value of ResponseTime = 2 seconds:

- the function mapping should detail that this corresponds to the CPU use of 40% in the machines type A (because the SP has last generation IBM Blade that have a price X),
- whereas the function mapping should detail that this corresponds to the CPU use of 70% in the machines type B (old Pentium IV computers that have a price Y).

Solution

In this example the context information used to attach the policy can be used to express the mapping of user high level metrics (response time) on the parameters used at administrative level (machineType). For each type of machine a specific policy is associated depending on Response time selected. The attributes grouped in the <context> element have to be evaluated by default with a logical AND operator.

```
<akp:policy xmlns:akp="http://akogrimo/policy">
        <akp:context>
                <attribute name="service">{QoS aware service EPR}</attribute>
                <attribute name="ResponseTime">2</attribute>
                <attribute name="MachineType">Pentium IV</attribute>
        </akp:context>
        <akp:statements>
                <akp:domain name="SLA-Controller" creator="{ManagedURI}" xmlns:sla="http://policy/SLA-
Controller">
                        <sla:assertion>
                                <sla:event>CPU_usage</sla:event>
                                <sla:condition>at_Least(70)</sla:condition>
                                <sla:action>{operation}</sla:action>
                        </sla:assertion>
                </akp:domain>
        </akp:statements>
</akp:policy>
```

The language used to express the context could be a subset, perhaps XML-based, of the LDAP filter syntax specified in the RFC-2254 (http://www.ietf.org/rfc/rfc2254.txt). As examples of the LDAP filter, using the original syntax specification we could describe the following expression:

- **(& (role="student") (AuthentToken=*))**

as the filter to define all the policies for authenticated students. The star operator (*) in this case requires that an entity, to be eligible as subject of the policy, must to have an authentication attribute associated to it,

- **(& (role="student") (!(user="francesco"))**

define the policies for all students unless that "francesco".

```
<filter>::= '(' <filtercomp> ')'
<filtercomp> ::= <and> | <or> | <not> | <item>
<and> ::= '&' <filterlist>
<or> ::= '|' <filterlist>
<not> ::= '!' <filter>
<filterlist> ::= <filter> | <filter> <filterlist>
<item> ::= <simple> | <present> | <substring>
<simple> ::= <attr> <filtertype> <value>
<filtertype> ::=<equal>|<approx>|<greater>|<less>
<equal> ::= '='
<approx> ::= '~='
<greater> ::= '>='
<less> ::= '<='
<present> ::= <attr> '=*'
<substring> ::= <attr> '=' <initial> <any> <final>
<initial> ::= NULL | <value>
<any> ::= '*' <starval>
<starval> ::= NULL | <value> '*' <starval>
<final> ::= NULL | <value>
```

As we work in Web Service environment could be suitable to use a language based on XML because has characteristic of extensibility, flexibility, and internationalization and well supported by software library. So we can think to express the LDAP language with XML dialect. Probably, by using this approach, the editor should have a support to help the policy author to write a correct syntax.

A.6.4. Data Access Authorization policy

Description

The Storage Element (SE) is the component which is in charge of storing data. New data can be uploaded to the SE, could be retrieved from the SE, moved, cancelled and so on. The Replica Location Service (RLS) allows the registration and discovery of replicas. The Data Replication Service (DRS) allows users to identified a set of available data in their grid environment. All this sub-components of the Data Management should cope with data and with policies associated to the data. This means that data access should be regulated

Example

A typical example is the following. A user wants to register some data in a SE. It should be possible to know if the user is allowed to access the particular SE. It is also necessary to know if the user can access just a portion of the SE. What it is needed is that policies regulate the access to the data. This policies should be related to the user identity.

These authorization policies could be treated both at VO level or VHE level. In this section, we address the last option: it is necessary to check locally if the user identity is authorized or not.

Solution

In the following code snippet we have a policy applied to a user, a student, which can use just two services, Storage Element service and Speech-To-Text service. In this case an **attribute** named **"identity"** is defined to the express uniquely the identity of the service invoker. The two services are accessible but with some restrictions.

```
<akp:policy xmlns:akp="http://akogrimo/policy">
        <akp:context>
                <attribute name="identity">XYZ</attribute>
        </akp:context>
```

```
        <akp:statements>
          <akp:domain name="ControlAccess" creator="{ManagedURI}"
xmlns:acc="http://policy/ControlAccess">
                    <acc:granted>
                     <acc:service name="Storage Element">
                              <acc:service_EPR>{storage element EPR}</acc:service_EPR>
                              <acc:service_restrictions>…</acc: service_restrictions >
                     </acc:service>
                     <acc:service name="Speech-To-Text">
                              <acc:service_EPR>{ Speech-To-Text EPR}</acc:service_EPR>
                              <acc:service_restrictions>…</acc: service_restrictions >
                     </acc:service>
                     ...other granted service or reference to Student granted list ...
                </acc:granted>
          </akp:domain>
        </akp:statements>
</akp:policy>
```

A.6.5. Policy Query

A Policy Requestor is the agent that has to query the policy manager to retrieve the policy related to the action invoked by a Service Consumer. In the Policy Manager interface the method to execute a query is:

- **Policy QueryPolicy(Context aContext)**

The context is obtained by the Policy Requestor collecting all the relevant information about the Service Consumer. This information is expressed in the form of attributes and has to be associated to the Service Consumer during its life cycle in the VO. A possible mechanism is to adopt the WS-Addressing properties to attach this information to the Service consumer, other way is to store a reference to a system component of the VO.

In the following figure a Service Consumer (Policy Subject) request an action to an Agent. The Agent collects all the data profiled during the Service Consumer life cycle (in the figure represented by the attributes A, B, C, D) and queries the policy manager to obtain a set of policies applicable to the subject. The Policy Manager matches two policies. The more general policy for all the subjects having defined the attribute A and the more specific policy for the subjects with attributes C and D. The policy obtained is the merge of the statements part of the attached policies P1 and P2.

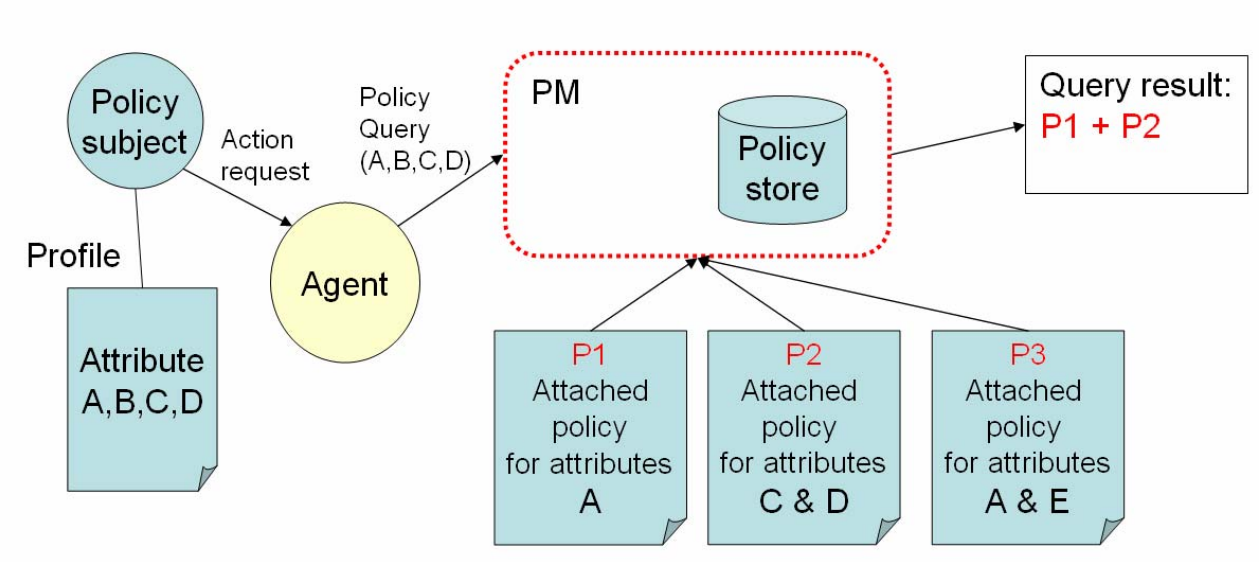

Figure 42: Service Consumer (Policy Subject) requests an action to an Agent. The policy obtained is the merge of the statements part of the attached policies P1 and P2