D4.4.1

Architecture of the Application Support Services Layer V1

WP 4.4 Grid Application Support Services

Dissemination Level: Public

Lead Editor: Giuseppe Laria, CRMPA

07/09/2005

Status: Final

SIXTH FRAMEWORK PROGRAMME

PRIORITY IST-2002-2.3.1.18

Grid for complex problem solving

Proposal/Contract no.: 004293

Context

Activity 4	**Architecture definition**
WP 4.4	**Grid Application Support Services Layer architecture definition**
Task 4.4.2 **Task 4.4.3**	**Layered Architecture (subsystem and interfaces definition)** **Detailed Design**
Dependencies	**D2.2.4, ID2.3.1, D2.3.1, D3.1.1** It will be the basis for D4.4.2 Prototype Implementation of the Application Support Services Layer deliverable

Contributors:

Contributors:	Reviewers
Aniello Rovezzi (CRMPA)	Bastian Koller (USTUTT-HLRS)
Annalisa Terracina (DATAMAT)	Dimitris Skoutas (ICCS-NTUA)
Antonios Litke (ICCS/NTUA)	Antonios Litke (ICCS-NTUA)
Brian Ritchie (CCLRC)	Ferry Sapei (UHOH)
Francesco Verdino (CRMPA)	Ruth del Campo (USTUTT-RUS)
Giuseppe Cirillo (CRMPA)	
Giuseppe Laria (CRMPA)	
Josep Martrat (Atos Origin)	
Julian Gallop (CCLRC)	
Ludovico Giammarino (DATAMAT)	
Pierluigi Ritrovato (CRMPA)	
Raul Bori (Atos Origin)	
Robert Piotter (USTUTT)	
Tom Kirkham (CCLRC)	

Approved by: Stefan Wesner (IP Manager)

Version	Date	Authors	Sections Affected
0.1	22/11/04	Giuseppe Laria	All, template definition
0.2	09/12/05	CRMPA Team	All, template enrichment and tasks distribution
0.3	29/01/05	CRMPA Team	Added section on VO Management, Added paragraph 3.6
0.4	11/02/05	CRMPA Team	Added Legacy Application Integration section Added overview contributions provided by partners on each topic (sections 3.X.1) and some initial contributions on architecture description (sections 3.X.2) Added reference template
0.5	20/05/05	CRMPA Team	Added section on SLA Negotiation Added section on Policy Manager Added section on Orchestration Updated sections 3.1 on VO Management and revision of template
0.6	26/07/05	CRMPA Team	Updated all sections taking into account partners contributions
0.7	26/08/05	All	All, merged review comments and integration requests
1.0	07/09/05	All	Merged minor changes in overall deliverable

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
BP	Business Process
BPEL	Business Process Execution Language
BVO	Base Virtual Organization
BVO-PS	Base Virtual Organization-Policy Store
EHR	Electronic Health Record
EMS	Execution Management Service
GrSDS	Grid Services Discovery System
GT	Group Token
HMES	Heart Monitoring and Emergency Service
HSP	Health Service Provider
HTTP	HyperText Transfer Protocol
ID	IDentifier
LDAP	Lightweight Directory Access Protocol
LPM	Local Policy Manager
MDVO	Mobile and Dynamic Virtual Organization
MGT	Management Group Token
NP	Network Provider
OpVO	Operative Virtual Organization
QoS	Quality of Service
RDBMS	Relational Data Base Management System
RHN	Regional Health Network
SA	Service Agent
SAML	Security Assertion Markup Language

SC	Service Customer
SGT	Session Group Token
SLA	Service Level Agreement
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SP	Service Provider
TBC	To Be Confirmed
TBD	To Be Determined
TLS	Transport Layer Security
UA	User Agent
VEHR	Virtual Emergency Health Record
VHE	Virtual Hosting Environment
VHE-PS	Virtual Hosting Environment-Policy Store
VID	Virtual IDentifier
VO	Virtual Organization
VOMS	Virtual Organization Management System
WP	Work Package
WS	Web Service
WSDL	Web Service Description Language
WSRF	Web Services Resource Framework
XML	eXetend Markup Language

1. Summary

The Grid Application Support Services Layer collects all services necessary to manage applications at application layer. In particular it contains the following major components:

- VO Management: include services and functionalities to manage static and dynamic Virtual Organizations, taking in account the nature of users that can be mobile.
- BP Enactor: manages workflow in order to guarantee the execution of different services involved into a business process.
- SLA Management: manages the negotiation phase with contract establishment and violations of agreements.
- Security: checks and guarantees secure access to business and infrastructure resources.
- Application Specific Services: defines services and tools to support the execution of a business process in Akogrimo environment, in the context of a specific testbed application.

Legacy Application Integration: gives input and guidelines to migrate legacy applications to the Akogrimo infrastructure.

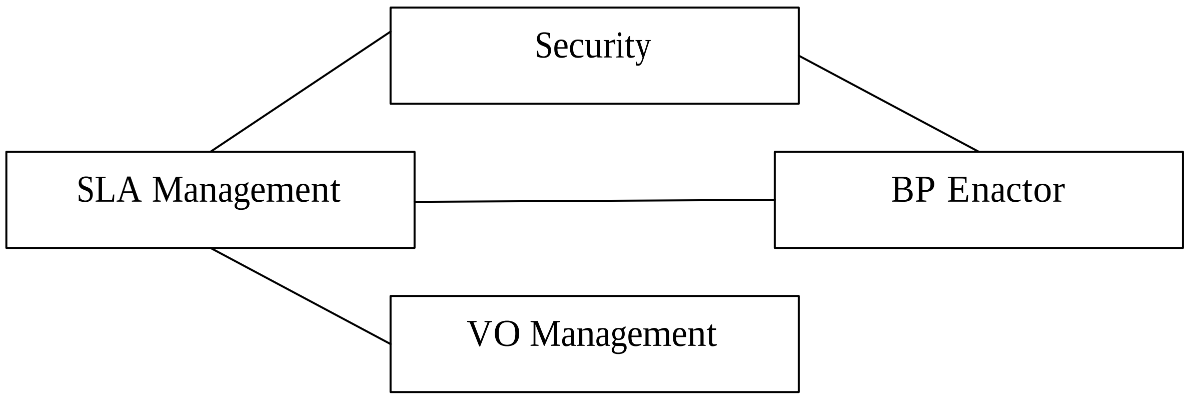


Figure 1 Major components in the Grid Application Support Services Layer

The figure (above) shows the major components, excluding Application Specific Services. It shows that VO Management directly communicates with all other components and this is reflected in the prominence given to it in chapter 2. Each major component and Legacy Application Integration guidance is described in a separate section in chapter 3.

This layer is produced within Akogrimo WP 4.4, a term which is occasionally used as a short hand for labelling this layer. Also, one of the important considerations in Akogrimo, the components of this layer make use of components produced in other layers and similar shorthand terms are used as follows:

- WP4.3 – Grid Infrastructure Services Layer
- WP4.2 – Mobile Network Middleware Layer

The details of these interactions can be found in the detailed component descriptions in chapter 3.

2. Overall Grid Application Support Services Layer description

Before the description of the major components in the Akogrimo Grid Application Support Services Layer, it is useful introduce some consideration on VO Management. This chapter presents an overall description of this layer by considering VOs as the fundamental concept. It describes the types of VO in Akogrimo and the life cycles of membership and services. It goes on to consider SLA and Workflow in this context.

The WP 4.4 (Grid Application Support Services Layer) provides services for enacting and enforcing the creation of VOs as well as for supporting workflow management or compound services. By means of VOs several organizations/providers are able to share resources and services within a managed, secure and trusted environment, but some problems arise when the involved entities are not bound to a fixed point of connection in its VO. Therefore, in Akogrimo context, the VO meaning will be extended to more flexible capabilities inferring the added-value view of a MDVO. To achieve this aim, Akogrimo leverages its dynamic behaviour by allowing to replace resources, to accommodate changes or to adapt to new opportunities in the business environment. Moreover, users and resources are not bound to a fixed location and therefore Akogrimo has to take into account mobility aspects like contextuality and personalization.

In the following, we will introduce some general concepts for this layer, providing an overview of the VO vision supported by Akogrimo.

2.1. General concepts

2.1.1. Base and Operative Virtual Organizations

We will refer to term “VO” in Akogrimo instead of “MDVO”, because MDVO will be the norm.

In Akogrimo we talk about mobile/nomadic resources highlighting the possibility to move them in the Akogrimo environment and continuing to provide/use them without changes and constraints. All entities/organizations that are able to provide resources are called SPs, otherwise we can have users and customers. In this last case a user will access/use a resource bought by a customer. Obviously all SPs, users and SCs have to be grouped and recognized inside a VO. The following discussion of SP entities is sufficiently general that it applies also applies to SC entities.

Moreover, in Akogrimo we can distinguish two kinds of VOs: Base VO (BVO) and Operative VO (OpVO). They have similar functionalities, but applied in different context. Indeed the BVO is a static VO that lives until at least one SP belong to it and manages all action performed in a VO. The OpVO is more dynamic adding to management action of the BVO all operation necessary to drive the execution of a business service. The OpVO has the same lifetime of the business service.

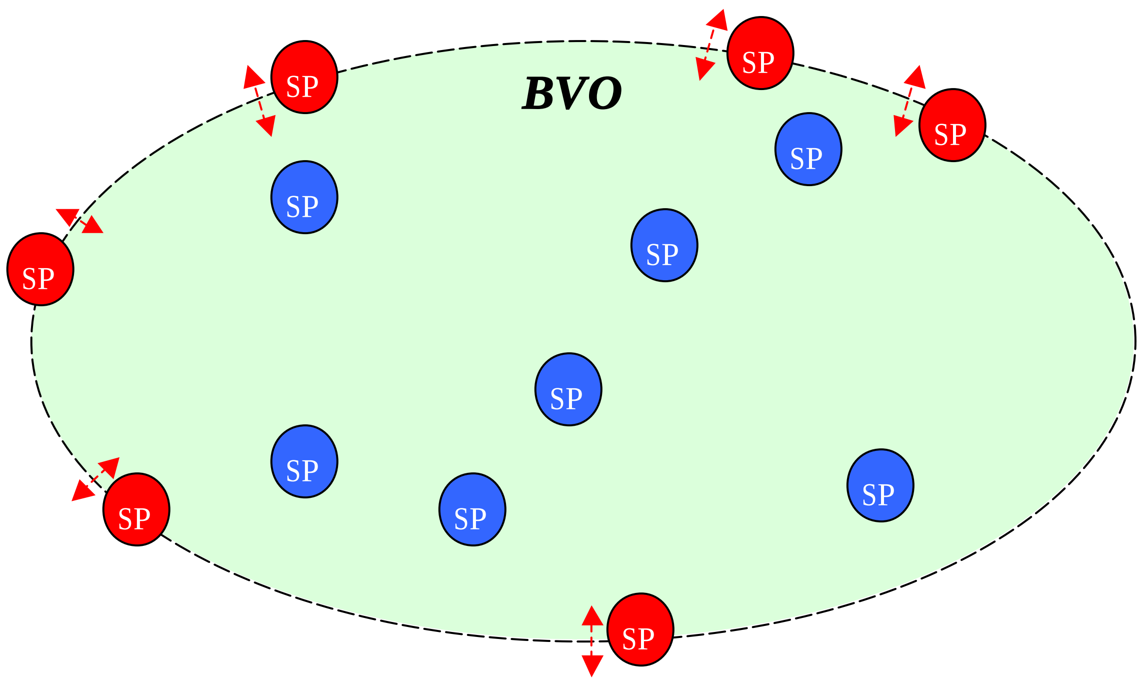


Figure 2 SPs members of BVO

In Figure 2 a general view of BVO is shown. The dotted oval contains all SP members of the BVO that are available and so, visible to the BVO. The BVO is open, namely new members can be included inside the BVO and existing participants can leave the BVO itself as well. This BVO includes some static SPs (blue) and some mobile/dynamic SPs (red) and, in order to ensure the service usage in a transparent way, the VO must manage these entities granting the ubiquitous and pervasive service presence for their customers/users. The mobile SPs in red are shown on the boundary and with arrows directed inwards and outwards the scope/visibility of the BVO to emphasise their variable connectivity status; this means that these entities can probably become unavailable, for reasons tied to their mobility, and the BVO manages these events.

Inside the BVO different OpVO, involving participants of the BVO, can be created. The *OpVO* is the transient environment corresponding to instanced services and users brought together for business purposes such as the consequence of workflow/orchestration instantiation where different static and dynamic SPs collaborate among themselves to provide services in a contextualized and managed way. The OpVO is open too, in the sense that participants could be substituted, if required, also after its instantiation, but with more restrictive constraints: the possible ‘substitute SP’ must be already known (in the sense that belongs to the BVO) during the OpVO creation phase. Therefore, from logical point of view an OpVO could be thought like a regular BVO representing entities belonging to different organizations brought together, but its functionalities are specialized because the OpVO is considered to be short-lived.

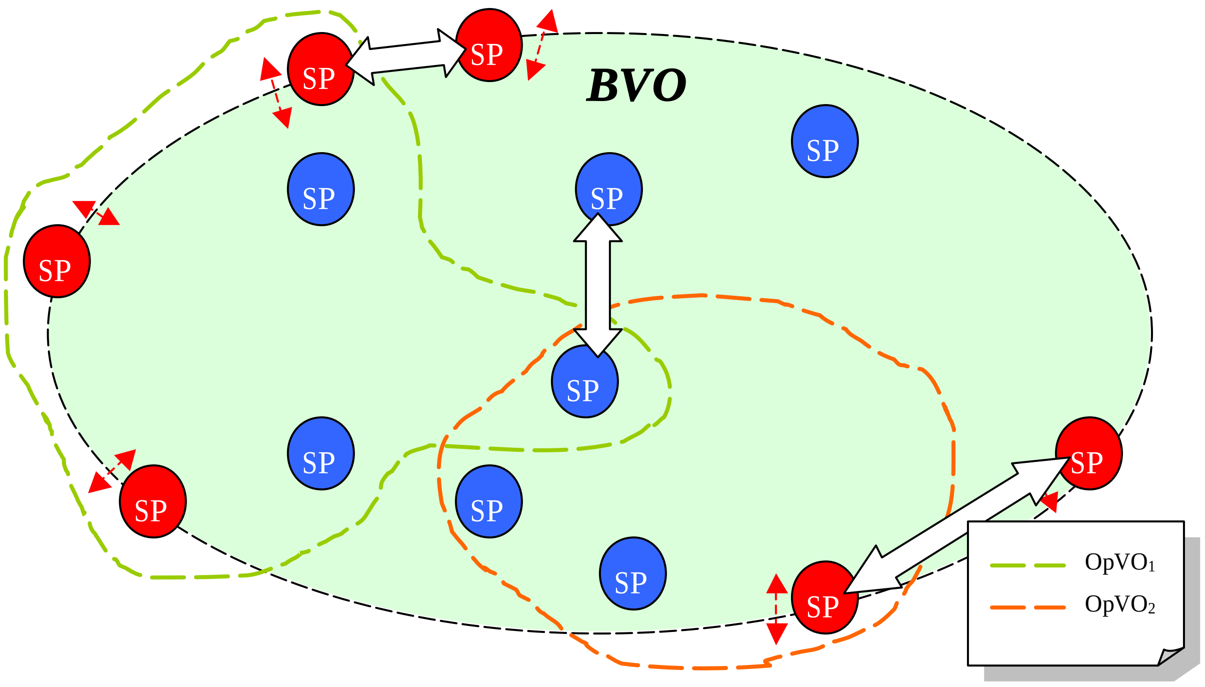


Figure 3 OpVOs from the BVO

In Figure 3 we have the same BVO with two OpVOs (1 and 2). Each OpVO includes several SPs, and some of them (just one in this figure) are shared between the two OpVOs. The shared SPs could provide different or shared resources (e.g. service instances) to each OpVO. During the execution of the OpVO a new member is allowed to join if the member is a user or the member is a new SP enlisted as 'reserve' during the OpVO creation (see bidirectional arrows in Figure 3). Reserve SP means only a normal SP that, during the research of the available SPs providing the services necessary to the OpVO creation, is not chosen for the actual provision of services, inside the OpVO in construction, but is considered for possible utilisation in case of service failures of the chosen SPs. We could think these reserve SPs as SPs that provide very good services for the OpVO but not the optimal ones; so, they do not result the best choice for the OpVO execution but may be worth in some special circumstances (e.g. failures). Obviously, since the BVO is a controlled environment, new members can join to OpVO if some binding rules are satisfied and thus are enabled to replace their corresponding.

In order to manage this environment we have to take into account two different aspects:

1. The first one is related to the creation of the BVO and its management in term of new participants' arrival and departure.
2. The second one is related to the creation of OpVOs and it is mainly related to the instantiation of a "workflow". In this context, workflow has a very general meaning, it identifies how the different service providers (and customers) involved in an OpVO have to collaborate inside the OpVO itself.

Adopting this VO vision, Akogrimo will be able to manage the lifecycle of the VO, using the BVO as an enabler VO that offers services useful to create new VOs (OpVOs). Further the BVO is responsible for the formation of business agreements (among BVO participants) in order to drive the creation of the OpVO. This reduces highly the complexity of the OpVO to include only those services specifically involved in the application execution.

We notice that, from now on, when we use the term VO we want to summarize general concepts that can be *applied to both BVO and OpVO.*

2.1.2. Membership Lifecycle

We think it is useful to point out an initial terminology for defining the several states that an entity may assume during its lifecycle, with respect to the VO.

At the beginning, an unknown entity subscribes itself to the VO, becoming a visitor. As a visitor, it is identifiable but is not able to make anything inside the VO; so, the subscription mainly deals with the identification of the entity that wants to participate. To use services in the VO, it is necessary to register the visitor to the VO, in order to make him a member; the registration mainly deals with associating an operative meaning (the role). As a member, the entity has a role and can make operations (invoking or being invoked). (We use the terms register or add or join interchangeably)

We define also the unsubscribe and removal phases respectively as dual of the subscribe and addition phases. Obviously, the removal phase has to be executed before the unsubscribe phase. This ordering is mostly logical: the removal phase deletes the role of a participating entity and the unsubscribe phase deletes the identity (for the VO) of the participating entity; so, it has no sense to remove the identity before the role. Furthermore, it could be possible to remove a participant as member from a VO (removal phase) but not necessarily remove its identification (unsubscribe phase).

As sub states being a member, we define a *logged in* and *logged out* member. A member may actually do something if he is logged in.

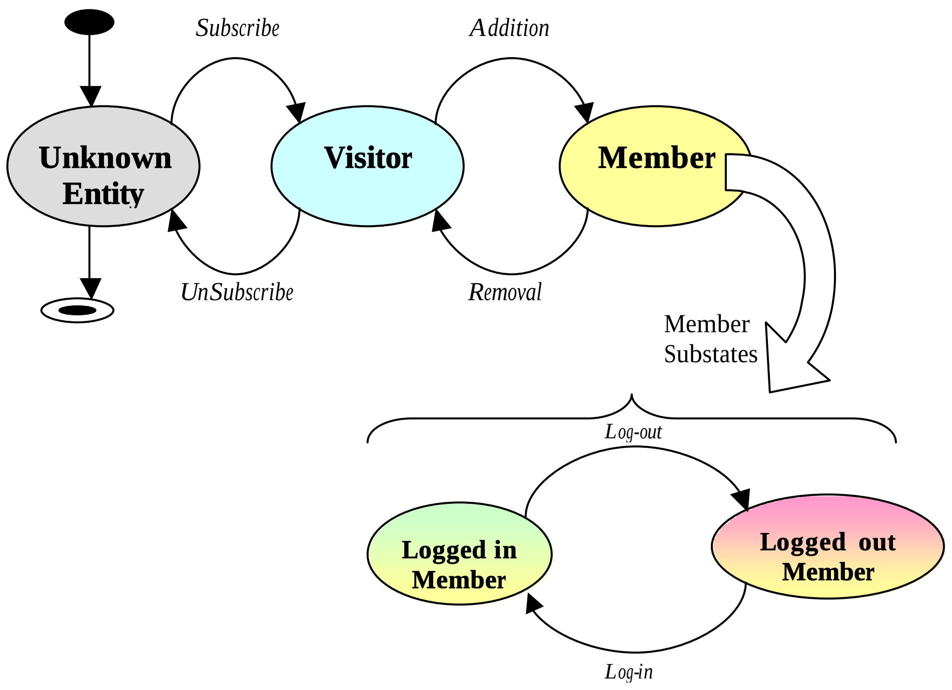


Figure 4 VO Membership Lifecycle

2.1.3. Profile, Policy and Membership management

Life inside VOs needs to be controlled, so each participant will be subordinated to some rules in order to access resources (i.e. services) provided by other participants. Furthermore each participant has to clarify the rules under which its own resources may be used by other participants. Furthermore, these rules need to be taken into account when OpVOs are created.

To achieve this aim, a BVO participant that will provide a service (i.e. a SP) has to produce also a set of policies (requirements) consisting of statements, rules or assertions that specify the correct or expected behaviour of the resource. These policies need to be verified in order to create OpVOs involving several participants. For each BVO participant, there will be created a profile stored inside a participant registry and there will be associated some BVO policies (created by the BVO administrator). Furthermore, they have to expose a list of attributes that will allow the BVO manager to understand which relationships can be undertaken with the participants.

Example: a policy could specify that the SP doesn't permit to rent their services to specific customers (e.g. belonging to particular nations because there are some restrictions in them), then each BVO participant had to expose a "nationality" attribute in order to allow the BVO manager to match policies and participants. If the user doesn't expose the attributes necessary to satisfy the SP policies or BVO policies (e.g. highlighting them in the profile) he will not be able to access resources under policy checking constraints. Obviously in a BVO we can have different type of resources and each one of them can have one or more access policies associated with them. If the resource doesn't have access policies on it the BVO participant will have free access on them. At the moment we don't foresee a dynamic change of a policy at runtime until the service or resources is used.

In addition to this kind of policies the BVO participant should specify if there are some technical restrictions to access their services (e.g. the local environment of the SP could support only particular security and authorization mechanism). In general we can say that each new participant will be characterized by policies related to different scopes (e.g. security, privacy, resource use, etc.) and his context/profile (that will be the ensemble of "attributes" and possible constraints, e.g. it is a mobile user with some specific characteristics).

On the basis of this consideration, we imagine a hierarchical architecture with a BVO. In particular we can imagine that inside the BVO there are different VHE, that can be treated as local environment associated to specific SPs. Then we can foresee a BVO Membership Manager that needs of interacting with some LPM in the local environment of the organizations (VHE) for enabling different entities/resources to communicate among them. This means that whether an OpVO is going to be created, the LPMs of local environments involved into the OpVO will be contacted by asking them to set up the environment in order to permit the communication among the parties engaged within the OpVO (for more details see paragraph 2.2.4).

We refer to the local environment as a Virtual Hosting Environment (VHE). Policies, whether for security or resource control or some other sort, are expected to vary from one VHE to another, but are described in a standardised language. We use the word virtual in this connection, because the VHE does not necessarily correspond to a specific single host computer.

In a general description of policies we refer to three types of scopes:

- **BVO policies**: are policies related to BVO supporting services and having visibility to all members of BVO. For instance, a Workflow Enactment service can restrict its use to only working days.
- **VHE policies**: are policies mainly related to local administration and their details don't need to be known outside the BVO. For instance, a policy related to a resource use

(service S), in order to accomplish its goals, can use one of resources R_1, R_2 or R_3; all belonging to the same VHE. This information has no visibility to the rest of BVO since its relevance has visibility only within the VHE.

- **VHE-to-BVO policies**: are policies related to resources in the VHE but, since their applications involve BVO supporting services or entities belonging to other VHE, their visibility must be extended outside the VHE. For example: two component services, which are involved into the execution of a workflow, belong to different VHEs and each of them has to respect specific policies. Because the execution of the workflow runs involve BVO and OpVO then could be feasible to make a match between all policies at different level.

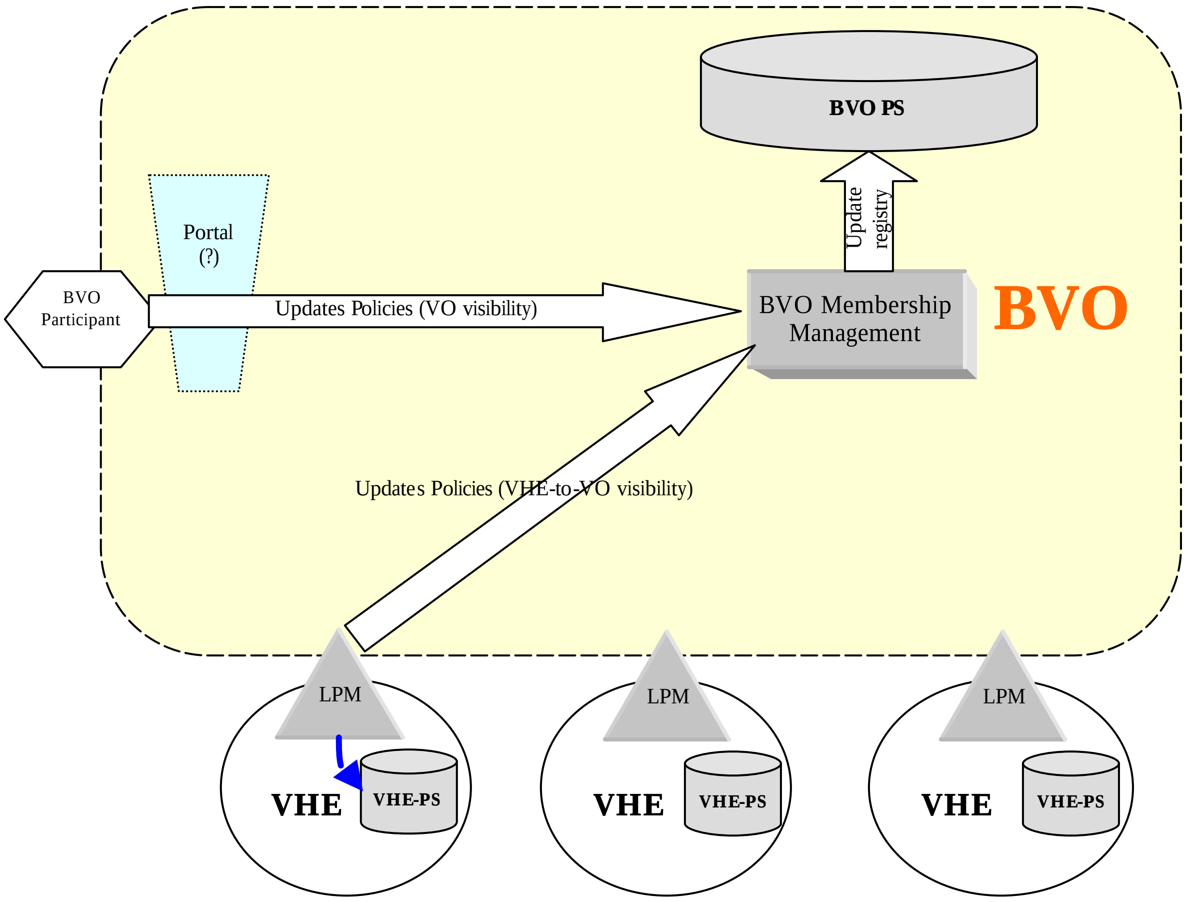

BVO PS
Update registry
Portal (?)
BVO Participant
Updates Policies (VO visibility)
BVO Membership Management
BVO
Updates Policies (VHE-to-VO visibility)
LPM
VHE
VHE-PS
LPM
VHE
VHE-PS
LPM
VHE
VHE-PS

Figure 5 VO policy management

Each of these types of policy is held in a corresponding Policy Store. In Figure 5 a general representation of BVO policy management is depicted. The BVO Policy Store (BVO PS) holds policies related to VO entities (e.g. the SPs, SCs described in 2.2.1 or BVO supporting services themselves), while the Virtual Hosting Environment Policy Store (VHE-PS) holds policies related to VHE resources. In order to define new policies about an entity (i.e. a service) with VHE-to-VO visibilities, a BVO participant having the rights (e.g. the Service Provider) has to apply for support to the Local Policy Manager of its VHE. In the case of services with BVO visibilities the interactions can be direct from the BVO participant (e.g. BVO administrator) and the BVO Membership Management because the policies for services are being defined have only VO relevance.

2.2. Main operations identified

In this section, we describe the main operations affecting BVO participation – subscription, registration, log in and log out – and BVO operations affecting services – publish, buy/acquire, use and destroy.

We remember that, in Akogrimo, there are two specializations of VO concept in terms of BVO and OpVO to achieve specific needs. So, in the next paragraphs, we will use the term VO to refer both BVO and OpVO (as stated in 2.1.1).

2.2.1. Subscription and Registration

Within the Akogrimo BVO each participant has to be recognized and its role has to be known in order to manage the regular BVO access and service use. Indeed, in Akogrimo, BVO should provide capabilities to subscribe and register new participants inside the BVO itself. The subscription and registration (and the dual operations unsubscription and removal) are two separate and different operations.

The first phase is concerned to BVO *subscription*, where an unknown entity (person or resource) becomes a BVO visitor for an undefined period of time (long term subscription). As visitor, the entity has not yet a role inside the BVO; so, another operation is necessary to achieve this: roles are assigned to visitors with the operation *addition* (or *join* or *registration*). This last operation adds/registers a given recognized entity to a VO with one/more associated roles that define the set of privileges for accessing VO resources. These roles should be defined by the VO Administrator and released to participants of BVO allowing them to access and use the available resources in a controlled and secure manner. At BVO level, the BVO Manager is the real entity in charge of enrolling users and resources (along with managers assigned to them) into the BVO.

At the moment, we foresee the subscription operation only at BVO level as a *una tantum* operation; so, to become visitor, it is necessary to subscribe to the BVO. After the subscription, it is possible to make a registration. The registration is requested *per-VO*: this implies that if a visitor wants to access/provide services inside a given VO then he has to register to this VO (BVO or OpVO). Typically, we will have one (or few) BVOs and many OpVOs. For example, a visitor that only registers to the BVO can invoke/offer services inside the BVO but can not access/offer services in an associated OpVO, unless explicitly registered in that OpVO.

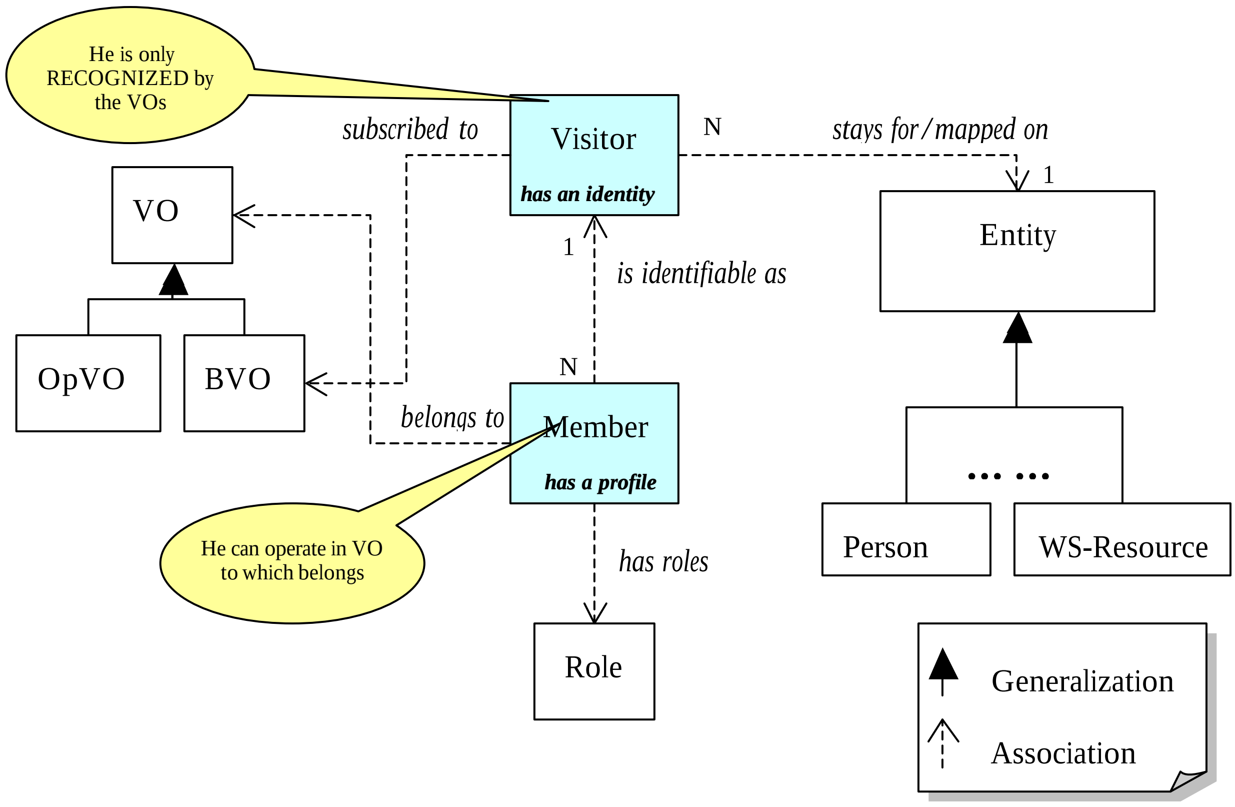


Figure 6 Visitor and Member

The Figure 6 shows graphically some relation between the introduced concepts. The OpVO and BVO are concrete specialization of the abstract VO. A visitor has to be subscribed to a BVO and is a first (identifiable) representation of an entity inside VO environments. A member has to belong to one VO (OpVO or BVO) and is identifiable as a visitor. As consequence of several registrations, a visitor may be associated to many members.

During the subscription/registration phases, a distinction must be made between services and users that will become BVO members. From business point of view this distinction is needed for enacting who provides usable functionalities and who will use them. As generic idea (but not only the unique choice) the subscription/registration phases can be accomplished by different entities:

- The network operator, service providers or other general organizations (e.g. a bank) wishing to become BVO member can personally contact (actually, who contacts is their representatives) a BVO administrator who will subscribe and register them. In this way they will become BVO-trusted entity and will be allowed to publish their services or enable users to access in.
- A user that wishes to become member of BVO could accomplish through a portal that provides a subscription and registration phases with several forms to be filled in. A way to access to this portal is by means of user's network operator that represents a trusted entity for the BVO and guaranteeing for the new BVO participant.
- Services are made available within the BVO by means of their provider that will notify the VO environment of their presence. This task can be effectively executed by service provider administrator that could access to a BVO portal and provide all needed information for service publishing.

2.2.1.1. VO roles

Inside a VO, each member assumes a specific role, mainly based on action performed, so the VO should be able to store and manage them. In particular, when the participant decides to register to the VO, on the base of action taken, he can assume a specific role. For this reason in Akogrimo environment we can distinguish two main categories of roles: BVO roles and OpVO roles.

- **BVO roles** are base roles covered by all members that belong to the BVO. We can distinguish them in:
  - BVO Administrator: is the entity in charge of managing the BVO environment by issuing roles, constraints and privileges. The BVO Administrator has the main role to enable new organizations to join to BVO by making them trusted within BVO environment.
  - SP, sells/provides services. This role is different from the Service (not defined explicitly here) that actually implements the functionality; it is a trusted entity involved within the BVO and is able to publish service for their purchase by other BVO members. The SP embodies the management of a VHE where services are brought together to represent the overall capabilities provided by SP. In this context, the SP is also in charge of assigning/revoking roles to/from the BVO members affiliated with its administrative domain (or VHE), as well as of administrating authorization of BVO member to its domain.
  - SC buys/consumes services. This role should be different from the User (not defined explicitly here) that actually only uses the service, stating nothing about payment; this is the entity recognized within the BVO (and thus trusted to BVO) able to acquire services and make available to its users. For example, a SC could be a network provider that acquires services on behalf of its clients; the SC will charge itself of payment and could require its service requestor to pay in alternative way such as a debt on client fee. In a similar manner to a SP, the SC has to manage its environment and ensure the right access and use to/of its environment.
- **OpVO roles** are business specific roles. For instance if an OpVO is set up to provide e-Learning application, there could be roles for students, teachers, experts…

This difference is needed for distinguishing who will provide resources, what kind of resources is provided, who will buy and who will utilize them.

Besides the introduced roles, we can introduce some basic meta-roles; these meta-roles may be used to better characterize each role. Indeed, each role may have one/more meta-roles. As basic definitions, we introduce:

- **User**: is only able to invoke services. The user will be related to capabilities and constraints (privileges) to use services within the BVO or inside the OpVO. We can introduce also the following specializations:
  - **Management User** : is an user with management purposes; so, it is necessary/responsible for some management operations inside the VO; as first idea of entities that can achieve these meta-role, we can list not only the obvious Administrator member but also the management services foreseen in the main components of the WP4.4 Layer (BP Enactor, SLA High Level Services, VO Management, etc) that have to use/invoke other services; furthermore, it will be possible that also other kind of entities (virtualized as services) belonging to other WP4.X Layers may assume this meta-role;

- **Business User** : is the actual user of the bought service; typically, it is necessary to account his service usage;

- **Service**: is only able to provide functionalities. The service will be related to some constraints (access list), in order to enforce the authorized access only. We can introduce also the following specializations:
  - **Management Service** : is a service with management functionalities; so, it is necessary/responsible for some management operations inside the VO; as mentioned for the Management User meta-role, we can provide as simple examples (of entities emboding this meta-role) the list of the management services of the main components of the WP4.4 Layer (or, if necessary, lower layers);
  - **Business Service** : is the actual bought service; typically, it is necessary to account his service usage and his QoS;

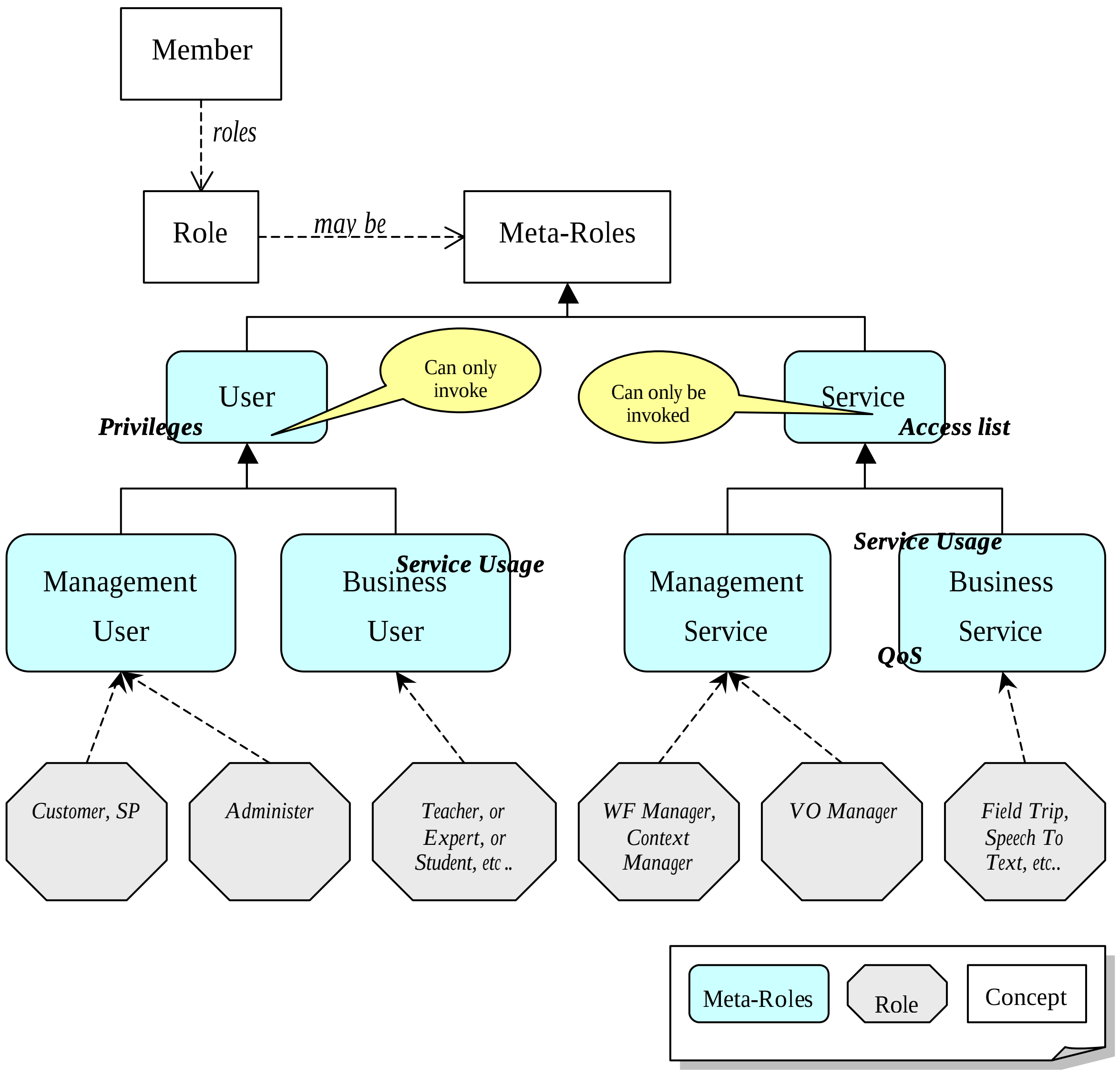


Figure 7 Member and Roles

During the registration phase, the new VO member (or its representative such as the VO manager) will provide information about one of these VO roles it is going to cover together with the policies enabling to interact with (e.g. security, payment and negotiation aspects). Obviously the two operations (subscription and registration) can be treated as two actions common to BVO and OpVO.

Users wishing to register themselves should use the portal provided by their network providers or an application client incorporating the registration phase and available on a user's mobile/PC. By means of this portal, for instance, the user will be at first identified in the supporting network by using one of several possible identification mechanisms (e.g. knowledge-based, cryptography based, biometrics, secure signature, etc.; see paragraph 3.1.1 of [9]) and gaining a SAML token. This token will be utilized by VO manager to ensure itself of the user's identity and for instancing a new VO session. The necessary credentials depend on the role being requested by the candidate user.

Within a VO a 'participant registry' can be envisaged in order to enumerate all entities registered with the VO. Each entry of this registry should be enriched with metadata that explains what is the role of this participant inside the VO. It is possible to envisage "some" intelligent knowledge creator system that, when the participant acts inside the BVO (interacting with another participant through the OpVO mechanism), could update this registry with new metadata, describing which kind of relationships are possible to be created involving the specific participant.

At the end of this phase, the new VO member will receive some credentials (see Figure 8) that allow access to the VO on subsequent occasions (log-in phase) and acquiring/using/providing services in according to its role and rights.

2.2.2. Log-in and Log-out

After the subscription/registration phases, the new entity (user, service, SP, etc.) is allowed to access VO functionalities and be recognized by other members. The log-in phase comprises the authentication and deals with the submission of registration certificate by BVO member to BVO management services; subsequently a new temporary token is provided to member enabling it to acquire, use or sell services (or be used in the case of service). The lifetime of this token defines the *session* at VO management level.

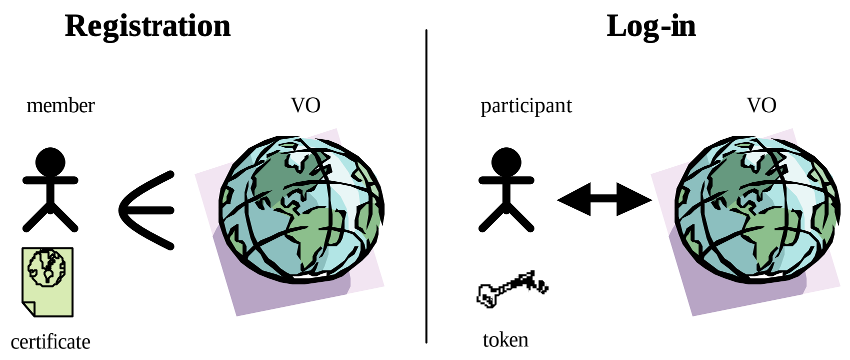


Figure 8 Registration and Log-in

As depicted in the Figure 8, at the end of registration phase the new entity is a *member* of the VO (BVO or OpVO) and a certificate is released to it in order to subsequently proceed with the log-in. After the log-in. the member will become a logged member of BVO and is able to access Grid functionalities by means of its token.

2.2.3. Publishing

When the SP has become a BVO participant, after subscription and registration to the BVO, then new services could be published into the GrSDS [9].

The GrSDS is a “static” discovery registry (it could be an UDDI based registry with some “semantic addition”) where the static description of services (i.e. WSDL description) is published. The SP, acting as BVO participant, will publish its services in the GrSDS. To achieve this aim, the SP has to provide a contract template (i.e. SLA template, where are collected QoS info, charging policies and service requirements) associated to each service and published against a SLA-Template repository. These interactions are represented as in Figure 9 with the grey arrows.

Of course, in GrSDS, some relationships must be created between the published service and the SP. This could be useful when a customer wants to set up an OpVO, in order to make the best match between customer needs and provider capabilities, considering in case membership rules related to the specific SP.

We describe another interesting interaction mechanism for the publishing: the GrSDS could receive data from information source coming from the SP local environment and storage this data allowing also to retrieve them in other context (e.g. during the service buying phase). This interactions are represented in Figure 9 with the blue dashed arrow. Following the WSRF model, the information source could be WS-Resources that simply have a status and an interface specifying its capabilities and how to interact with. In order to make other VO members aware of the presence of these services, their interface should be localizable within the GrSDS. This publishing phase can be realised using the notification mechanism provided by WSRF specification with a previous recognition phase of information sources and subscription phase to their notifications; this phases may happen when a new participant wants to register itself to the BVO, as a member with the role of SP, providing the required information to refer and to subscribe to the his information source.

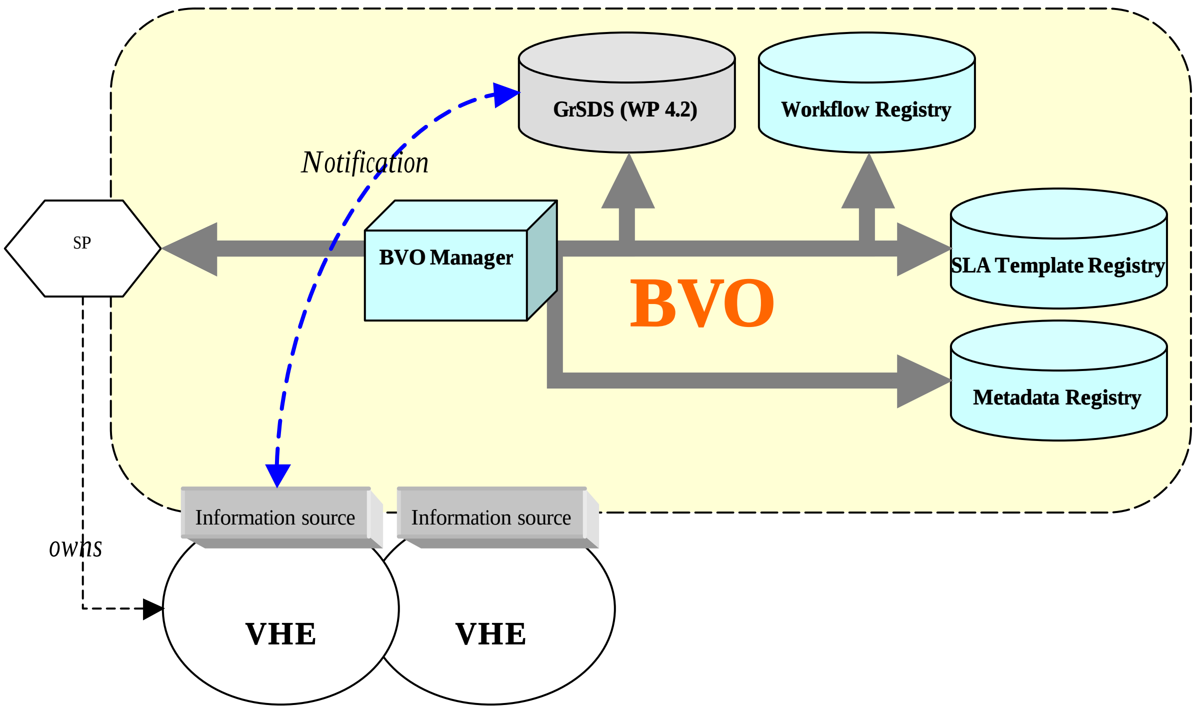


Figure 9 Service publishing phase

In Figure 9 the main components that come on stage are depicted. The BVO participant (i.e. the SP) knows the information about the service to publish and will indirectly interact (the grey arrows in the figure) with a service registry for storing service functionality interfaces as well as with the SLA-Template registry for storing its service requirements parameters. Again, adding metadata like semantic information could be also provided in order to improve the research capabilities on services, for this reason we can foresee a metadata registry.

In the case that the service has a more complex nature, by being a composition of some services that compound the overall functionalities for the service, the compound service can be described by means of an orchestration or workflow language with semantic capabilities for specifying what type of services the workflow composes together. In any case the compound service will be represented with a classical service interface published inside GrSDS. In this case the provider of this service will be represented by the reference of the Workflow Manager that is able to execute it.

2.2.4. Acquiring a service

As default step, in Akogrimo vision, a service consumer has to buy a Business Service (or have it bought by a SC) before he can use it; this fact gives reason to our introduction of the SC role. So, with respect to ordinary VOs (in OGSA vision, [14]) that support and enable the availability of services to all their members, we foresee a set of negotiation (both economic- and QoS-oriented) services in order to reach some agreements and to match consumer-service requirements. Obviously, some services provided by third parties can be free-of-charge, but also for these services a negotiation phase must be accomplished. In this last case we can talk about "light negotiation phase" because the final consumer will not pay for their usage, but the service should be guaranteed with some QoS requirements or in the worst case with a best practice QoS requirements. In any case this should be specified inside the SLA template and in the final SLA contract.

The result of this phase, which takes place in the BVO, will be the creation of an OpVO composed by services required by customer and bound to users making use of them.

Who really has the rights for service acquiring is the BVO participant assuming the SC role.

Before starting the description of the steps to buy a service we show how data are stored. Each business process appears as a simple service (web interface), which means that it is published inside the GrSDS, but some metadata of the published information is different (e.g. the SP of a business process will be the Workflow Manager that is in charge of finding and associating the right workflow template to execute the business process). Inside the Workflow Registry there are all workflow templates useful to execute the business process and additional metadata to manage semantic information. The Workflow Manager is in charge to find the best match between business process discovered in GrSDS (based on service customer input parameters) and workflow templates stored in Workflow Registry.

We will explain steps in order to provide a complex service (business process). The main steps can be summarized as follows (see Figure 10):

1. SC (belonging to BVO as participant) requests the acquisition of a business service passing through an access point (maybe a portal or an agent that recognizes and forwards the request).
2. The request arrives to the Workflow Manager (inside BP Enactor block) – step 1. This entity receives the metadata describing requirements on the requested service (that could be complex or simple) and make a search of it on GrSDS.
3. If the requested service is a complex service (behind it there is a workflow that involves different component services) the SP is the Workflow Manager itself. Then the Workflow Manager chooses what is the best workflow template in order to provide this service. It will decide what are component services involved into the workflow and provide a list of them to OpVOBroker (inside VO Management block) – step 2.
4. In the case the service requested is a simple one, the workflow manager recognize a different service provider and forward the request to the OpVOBroker – step 2.
5. The OpVOBroker receives the list of component services and makes a search of their providers in GrSDS – step 3. Then based on the returned list makes some internal ranking (on the base of profiles, credentials, and other available info on customer and provider) and tries to negotiate for each component service until there isn't a SP able to provide them.
6. The SLA-Negotiator is involved into the negotiation phase (step 4) in order to check the resources availability to provide business service. When all component services are negotiated the reservation will be made (step 5) and the OpVO creation process starts. Otherwise if OpVOBroker isn't able to negotiate for all component services, the control comes back to Workflow Manager that will provide an alternative service/s to be negotiated. If there isn't any service to renegotiate the process is stopped and the service can't be provided with the required QoS.
7. When all component services are negotiated the Workflow Manager contacts the OpVOManager in order to set up the OpVO with the right access policies.
8. The Workflow Manager subscribes itself to context change of business service and to violation monitoring service.

9. At the end of these steps an identifier is returned to the customer to the access services in the new OpVO.

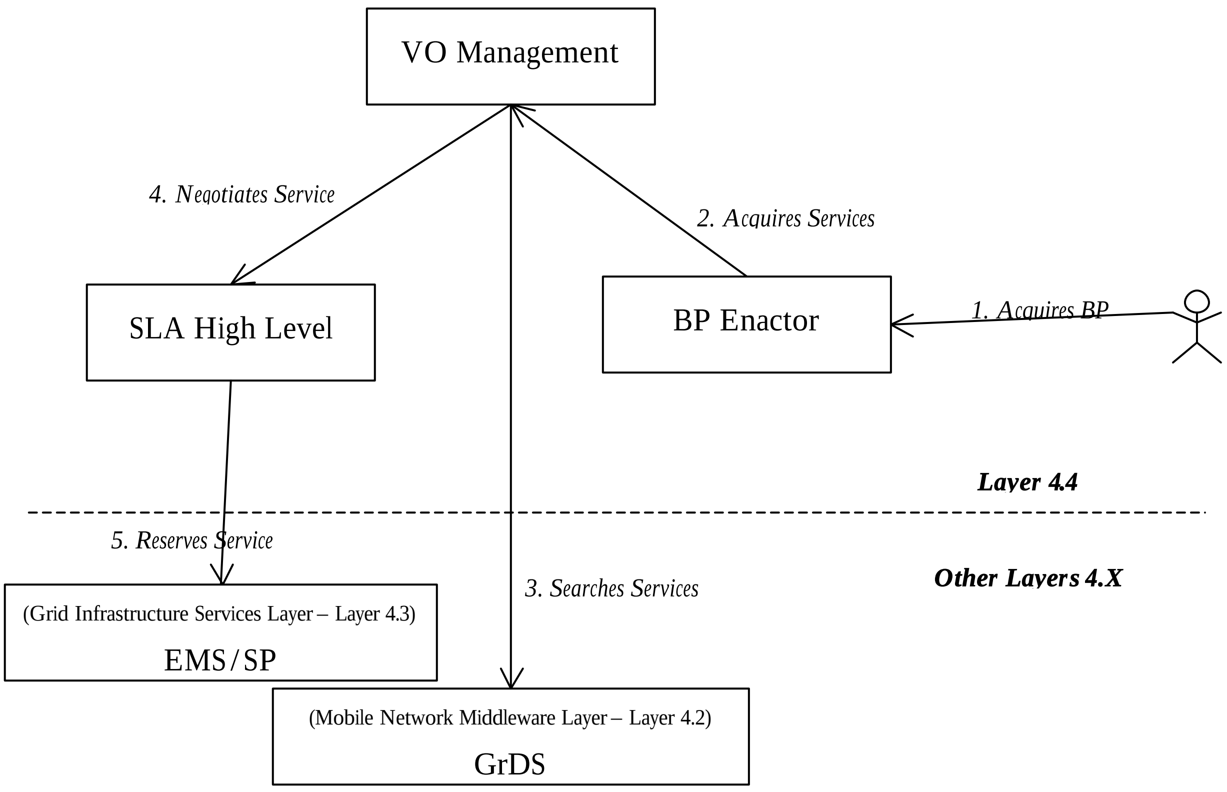


Figure 10 Collaboration diagram for Acquiring BP

We notice that in the previous steps description, we have hidden intentionally the interaction related to the security issues, to provide a more abstract and simple view.

2.2.5. Using a service

After buying a service, the SC may decide the actual users of its services and add them to the OpVO, using management functionality provided by the OpVO Manager. This operation is the registration or join of new member to the OpVO.

This registration implies also a check to verify that real users can be added within the workflow execution to match their profile (or part) against the BP-role of process definition.

Further, a member can actually utilize this service only if he has logged-in to the OpVO.

During the service purchasing, the set of possible users is also populated with the constraints and capabilities they have. For instance, in the e-Learning scenario a professor can require to buy a service to his university (as a BVO member SC) that operates on his behalf by also issuing the possible users such as students, professors or experts. This Customer can be embodied by University Institution providing e-learning services and bridge for his students, professors and technical to (Base) VO. Once a service is bought and possible users are enacted, a student wishing to access Field Trip functionalities should require the OpVO Manager to enable his access and, after profile checks, the OpVO will arrange some supporting services to take care of the user's requests. One of the main purposes of these supporting services is to ensure the user security accesses.

In a more complex condition such as a workflow where several SPs are involved in a common OpVO, a SP can need functionalities of other SP so that the first one becomes the "user" for the required services. Differently from the users/humans presence, the SPs as users are already well known to other SPs at the buying phase, therefore a strict binding among these SPs is created to provide the overall functionalities for requestors in the same way of using a unique SP.

At last, the participants involved within an OpVO assume (also dynamically) one or more roles defined during the service publication and different from roles defined for BVO participants (see paragraph 2.2.1). The OpVO roles represent the capability to invoke/offer functionalities from/to other OpVO members. So, these roles can be viewed as the roles at BVO level specialized for a specific business domain. For instance, returning to Field Trip scenario for e-Learning domain, the OpVO can involve some services such as the FT and some users such as the professors, the experts, the students, etc.; in this case a professor role can have the right to utilize some functionalities of a service that a student can't. These roles are application-specific because who defines services that will be involved within the OpVO is in charge of modeling these roles for the possible participants.

2.2.6. Service destruction

By service destruction we mean the dismissal of the corresponding OpVO and the releasing of its involved resources. Because the OpVO is a transient environment that can be viewed as a "specific business service VO" with a possibly short life-time living, in a natural situation the destruction can be issued in three ways:

- When the goal is successfully reached
- Time-expired: the prefixed time-living for the service established during the buying phase is over (we refer to service as which the customer has bought).
- Explicit request: the user having the rights (SC) can request the immediate or scheduled destruction.

As consequence, all resources reserved for the OpVO are released and some BVO reassessment are accomplished (e.g. revoking some users' rights) so that the functionalities can be no more accessible by users.

3. Detailed architecture description

3.1. VO Management

3.1.1. Overview

3.1.1.1. Objectives

The Grid presents an opportunity for a type of organization that crosses conventional organizational boundaries and offers a collection of services that require the cooperation of several conventional organizations – VO. In Akogrimo, this goes further: with a MDVO, people and services are mobile, either changing their location or their connectivity. The immediate connection for a device can be passed from one access network to another. Supporting devices could be mobile, or could be pervasive allowing a person or service to connect when in the vicinity. In this document, we drop the word “mobile” but always assume it, unless stated otherwise.

A VO has participants who may be service users or providers or customers. Participants take on roles and there are policies for associating roles and participants with rights, restrictions and responsibilities within the VO – sometimes the word member is also used. The procedures for establishing the environment for service usage and provision are grouped under the heading of VO Management.

Section 2 describes the life cycles of members and services.

3.1.1.2. Example Scenario

3.1.1.2.1. eHealth testbed organizations

In the eHealth testbed validation document [1], several conventional organizations are involved. For convenience, we summarise below the essentials from that document as they affect VOs:

- A university hospital, regional hospitals, medical specialists, general practitioners, emergency medical services and an emergency dispatch center establishing a regional health network (*RHN*) – the initials differ from those used in [1].
- The RHN is headed by the university hospital and provides telemedicine services to the partners and the patients attended by partners of the network. These services are focused on particular diseases and risk groups.
- For providing the services the RHN collaborates with a health service provider (*HSP*) and a network operator (*NO*).
- The NO hosts an infrastructure to provide telemedicine services over its network. It offers computational, network and data collection services.
- The HSP distributes the telemedicine equipment, provides advanced medical analysis services, configures application services specific to the health network’s needs and is responsible for the patient-side accounting and billing. If the customer of the eHealth services is the hospital that in turn offers these services to its patients, the hospital will be accounted and billed. If the hospital provides accounting and billing to each patient, it can use accounting reports of the eHealth services, which specify usage for each patient, and itself computes the bill.

- The heart monitoring and emergency service (*HMES*) is one of the services provided by the regional health network and addresses the patients with an increased risk on suffering a heart attack or apoplectic stroke. Patients belonging to this risk group are subscribed to this service. The main objective of this service is the early recognition of heart attacks or apoplectic strokes and a proper treatment as fast as possible. In the case considered here, just one task is offered by the service. During the treatment process of a patient in this risk group the attending physician can recommend the application of a permanent monitoring service to observe the cardiac function that assures a more detailed diagnosis, a fast alert triggering, and a disease-specific response in the case of an emergency.
- In the testbed, the people involved also include an emergency *first responder* and the police services, but we do not require them in the explanation of the services in this document.

3.1.1.2.2. Scenario based on the eHealth VOs

It should not be necessary to model all the existing, conventional organizations as VOs. We therefore outline here some possible candidates for a VO in the eHealth scenario, which involve conventional organizations pooling resources. Note that these are not necessarily the same as in the section 3.5 – because here we are simply using the examples to illustrate the possibilities of VOs.

- *EHealth BVO*: Below we distinguish between BVOs and operative VOs, where a BVO provides an environment within which operative VOs may be created. For the eHealth scenario, we have a BVO which needs to include services and users from the RHN, the HMES and the Network Operator, which offers computational, network and data collection services.
- *Patient Monitoring VO*: Patient monitoring is the subject of an Operative VO service. In principle the infrastructure should allow the possibility of either a separate VO being set up for each patient monitored by the HMES or one patient monitoring VO for all patients. This issue is discussed in Annex A. We assume the former in the discussion here. Although there is not an emergency, when this VO is first required, reducing the start up would be an advantage. We can make the assumption that, apart from the patient, all participants in this VO are already members of the eHealth BVO and authorisation and authentication for them has been successful. So the Akogrimo infrastructure should make it possible for authorisation rights and restrictions to be derivable from an existing VO, the eHealth BVO. Another distinction between this VO and the previous one is the need to be able to run a workflow (implementing a business process) – the term Operative VO (OpVO) will be introduced below. Monitoring equipment attached to the patient and accessed via the phone can be modeled as services and therefore requires mobile services.
- *Patient Emergency Response VO*: This too is the subject of an Operative VO service When an emergency is declared, organizations not involved in patient monitoring need to be involved. Here a speedy start up is required and some automation becomes critical here, while still maintaining accuracy and resilience. Additional sources of patient monitoring data (e.g. in an ambulance) will be required with additional mobility requirements and the patient's sensor data may be connected to the ambulance communications equipment instead of his mobile phone. Merging of several data sources will be needed, including several sources of mobile monitoring data and the historical database on a fixed server. Here too there could be just one OpVO for the function handling multiple patients or one for each patient.

The Testbed Validation Scenario document [1] contains a list of actors, roles and services for the eHealth testbed and these are associated with particular VOs. An actor may be an organization or an individual.

3.1.1.2.3. Elearning testbed scenario too

A future version of this document will consider VO management from the point of view of the eLearning testbed scenario.

3.1.1.3. Functional Capabilities - Concepts

This section outlines concepts and properties of a VO in Akogrimo. The first bulleted list contains quite familiar VO concepts, but their relevance to Akogrimo is discussed (section 3.1.1.3.1 onwards). The second bulleted list below contains concepts introduced in Akogrimo and are described from section 3.1.1.3.8 onwards. There may need to be some adjustment when existing plausible implementations are examined in more detail, except where fundamental Akogrimo principles would be compromised.

After the concepts are listed, they're discussed in more detail.

- VO owner (e.g. an administrator in a university hospital)
- VO Manager
- The identity of a VO
- The conventional organizations and individuals participating in the VO (e.g. a regional hospital or a doctor there)
- The services offered by a participant
- Participant's role – organizations have roles and so do individuals (e.g. a doctor at a regional hospital) - an owner could be a predefined role
- Policies associated with a role
- User agent
- Service agent

This 2nd list contains concepts which are introduced in Akogrimo.

- A BVO: this was introduced earlier in the example of the eHealth BVO
- An OpVO: this was introduced earlier in the examples of patient monitoring VO and patient emergency response VO. The first list of concepts is presented below with a BVO in mind; variations on this are presented in the description of the operative VO.
- An OpVO Manager
- An OpVO Broker

3.1.1.3.1. VO owner

An individual initiates a VO and as a result becomes the VO owner. The owner sets up an initial set of roles and policies. For example an administrator in a university hospital could set up an HMES and become the owner.

It is required that some VOs (for example, patient emergency monitoring service) be set up quickly, thus adding an aspect of dynamism that goes beyond that commonly associated with VOs.

In other work on VOs (e.g. TrustCoM [55]), the two separate operations, (i) identify the requirement for a VO and (ii) form a VO. The former is a necessary preamble, but need not be part of the Akogrimo infrastructure.

It could be argued that anyone could set up a VO and become the owner and hence define the rights and restrictions of anyone else wishing to join. There could be security implications where unsuspecting people are trapped into giving away details of their identity (like spoofing of email messages from banks). It is suggested that this problem be ignored in the Project's 1st cycle.

3.1.1.3.2. VO Management and the identity of a VO

It must be possible to distinguish one VO from another globally, to ensure global distinctness on creating a VO and to allow its participants to identify it when VO Management operations are required. This concerns the identity of the VO. There are several ways of implementing the VO identity:

a) Each VO is identified by being associated with a unique VO Management service instance, i.e. the VO **is** the service instance. In all other alternatives, a single VO Management instance could manage multiple VOs.

b) The VO could be identified by an XML Namespace. However the namespace normally refers to the specific XML-based language, not an actual XML document. Furthermore, the URI identified by an XML namespace is allowed to refer to a non-existent document and that makes it hard to check whether or not the VO already exists.

c) Implement as an XML document at a specific URI – with some suitable prototypical tag such as <AkogrimoVO>.

d) Implement as a registry of VOs: if it's not in a VO registry somewhere, it's not a VO.

Considerations need to include scalability, redundancy (possibly using replication – consider the mobile, possibly disconnected situation), multiple simultaneous operations and interworking with non-Akogrimo VOs.

It must be possible for the VO Owner to choose whether the identity of the VO is to be private. This may mean allowing the possibility of encrypting all VO Management messages.

It may also be desirable to allow VOs to be published. How should this be done? Possibly using the Service Discovery mechanism, which implies that the VO would be implemented as a service (see option (a) above).

In option a), a single VO Management service instance is responsible for managing the resources associated with the VO.

With the other options, the VO identity and resources are dissociated from any particular VO Management service instance. In principle a service from a different SP could be invoked each time a VO Management operation is required and would be passed the details of the VO identity.

3.1.1.3.3. VO participants

Participating individuals

Section 2.2.1 introduces subscription and registration of VO participants. An individual participant may or may not be associated with an existing conventional organization. Medical staff would be associated with a hospital, but a patient is a patient whoever they work for. (In some situations and in some countries, a patient would need to be a citizen of that country or be part of some national health insurance scheme or have sufficient financial reserves, so some authentication would be needed, even for the patient – some decision on the authentication of patients needs to be made – preferably a simple one).

When an individual attempts to join a VO, some form of identification/authentication and character reference may be required in order to allow them to fulfil a particular role in the VO. Certain roles (e.g. a patient, whose only access to data is the monitoring equipment allocated to them) would require a different degree of security protection than others (e.g. a database manager).

This introduction of an individual in a BVO could be largely a manual process. An individual, who is associated with a recognized conventional organization already, may be allowed to join the VO by means of credentials made available by that organization.

Participating organizations

It may be possible to treat individuals and organizations (e.g. doctors and hospitals) the same way, in terms of their participation in a VO. This may be desirable in order not to proliferate the number of different kinds of entity in the system. However it is possible that individuals and organizations may need to be treated differently. For instance: different individuals in the same conventional organization who join a VO may use different authentication credentials from each other. An organization may have a code known to the VO. An individual in the organization could use an additional code, specific to each participant and provided by the organization. This additional code is transparent to the VO. In case of a problem, the organization will be responsible to the VO but the organization will be able to know who is responsible among its participants thanks to the additional code.

Participants' information

Some information about each participant needs to be kept accessible within the VO. For an individual, these include:

- The identity: used to distinguish this individual when entering into a logged in session with this VO
- Other information related to the individual's identity such as, name and conventional host organization
- Details related to authentication

- Contact details – either stored directly with the VO participant information or available outside the VO, by looking up via the participant's identity
- The possible roles which the individual can adopt within the VO.

This information persists throughout the lifetime of the participant's association with the VO. This is accessible for reading and updating, using as a key the VO identity and the participant's identity. This can be implemented using a participant registry, which can be created when the VO is created. This Akogrimo layer (WP4.4) is responsible for implementing the participant registry and scalability considerations suggest that this should be kept independently for each BVO.

When a participant intends to partake in a logged session with the VO, other information about the participant needs to be maintained. This includes:

- Network Home Address
- Context
- The role that the participant has declared when establishing access

Although ideally a logged in session implies a "continuous" period of access, to this is not necessarily possible due to spasmodic connections and one of the goals here is to maintain session, role and workflow status, despite connection discontinuities.

Recursive participation

In principle a VO may itself become a participant of a VO. Although attractive in general it may introduce unsuspected complications and might not be necessary for the testbeds, and so is not considered part of this design. However there does need to be some relationship between one VO and another (for instance a Patient Monitoring VO is related to a host HMES VO), but this can be handled in a different way which is discussed under OpVOs and under Relationships between VOs.

3.1.1.3.4. The services published by a participant

Services are potentially diverse and examples may include:

- Data analysis
- Access to data from sensors monitoring the patient: in the pre-emergency phase, connected by his mobile phone and in the emergency phase using a connection from the ambulance to the RHN
- Establishing and managing a conversation connection between a specialist doctor and the emergency staff
- Obtaining a text message response from a specific doctor to confirm that they are available

Further details of service publication are in section 2.2.3.

3.1.1.3.5. Participant role

Participants may be service providers or service consumers/users. To preclude complex statements about what a participant may do in a given situation, it is simpler to say that a new participant P may undertake possible roles R, S or T and at the present moment is accessing the VO in the role S.

We distinguish between different sorts of role, belonging to the VOs:

- **Infrastructure role**: Some roles are predefined and must always exist inside a VO, because necessary for the management of the VO itself – for instance the VO Owner.
- **Business role**: Business-specific roles may be established in a specific OpVO and represent the roles, defined in the application, that the customer needs to accomplish his business goals e.g. FirstResponder, DataAnalyst.

A role is defined using a set of generic terms, for instance:

- What can a given role *offer* inside a VO - services provided by someone in this role (*service* viewpoint)
- What can a given role *do* inside a VO – what services can be accessed by someone in this role (*user* viewpoint)

Roles need to be updateable and accessible and are relative to the specific VO. There could be a role registry, managed by a generic registry mechanism or could be stored with the VO.

3.1.1.3.6. Policies associated with a role

The purpose of having a mechanism for expressing VO Policy is to allow some flexibility in the operation of VOs and to allow the flexible aspects to be stated in some visible and processable document. An aspect of a VO which might be the subject of a policy is: which VO Role Descriptions are subject to which security restrictions?

VO Policies can be stored using a general Policy Manager service, which can be used to manage and store VO policies. VO policies and policy enforcement and decision points will also need to be provided.

Each policy domain needs to provide information in the form of a policy template and an example for VOs is provided in D4.3.1 Annex A.6 [8]

3.1.1.3.7. User and service agents

User and service agents are general mechanisms (in particular services) in order to check/grant access rights at VO level. For example a participating user of the VO is represented inside the BVO/OpVO by a user agent. To this agent is associated a role and VO services can check/assure user access rights inside BVO/OpVO. A service agent has similar behaviour. These are discussed in more details in the section on security (3.4).

3.1.1.3.8. Operative VOs and BVOs

In Akogrimo, an OpVO can be set up on demand in order to provide an operative environment within which a business process can run. Since it is able to makes use of services, policies, roles and especially rights and restrictions already established in a pre-existing VO (a BVO), an OpVO adds further dynamism in its initiation beyond that commonly associated with VOs. As described in section 2.1.1 multiple OpVOs may be associated with a single BVO.

Operative VO

The OpVO is a dynamic organization with a specific task: let a user or a service achieve a specific business goal. In general behind an OpVO will be executed a workflow template and the overall users and services involved during lifetime of OpVO will represent a business process.

For example, *Patient Monitoring VO*: A separate VO could be set up for each patient monitored by the HMES – but see also the discussion in Annex A.

BVO

A BVO is a quite static organization which actually has a broad task: let participants establish (search/negotiate) OpVOs and provide basic identity management (profiles, participant registry…). A service may be published and associated with a BVO but is not capable of execution within a VO until the creation of an OpVO which includes that service.

For example, HMES (This VO provides all the necessary HMESs) and Network Operator within the eHealth testbed. The services in an HMES VO could request services belonging to the Network Operator VO)

OpVO Initiation

Given a requirement for an OpVO, one can be set up based on a pre-existing BVO. An OpVO would require a set of participants fulfilling certain roles and a set of services provided by those participants. A BaseVO would already have been set up as an umbrella organization containing a range of participants for each role and a range of published services. The application designer would need to ensure that the BaseVO is adequately populated.

Forming the OpVO would proceed by inheriting the BaseVO roles and policies and by selecting a set of participants and published services already established and authenticated in the BaseVO. The criteria for selection (e.g. location, specialisms) would depend on the circumstances which led to the OpVO being required.

Some of this would proceed automatically, for instance authentication. Other aspects of this OpVO formation may rely on actual human negotiation to determine availability, although even this could be automated to some extent.

The OpVO could be created by means of copying from the BVO, thus setting up the OpVO as an independent entity. An alternative is that the OpVO is created by defining its relationship with the BVO. Thus any changes in the BVO are, if relevant, reflected in the OpVO (e.g. change of contact details for a specialist; or new recruit is immediately drafted into the OpVO). This latter alternative may be preferable.

A BVO could be regarded as a large organization, capable of forming multiple OpVOs each for a specific case. So in the eHealth scenario, the HMES could be a BVO which then is the basis for a Patient Monitoring OpVO, one for each patient.

Discussion has led to the preference that there is a 1 to 1 relationship between an OpVO and a Business Process. So, the purpose of a particular OpVO is to provide an environment within which it is possible to enact the business process with which it is associated. One possible problem with this is that it could lead to an artificial inflation of the extent of a single business process or an artificial specialisation of the purpose of an OpVO. So the business process associated with the Patient Monitoring OpVO might have to be titled: Monitor_The_Patient Whether this is satisfactory could be determined by working through some use cases.

3.1.1.3.9. An OpVO Manager

The identity and management of a VO was discussed in section 3.1.1.3.2 and a number of alternative implementations were discussed. A corresponding decision should be made here. In fact, where possible, an OpVO should be managed by mechanisms which are similar to a BaseVO in nearly all respects.

3.1.1.3.10. An OpVO Broker

This entity is responsible for business service acquiring. It is offered by the BaseVO and acts on behalf of final customer that wants to set up an operative VO. It is in charge to conclude the negotiation phase for each component services involved into the global business process asked for the customer. In order to conclude the negotiation it has to lead the search phase performed by GrSDS (belonging at 4.2 level) and to rank service providers list by using ranking criteria (e.g. based on matching between customer and service provider profile or using some additional input parameters provided by the customer, etc).

3.1.1.3.11. Relationships between VOs

The idea of a BVO and its associated OpVOs amounts to is a 2 level hierarchy of VOs. It is possible to conceive of circumstances in which more levels may be needed, if the organizational arrangements in the application are complex. Although this may be worth studying in general, the present design envisages just 2 levels, the BVO at one level being the parent, but not an ancestor, to multiple OpVOs.

3.1.1.4. Functional Capabilities - VO Management Operations

A VO has a collection of management operations associated with it. It is likely that these are all Grid Services.

- Form a VO and designate the owner of it
- Associate a role with a VO (alternatively associate with the BVO/OpVO a collection of roles in the form of a VORole document (in XML)) – allow changes and enquiries

- OpVO Broker operations.
- Form an OpVO by association with a BVO
- Associate a participant organization or participant individual with a role - allow changes and enquiries.
- Finally, dissolve the OpVO when the business process is completed (e.g. the patient doesn't need to use requested e-Health service)
- Dissolve a BVO (e.g. when there aren't organizations inside it able to provide services or the owner of VO doesn't need to maintain it up). Even though some BVOs might persist for a long time, the possibility of dissolving a BVO should still be available.

Many aspects of a VO can be controlled not explicitly by passing parameters but by populating suitable XML documents (e.g. for describing a role and its associated policies) and this simplifies the list of VO Management Services.

3.1.1.5. Position with respect to the Akogrimo infrastructure

3.1.1.5.1. Interactions with Akogrimo infrastructure components

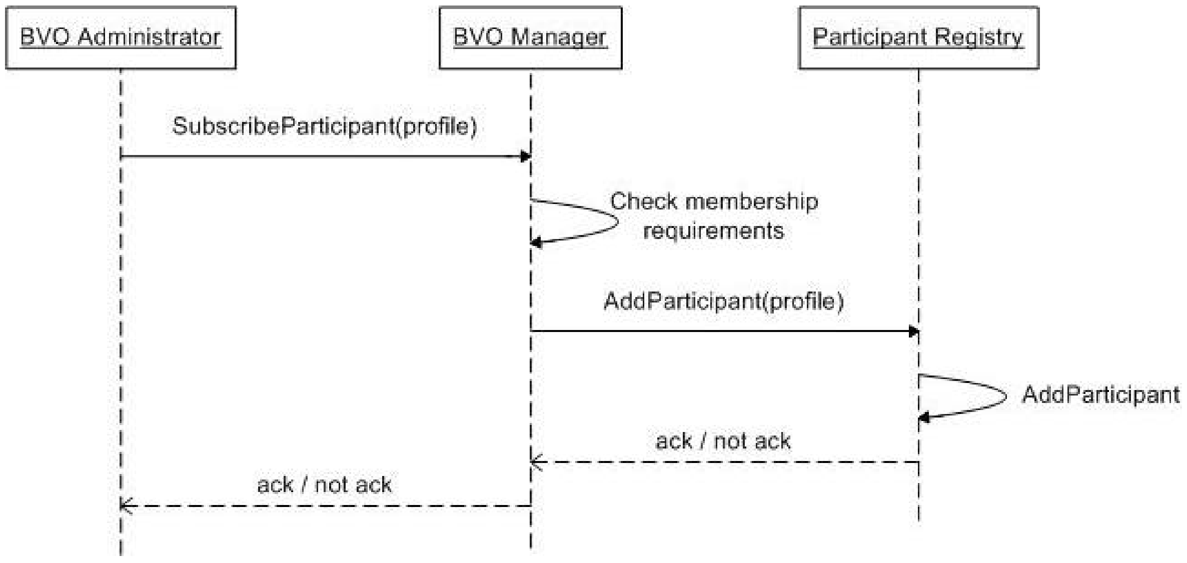


Figure 11 Subscription to the BVO

The subscription of a new VO participant should be requested by the BVO Administrator (a person) to the BVO Manager (a service) that checks some membership requirements and, in turn, requests the addition of the participant profile to the Participant registry (Figure 11).

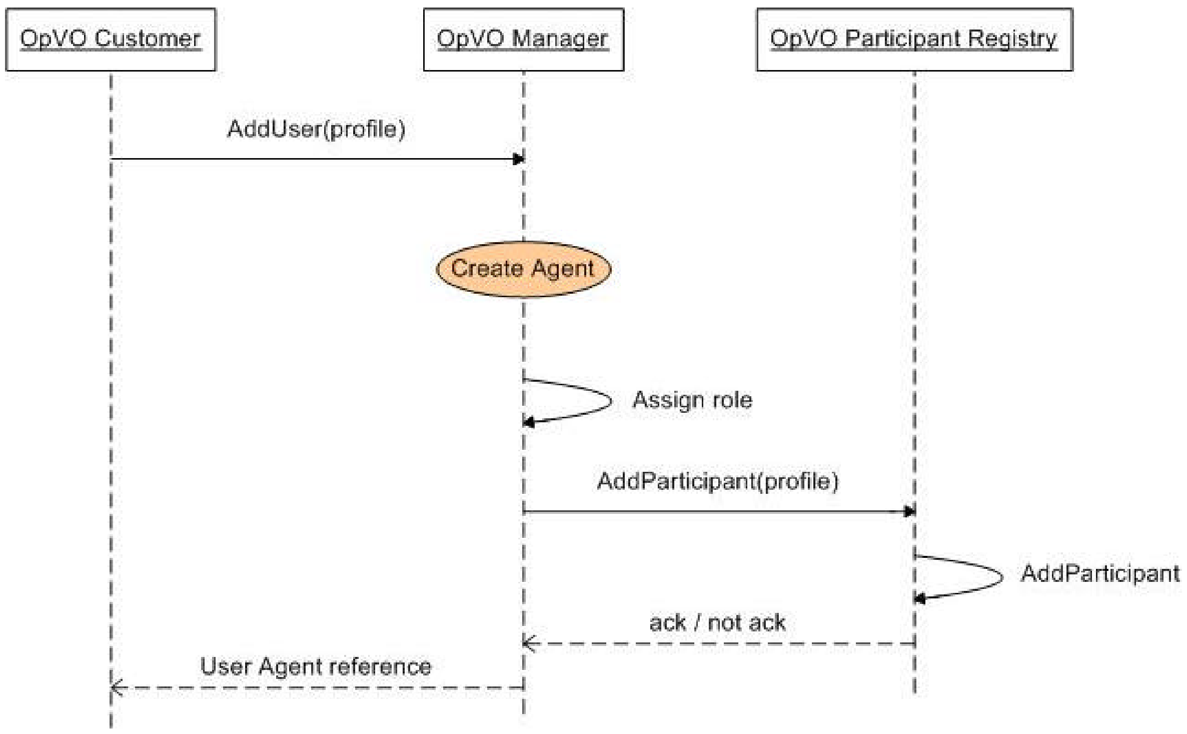
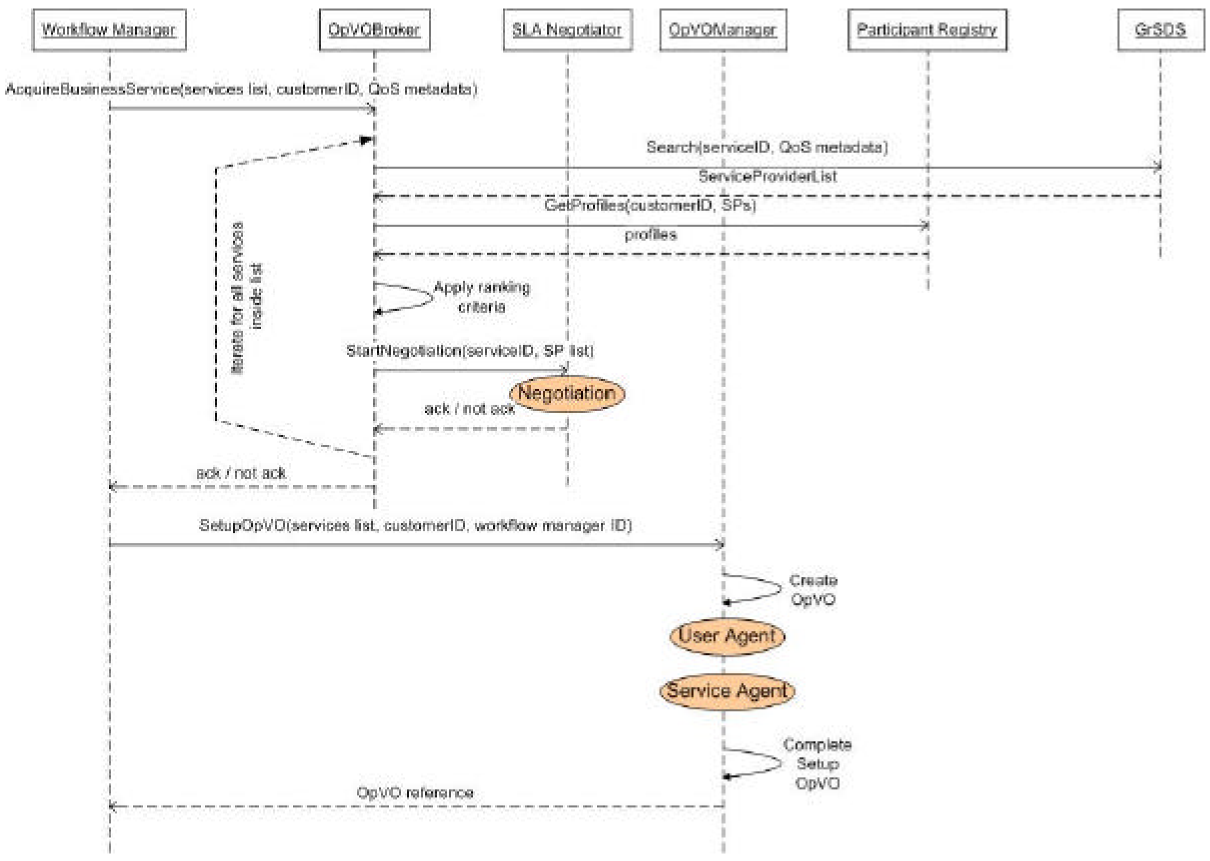


Figure 12 Add a User to an OpVO

When the OpVO is created, a new user may be added at run-time; this operation should be requested by the OpVO Customer (a person) to the OpVO Manager; this component will create a user agent (a service that represents the user in the VO - it is represented by a ws-resource) and assign a role to him. Moreover it requests the addition of the user to the participant registry of the OpVO (Figure 12).



Workflow Manager
OpVOBroker
SLA Negotiator
OpVOManager
Participant Registry
GrSDS
AcquireBusinessService(services list, customerID, QoS metadata)
Search(serviceID, QoS metadata)
ServiceProviderList
GetProfiles(customerID, SPs)
profiles
Apply ranking criteria
Iterate for all services inside list
StartNegotiation(serviceID, SP list)
Negotiation
ack / not ack
ack / not ack
SetupOpVO(services list, customerID, workflow manager ID)
Create OpVO
User Agent
Service Agent
Complete Setup OpVO
OpVO reference

Figure 13 Acquire Business Service

Figure 13 outlines the phase of acquiring a business service. The Workflow Manager, having determined which services are required, passes the list to the OpVO Broker and asks it to return a list of corresponding specific services satisfying certain criteria, which could include some QoS metadata (may be also some budget constraints). The OpVOBroker is in charge of looking for (using GrSDS component, of WP4.2) available services, applying some ranking criteria and then asking the negotiation to the SLA Negotiator. (Obviously, this sequence shows only the main kind of interactions between components and omits some detail). When the services are found, the OpVO Manager may ask for the creation of the OpVO to the OpVOManager that will set up the OpVO with the required agents for users and services.

Identified interaction	**Rationale for interaction**	**Issues to be addressed**
BVO Manager – VO Security subsystem	Authentication and authorization	Define what functionalities belong to security subsystem and what others are covered by BVO / OpVO Managers
OpVO Manager – VO Security subsystem	Authentication and authorization	
OpVO Broker – SLA Negotiator	Negotiation and contract	
BP Enactor – OpVO Manager	OpVO initiated by Workflow	

Table 1 Summary interactions between VO Management and other WP4.4 components

Identified interaction	**Rationale for interaction**	**Issues to be addressed**
Participant Registry – Generic Registry mechanism	To store participant information	Need to decide whether the VO Participant Registry uses the Generic Registry Mechanism or a registry designed for this purpose
VO Policy Manager – Policy Manager	To obtain low level policies	
BVO Manager - GrSDS	To publish a service	To decide what metadata needs to publish
OpVOBroker - GrSDS	To search published services	

Table 2: Summary interactions between WP4.4 VO component and components outside WP4.4

3.1.2. Architecture description

3.1.2.1. Architecture overview

The concept of VOs that span computing networks and working environments can be seen as the essence of Grid computing. VO's provide the logical architecture behind the distribution and management of services within Grid computing environments. Akogrimo's use of VO's in mobile computing environment has led to a redefining of the traditional VO's architecture.

Within Akogrimo the relationship between Virtual Organization's and GRID resources and services have been reviewed to reflect the mobile nature of the Akogrimo GRID. Within traditional GRID infrastructures the Virtual Organization is static and based on fixed resource. In Akogrimo the virtual organization involves mobile resources and is often required to be mobile in its nature.

This mobility in relation to the hosting environments making up a VO challenges the GRID in the way it manages and accesses services. The ability of hosting environments within a VO to drift in and out of range, change their IP address, suffer from fluctuations in relation to the quality of service of the VO are some examples of the problems that mobility poses to the GRID architecture.

Architecturally within Akogrimo this problem is addressed by the creation of two types of VOs: the BVO and OpVO. The BVO exists as a core VO in the sense that it contains specific key services in relation to workflow, security management, service level agreements and brokering. The Brokering service is required during the creation of the OpVO, it is used to rank possible services for use in the OpVO when executing a workflow.

The OpVO is a collection of services grouped in order to achieve the completion of a specific task. This set of services is not limited to the ones that are managed and owned in the BVO. The OpVO also includes services that are registered with a BVO service but are managed, owned and located outside the boundary of the BVO. These services in Akogrimo are often mobile in nature.

Services, BVOs and OpVOs are illustrated in Figure 14 below. The External services are illustrated as separate to the BVO services. This is because they are possibly not owned or constantly in connection with the BVO. These services are likely to be mobile in the model. The main time they become connected with the BVO except from registering is when they are being used in an OpVO. This is also illustrated in Figure 14.

The OpVO Broker selects services into the VO using rules from the external services and the key BVO services in relation to workflow and selection requirements. This takes into account characteristics of the services that are affected by mobility from location to network connectivity. As the broker can bring in replacement external services if others fail. Within the BVO the services are less mobile and generally are more robust and stable.

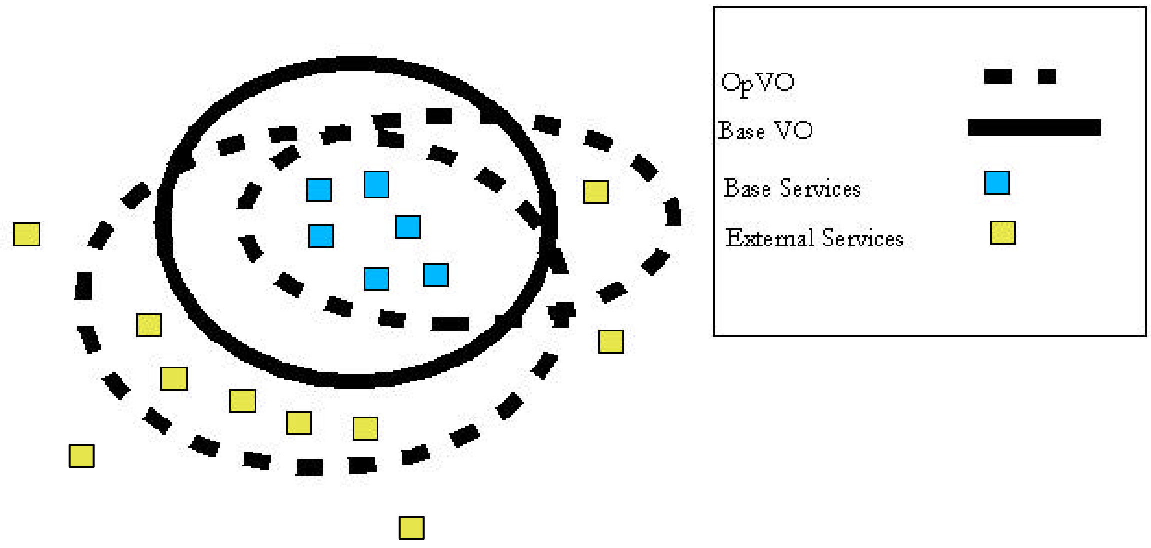


Figure 14: Services, BVO and OpVO's

3.1.2.1.1. Functional requirements

Below are what seen as the likely functional requirements/service of a VO

- *Membership Registry*. There needs to be a central registry for all services involved in a VO. This registry will be a store of service information from application attributes to network related information.
- *Participant Registry*. This registry will contain static data (i.e. profiles) of VO participant. This information is useful to manage participant credentials and to associate a role to each user that will access to an OpVO.
- *Management*. In the VO architecture there is a need for a management service. This service will act as a central point of VO management from security to the establishment of OpVO's.
- *Ownership*. In terms of legality and policy the VO will need a VO Owner.
- *Brokering*. Brokering is a key stage in the establishment of an OpVO. The brokering service will broker between services internal and external to the BVO when establishing the creation of an OpVO its remit and members.
- *Security*. Security will be implemented at all levels of the network from VHE to BVO level. In terms of VO the BVO will be the central point of control for authorisation of users and services, as will credential mapping for use of user service agents within OpVO's.
- *SLA*. Access to SLA repository and function in the VO is essential to aid the brokering phase of OpVO creation.

- *Grid Service Discovery Service GSDS*. This repository service (outside WP4.4) will be called upon during the discovery of services registered with the VO Manager.

3.1.2.1.2. Non-functional requirements

- *Twin speed Brokering*. Interaction between broker and internal and external services in the execution of workflows and tasks must be carried out in a fast efficient manner in cases where users are charged related to use time or the value of a deal can change over time (i.e. share price). In cases where this is not an issue and there is no financial charge or other overhead which requires fast brokering in relation to a service, a slower brokering process could be used. This could save the VO on brokering resources and also allow for the inclusion of other more detailed specifications in the brokering process.
- *Speedy VO creation*. The creation of VO's to execute tasks, in terms of establishing communication channels and security / roles between services must be quick and efficient.
- *Scalability*. Each VO must be able to adapt to large numbers of members either leaving or joining over long or short periods of time. The VO must be robust enough to be able to manage and monitor the demands of its member services. Also the Architecture must be able to handle multiple base and OpVO's.
- *Change Management*. In cases where the network /service attributes change within the VO this change must be managed, for active services and non active services in Akogrimo.
- *Network fault tolerance*. Faults in network connectivity or service execution must be catered for within the VO. The ability of mobile devices to drop out of range must be expected to occur in the architecture and the VO must be able to deal with this.

3.1.2.2. Use case view

3.1.2.2.1. Actors

A list of actors involved in VO management use, cases are:

- Service provider and network provider
- BVO Administrator
- Workflow Manager, SLA Management entities
- GrSDS (from outside WP4.4)
- OpVO customer
- OpVO Manager

3.1.2.2.2. Use Cases

Below are described main use cases that address VO management functionalities.

Use Case	**Subscription and unsubscription**
Description	This use case describes how to add a participant in Akogrimo BVO. The participant could be a NP or a SP, but also a simple participant as a final user. Flow of events: • In the first part of use case SP or NP asks to BVO Administrator to add itself, as participant, inside the BVO. During the unsubscription phase a trusted part entity is removed together with its users, recognized as participant of BVO. • In the second part of use case a simple user is added as BVO participant throught a trusted entity recognized by BVO Manager. A user could be also unsubscribed by its trusted entity.
Actor	SP, NP, BVO Administrator
Preconditions	BVO Administrator knows SP or NP. BVO Manager recognizes trusted entity that wants to add a “simple user”. In order to execute the unsubscribe action each participant has to belong into the BVO
Postconditions	A new participant is added to the BVO An existing participant is removed from the BVO

Table 3 Subscription and unsubscription

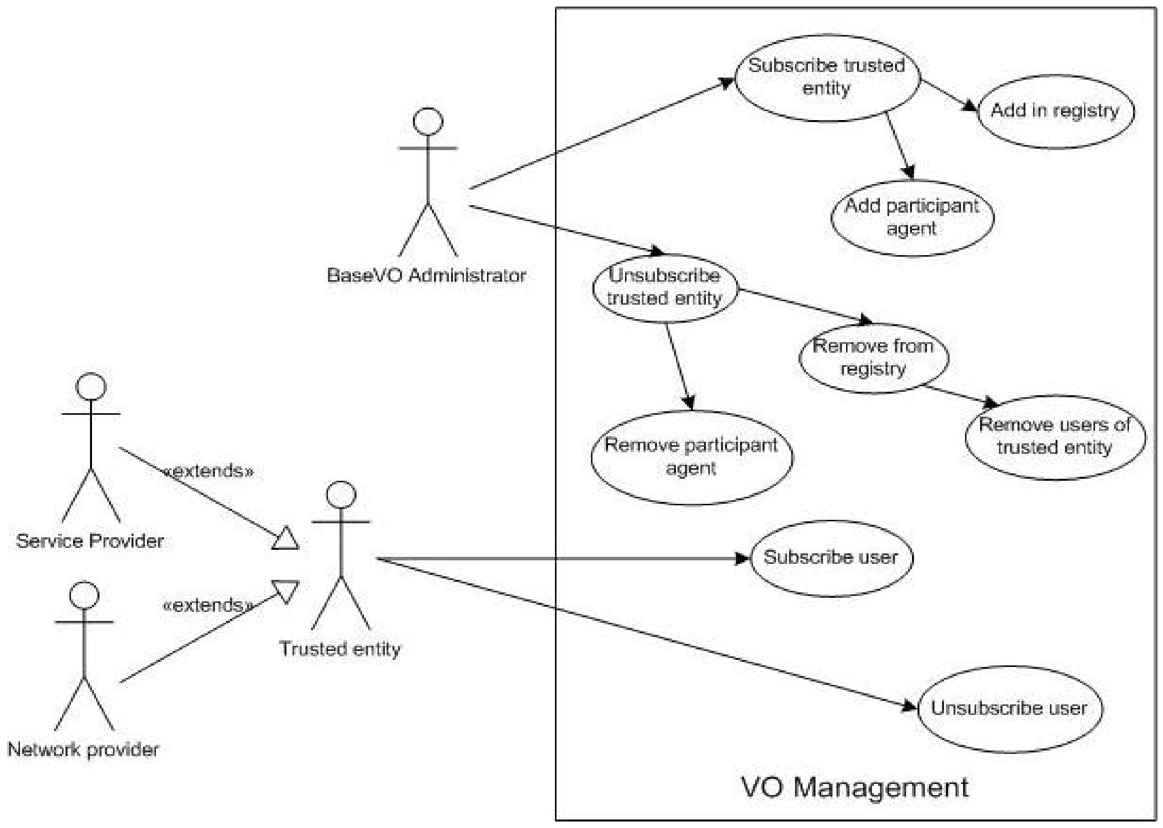


Figure 15 Subscription and unsubscription

Use Case	**Acquire a business service**
Description	In the use case are highlighted major steps when the Workflow Manager wants to acquire a business service (or a set of services). The subsystem has to perform a search on GrSDS and a negotiation for each component service involved into the execution of workflow or for the one service asked (if the business service isn't composed). At the end of acquiring service phase the Workflow Manager asks the creation of the OpVO. The Workflow Manager dissolves the OpVO, releasing all resources.
Actor	Workflow Manager, GrSDS, SLA Management
Preconditions	A SC wants to acquire a business service with specific requirements and uses BVO entities/functionalities to achieve task. A user doesn't need requested OpVO.
Postconditions	A business service is acquired and an OpVO is set up. The OpVO is dissolved.

Table 4 Acquire a business service

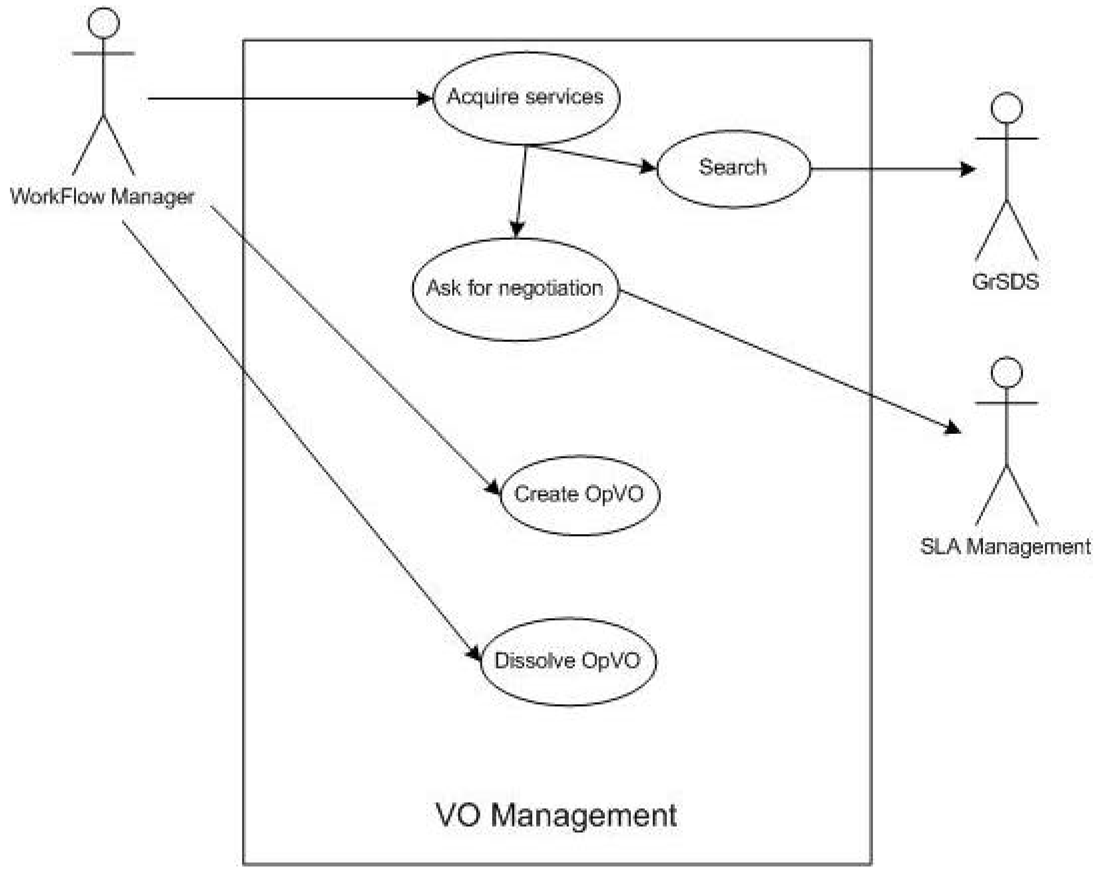


Figure 16 Acquire a business service

Use Case	Join a user to an OpVO
Description	A user wants to be added to an instance of OpVO in order to collaborate/take part in the execution of a business service
Actor	OpVO Customer, OpVO Manager
Preconditions	The user has to be a visitor (i.e. subscribed to the BVO) and an instance of the OpVO is available and set up
Postconditions	A user is added into the OpVO if he is recognized to have specific rights. The user will receive the reference to its UA.

Table 5 Join a user to an OpVO

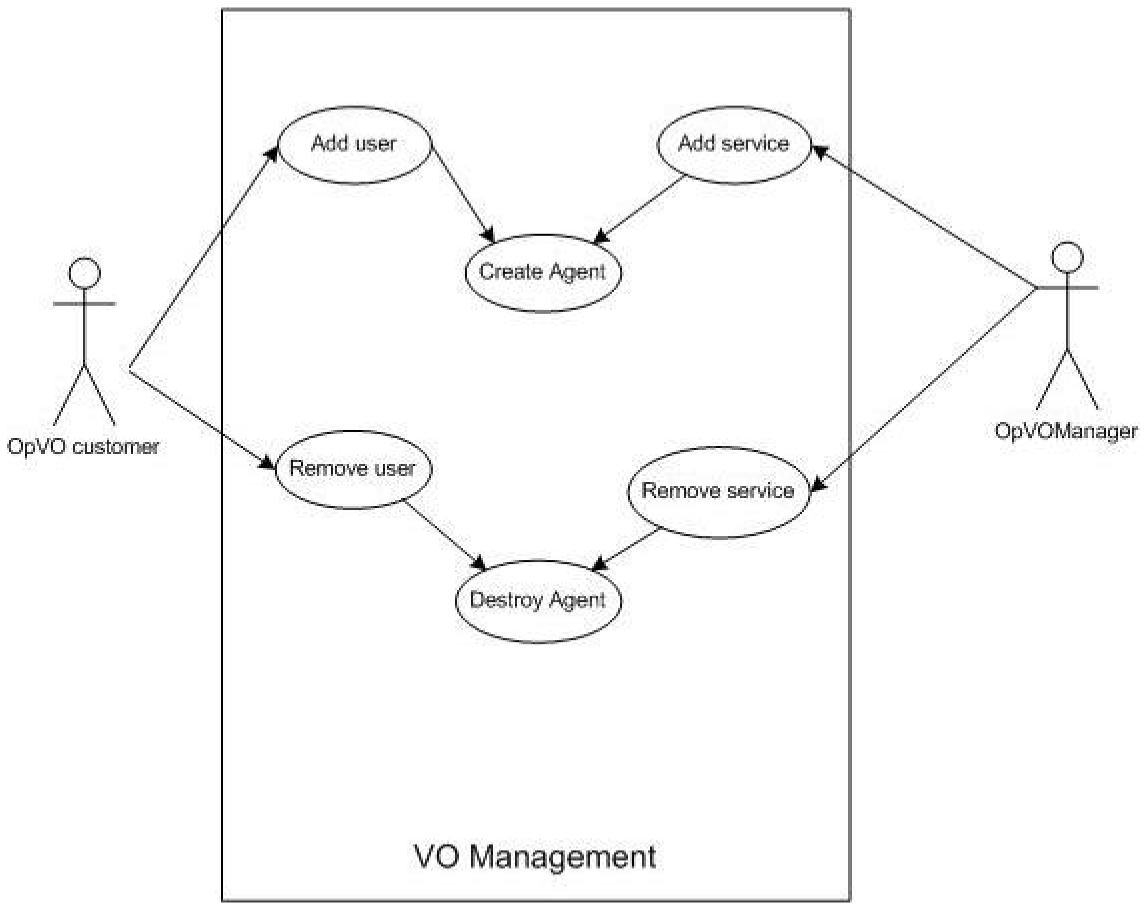


Figure 17 Join a user to an OpVO

3.1.2.3. Conceptual architecture description

As discussed previously the VOs within Akogrimo are split into two types. These are the BVO and OpVO. The BVO is central to all business processors that take part within any of the OpVO's associated with a specific BVO. It is envisaged the BVO's will be application specific. For example an architecture for e-learning could be split up and may consist of multiple BVO's for specific jobs including teaching, learning material and students. Off this each BVO could create multiple OpVO's to deal with separate lower level key tasks, for example in a student BVO, OpVO tasks could include roll call and homework distribution.

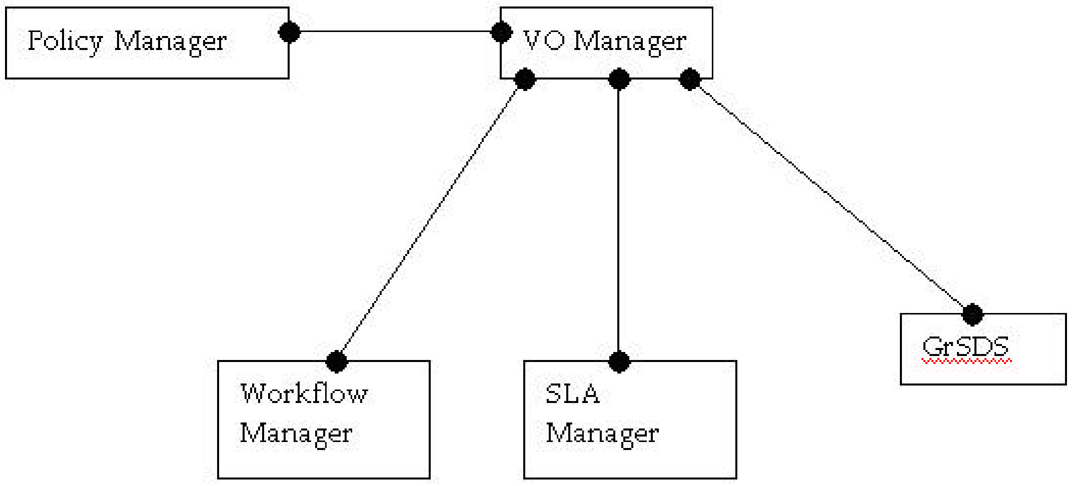


Figure 18: Core Akogrimo VO Management Services

Within the BVO there are specific services that have been agreed on in the architecture plans for the project, within the OpVO these services exist too, but also in OpVO's external services specific to the task are also included. Figure 18 shows the core VO management services.

Essentially the main responsibility of the services in the diagram is to enable the establishment of an environment that will conduct a specific task. Thus the key components reflect workflow, SLA, membership, brokering and security. In the process of task initialisation to completion it is envisaged each one of these services will be involved.

VO Manager

The VO Manager is key to the operation of the VO. It is the first point of contact for external services registering with the VO and also provides the first point of contact for users of the VO. The VO manager covers essentially functionalities for registration of users and services, but at the same time is high level coordination entity.

The VO manager's external interaction is with the VO Owner. They have the responsibility of providing membership rules for the VO, the VO Manager is the point of enforcement for security and policy within the model.

The VO manager has the power to instantiate an OpVO based on data it receives from trusted entities. It does this by setting up an OpVO broker, and establishing the OpVO Manager service.

GrSDS

This service (belonging to WP4.2 level) contains services published by SP throw BVO Manager or other alternatives mechanism (see 2.2.3) when the SP has been recognized.

OpVOBroker

The OpVOBroker service (belonging to VO Management block) essentially manages the negotiation phase, performing search operation on each component services involved into business service and provides ranking criteria to start negotiation with specific service providers..

The power to do this task i.e. security credentials and ID information of the task is passed down from the VO manager.

SLA Management

The SLA Management block contains information about Service level requirements specific to internal BVO requirements and external OpVO member requirements. Examples of internal SLA requirements can be seen to include rules associated with processing power/time used during the execution of the task, an external requirement could relate to a specific financial constraint or level of QoS expected. The SLA management provides this information in the SLA repository and it is enforced by the SLA enforcement component (located at 4.3 level). The SLA Management doesn't belong to VO Management and is described in detail in a separate section 3.3.

Workflow Manager

The Workflow Manager is essentially a service that gains information specific to the execution of a VO from external and internal sources to the BVO. The Workflow Manager contains a workflow repository of workflows located internal and external to the BVO.

The workflow manager's job is essential in this brokering phase, as information relevant to available workflows are central for a task to be completed. It locates the specific workflows and provides their necessary means of execution. In Akogrimo it is expected that BPEL and a BPEL engine requirement will be a part of the Workflow Management, at least for this first cycle.

In cases where the negotiation of one or more services involved into a workflow fails (e.g. can't be provided with specific QoS), the OpVOBroker requests from the Workflow Manager another list of services to negotiate. If the Workflow Manager is able to provide this new list then the acquiring process continues, otherwise it is interrupted because the requested business process can't be provided with requested QoS.

The Workflow Manager is in a separate subsystem from VO Management and is described in detail in a separate section 3.2.

Policy Manager

Finally and essentially in the BVO services security exists. This service is directly linked to the VO Manager and security policy is stored in the Policy Manager. In relation to a VO security is best visualised at policy level as opposed to a specific examination of potential threats.

Within the VO the security service will be relied on to provide rules and processes relating to the exchange of credentials in the system and Authentication, it can be best visualised as a partnership between the policy and VO Manager. It is envisaged that VOMS will have a role to play in the way security permissions are planned in Akogrimo, although essentially policy will be set directly by the VO Owner or indirectly by the VO Manager.

The initial key phases of Authentication and Authorisation can be seen as at the initial registering of a SP and associated services with the VO Manager, this process will be designed to establish a trust relationship. The authorisation phase will be relied on during the setting up of an OpVO and the provision of services within it; here external services like trusted A4C servers will be used.

Further information on Security and the Policy Managers role in its provision is described in detail in a separate section.

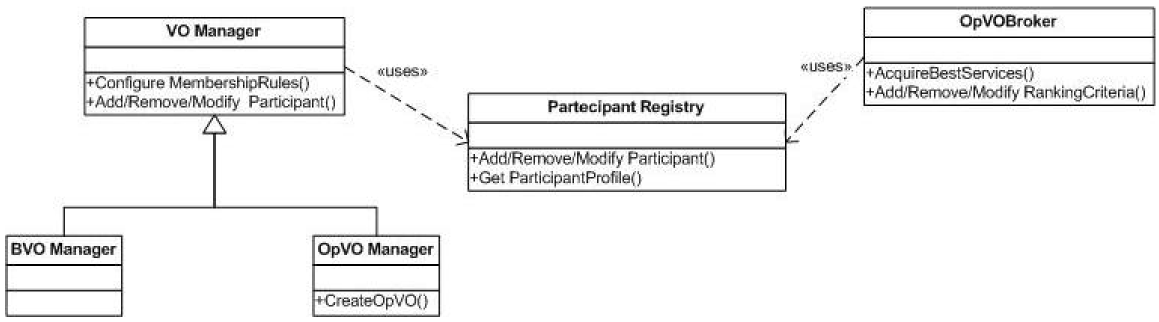
3.1.2.4. Logical architecture description

In VO management architecture we can identify different entities, that are able to address all aspect of management related Akogrimo BVO and OpVO. At the moment we can identify them with the following aspects:

- *Membership Management*. This block contains a registry where to store static profiles and roles of participants inside the Akogrimo VOs. Obviously the access to this registry is managed by a service that allow classical actions as add, remove and update of entries inside repository.
- *Brokering Management*. The OpVOBroker applies specific ranking criteria on the list of service provider that are able to provide specific services. This action is made in order to reduce the time necessary to complete negotiation phase, speeding up the process of OpVO instantiation. The action to rank service provider list can involve different entities and services in VO Management topic, because should take in account the relation existing between service customer and service provider, their credentials, access rights, profiles and so on.
- *Grant security accesses*. BVO Manager makes internal checks to determine what a specific participant is able to do, what roles can cover and in which way is bound to one or more running OpVO. Inside Akogrimo VOs we foresee an authorization mechanism for each participant (called also user) and service. This mechanism can be applied in the BVO as well in the OpVO and involves different base components of VO management like membership, participant management, role identification, credential mapping etc.

3.1.2.4.1. Main components description

The figure below shows the main entities of the VO Management. The BVO Manager and the OpVO Manager share same basic functionalities, related to the configuration and add/remove of the participants; these functionalities are summarized in the generalization with the VO Manager.



VO Manager
+Configure MembershipRules()
+Add/Remove/Modify Participant()
«uses»
Partecipant Registry
+Add/Remove/Modify Participant()
+Get ParticipantProfile()
«uses»
OpVOBroker
+AcquireBestServices()
+Add/Remove/Modify RankingCriteria()
BVO Manager
OpVO Manager
+CreateOpVO()

Figure 19 VO Management main components

For each component, we identify the functionalities:

ParticipantRegistry:

- Add/Remove/Modify Participant Profile
- Get Participant Profile

BVO Manager:

- Add/Remove Participant, the VO Manager is responsible to check if a given entity may become a member of the managed VO and then store his/her profile in the Participant registry
- Configure Membership Rules, allows the setting of rules for the membership, such as minimum requirements on the profile of the participants or other constraints useful to impose some compatibility between the members of VO

OpVO Manager:

- Create\Dissolve OpVO, due to the transient character of the OpVO, we need lifetime management of the OpVO;

OpVOBroker:

- AcquireBusinessService, this core functionality deal with the choice of the best service provider and the best service for the service customer; this task is achieved with the matching of their business goal/requirements using some ranking criteria;
- Configure Ranking Criteria, the criteria for the ranking may be dictated by the characteristics and goals of the BVO and so, granting flexibility, it is useful to configure these criteria

3.1.2.5. Involved technologies

At this layer we will map all resources and services as grid services, following WSRF specifications. For this reason we will use the most candidate underlying frameworks (WSRF.NET and GT4) that implement the WSRF specifications, in fact the major part of communications are SOAP requests over HTTP. Moreover we will manage some static information that needs to be stored in repositories and we can use RDBMS databases or xml native databases, on the base of specific needs.

3.1.2.5.1. VOMS

The Virtual Organization Membership Service acts as the interface for the user information database inside a Grid. Traditionally in computer systems this role of managing user data was conducted by LDAP. VOMS is a replacement for LDAP in Grid computing and offers an improved service in the management of user data in terms of security and functionality.

Aside from solving the various technical nuances Grid users face with LDAP (like mass updates and CN names crashing with a VO names), VOMS offers a new approach in the management of user data for the Grid. A good example of this is that VOMS provides separate instances for every VO in the VOMS database along with LCAS/LCMAPS as opposed to the gridmap file in LDAP.

The granting and verifying of credentials is improved in VOMS via the use of voms-proxy-init as opposed to grid-proxy-init. In this model the VOMS registration and VO servers are contacted in the process of setting up a job so all relevant credentials are matched to the user's request. Thus facilitates the update of the Information Service and removing the previous procedure of copying certificates that were needed to every node.

Within the VOMS' user management, additional attributes are used:

- The user's working Group (e.g. registration, integration, testing)

- The user's Roles (within a given "Group" i.e. inheriting the Group's privileges, e.g. production manager, user).

This allows the administrator to register members within a VO more effectively (members should provide their DN, CA and Group affiliation). So during the grid job submission when a member issues voms-proxy-init they get an extended proxy certificate from the VOMS server, including the member's Groups and Roles as additional attributes.

3.1.2.6. Configuration and Deployment

This section will be updated during implementation phase, because at the moment there aren’t enough details to provide inputs on it.

3.2. Orchestration

3.2.1. Overview

3.2.1.1. Objectives

In order to fulfil the business goals of a Virtual Organization, the services on offer must be combined and controlled through the use of structured workflows that reflect the business (process) models of the VO. The Application Support Services layer of the Akogrimo architecture includes support for such service orchestration, provided by the Business Process Enactor.

The mobile, dynamic grid world of Akogrimo introduces special concerns and problems that have to be catered for by the orchestration components of Akogrimo. Mobile clients and (particularly) mobile services may have changing requirements and capabilities, which must be taken into account in workflow management. In Akogrimo, we "decompose" large-scale business processes into multiple smaller workflow templates (in the first prototype, decomposition will be *a priori,* that is, external to the system), and so reduce the problem in two ways. Firstly, both the choice of template to use, and the instantiation of the template with "concrete" service instances, will be chosen based upon the current dynamic context (as well as drawing upon VO brokerage, negotiation and policy management). Secondly, context-dependent choices of action will be supported within the workflows themselves, though the intention is that each workflow instance should be of sufficiently short duration that context change is less of an issue.

3.2.1.2. Example Scenario

We have chosen to base the scenario on the Testbed scenario "Manage Virtual Emergency Environment" (in document D2.3.2, Testbed Validation Scenarios [13]). In this use case, an emergency call (from a patient or their heart monitor) is handled by an Emergency Centre Operator, who gathers further information, selects appropriate medical staff, and "initiates the emergency workflows". However, we have not adhered to the scenario rigorously, but have modified it to demonstrate the orchestration functionality in more detail, or to illustrate architectural arguments. In particular, we hope to demonstrate the behaviour of the Akogrimo orchestration components in the following situations:

- triggering of a workflow by some (possibly external) event: by a user request for a service, or by as an action from another workflow;
- choosing a workflow template: based on both static and dynamic (e.g. contextual) requirements;
- resolving a workflow template's abstract services into concrete services: using service discovery and negotiation;
- enactment and monitoring of the activities within a workflow;
- detecting and reacting to exceptions: SLA violations, failures to uphold policy;
- completing a workflow.

The scenario is initiated when a patient makes an emergency call, either by calling the Emergency Centre or when their heart monitor sends an alarm (which may be sent using the patient's phone, or by other unspecified means). This call is handled by an Operator in the Centre, who performs the next steps. We could consider the Operator as the *de facto* instigator of the emergency response workflow, and take this as the starting point for our scenario; but from a technology viewpoint it may be more interesting to consider the emergency call as the true instigator. That is, we consider that the emergency call itself triggers creation of an instance of the emergency response workflow. Thus the workflow is triggered by an event, and not (directly) initiated by a person. Initially, this workflow has a single “assigned” OpVO member, namely the patient himself. (We will consider the registration and logging-in of the patient as outside the scope of our scenario.)

Obtaining the patient’s attributes and context could be considered either as a step in the workflow or could be associated with logging in to the VO (though this would have to be automatically triggered by the (receipt of) the emergency call.) We note that the choice of workflow could depend on this information: for example, an emergency call from abroad would have to be handled in a different manner to a national call. (Or to be more prosaic, the choice may depend on whether the patient is at home, at work or on the move.)

The first action in this workflow (after the trigger event) is to be carried out by a user with the role EmergencyOperator, therefore the next step in processing is to choose an individual of this role. In other words, the emergency call must be assigned to an operator (and it would be sensible to choose an operator who is not busy). So we need to ask the OpVO Broker to assign a (non-busy) operator to the OpVO for this workflow. There may be further constraints; for example, that the operator should be able to speak the patient’s language. This means that information about the patient must be available to the Workflow Manager and/or the VO Manager before the choice of operator can be made. This illustrates that we may not know all role assignments, and hence not the full OpVO membership, until we are part-way through enactment of a workflow. Static constraints can be satisfied in advance and can be used to create a list of eligible participants for this role. Other constraints, dynamic ones, would need to be satisfied as close to the required time as possible, otherwise inappropriate choices might be made. The impact on an OpVO depends on when and how OpVOs are created; this is discussed in 3.1.1.3.10. If a new OpVO is set up for this case, it may be preferable for all participants satisfying the static constraints to be eligible for the role - or a subset such as the "top 3". If a new activity is set up within an existing OpVO, all potential role-fulfillers are already in the OpVO, except possibly for new recruits, which we do not consider here.

Next, the Operator establishes “two-way voice contact” with the patient (presumably only if the patient is conscious); presumably once established, this communication persists in parallel through most of remaining steps of the case, and the Operator cannot be considered available for further calls until the dialog is closed (that is, the workflow cannot be considered completed until the dialog has ended).

In the meantime (in other words, in parallel with the above task), data relevant to the patient’s condition must be retrieved. In practice, it would make sense (assuming sufficient capacity) to have such information included in the patient information retrieved as part of the initial response to the emergency call; for demonstration purposes here, we could view this as another step in the workflow wherein the data is collected; as this may be from a variety of sources, the step may itself be a complex sub-workflow.

Suitable parties for the emergency response must be found; in other words, available VO members must be sought and added to the OpVO for this workflow. Here, “suitable” includes both static and dynamic criteria, such as medical expertise specialised to the patient’s condition, location and (of course) availability. Again, in practice it would make sense to pre-determine those VO members who could satisfy the static constraints (expertise, etc.), and this may be part of the formation of the “Virtual Emergency Environment” in the original testbed use case; but the dynamic constraints (particularly location) are best resolved "on the spot". The choice may depend on information obtained in the parallel discussion between the Operator and the patient.

The final step in the use case is for the Operator to "start respective services and workflows". Though this could be considered to be the point where orchestration *really* starts, we could also consider it to be the end of our workflow, which finishes by having the Operator choose one or more new workflows to start. The Workflow Manager must ensure that the appropriate data is "inherited" by the new workflows; and remember that there may be parallel tasks (such as the voice communication) that must be completed before the workflow can be considered complete.

The above has not considered possible exceptions in the workflow. One example could be if the Operator has to stop dealing with this case part-way through. Then a new Operator must be chosen, and sufficient context transferred to her (along with membership of the OpVO) to enable her to continue the task. Might the handover process itself be described by a workflow?

3.2.1.3. Functional Capabilities

The Orchestration components of the Application Services Layer are intended to provide the following functions (naturally, through interaction with other components):

Workflow Template Management

- Selection of appropriate workflow templates based on semantic attributes
- Storage of workflow templates representing advertised services

OpVO Management

- Incremental formation of an OpVO upon workflow template enactment
- Destruction of an OpVO upon workflow template termination

Workflow Template Evaluation

- Resolution of abstract services into concrete services, through service discovery and selection

Workflow Enactment

- Execution and steering of a workflow control-flow
- Management and steering of a workflow data-flow
- Coordination and synchronization of services

Workflow Enactment Monitoring

- Subscription of services for monitoring
- Event polling for exception management

3.2.1.4. Position with respect to the Akogrimo infrastructure

This section provides a broad view on the position of the orchestration topic with respect to the whole Akogrimo architecture. Sequence diagrams and interaction tables will be presented to depict briefly the basic concepts behind the orchestration components making up the orchestration problem.

3.2.1.4.1. *Interactions with Akogrimo infrastructure components*

Following the example scenario of section 3.2.1.2, the Figure 20 is a sequence diagram depicting the initial scenario's workflow enactment process with respect to some Akogrimo components. Recall that the BP Enactor depicted is the WP4.4 component in charge of managing the whole orchestration process.

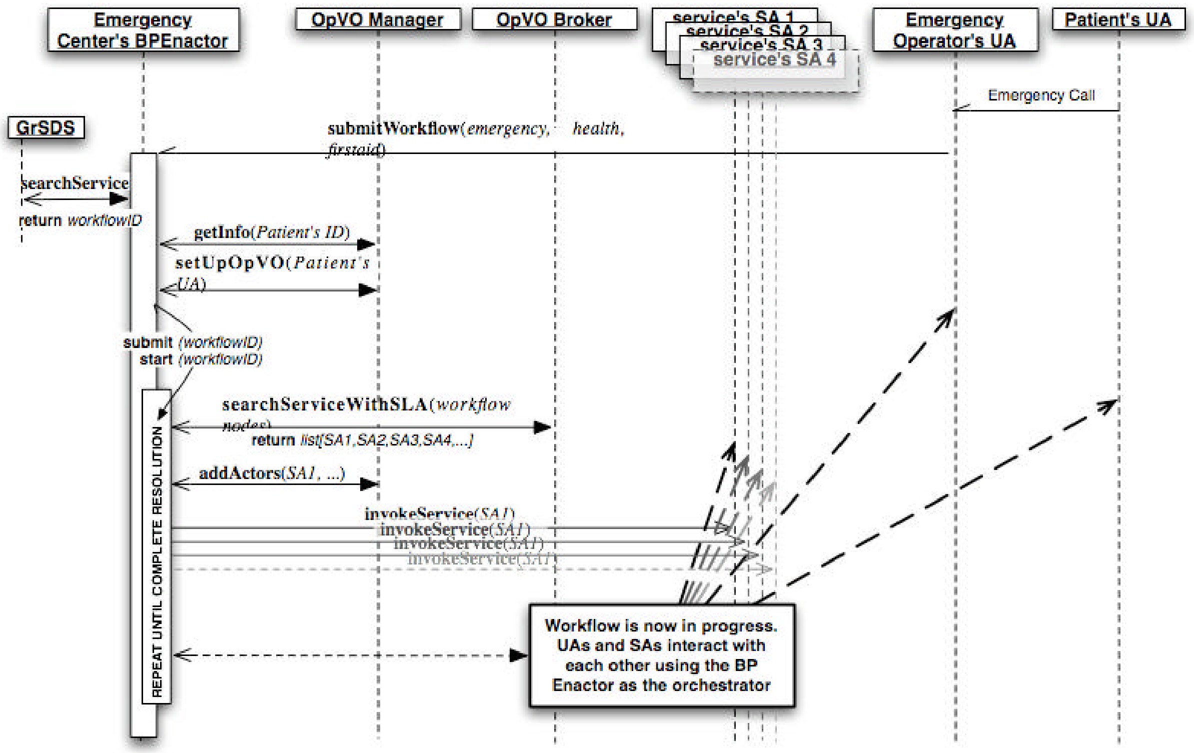


Figure 20: A high-level sequence diagram depicting the workflow enactment of the example scenario with respect to the other Akogrimo components

It is worth noticing that the functions depicted in this sequence diagram are directly taken from sections 3.2.2.4 and 3.2.2.5. Moreover, note that these are not the only interactions arising in a workflow enactment process. This is only the initial high-level sequence of steps of the enactment process. For a more detailed view of interactions and sequence diagrams of enactment see sections 3.2.2.2 and 3.2.2.3. Meanwhile, the following table lists the broad interactions between orchestration components and the Application Support Services layer. The interfaces are detailed further in section 3.2.2.

Identified interaction	**Rationale for interaction**	**Issues to be addressed**
OpVO Manager	OpVO creation, management and destruction	Are we foreseeing the possibility for a user to join an already instanced OpVO?
OpVO Broker	Service discovery and selection	

Table 6 Summary interactions between orchestration components and internal WP4.4 components

The following table lists the broad interactions between the Orchestration components and other layers of the Akogrimo infrastructure. The interfaces are detailed further in section 3.2.2.

Identified interaction	**Rationale for interaction**	**Issues to be addressed**
Grid Service Discovery Server	Find services matching user request	What semantic annotation should be used?
Context Manager	Subscribe/unsubscribe for particular context changes; poll for event changes	Might event push be possible?
SLA Enforcement	Subscribe/unsubscribe for particular SLA state changes; poll for event changes	Might event push be possible?
User Agent	Workflow initiation and control; return of results	
Service Agent	Invoke service methods	

Table 7 Summary interactions between orchestration components and outsider WP4.4 components

3.2.2. Architecture description

3.2.2.1. Architecture overview

The service orchestration problem is the problem of modelling, expressing and successively enacting a set of services in order to make them cooperate in a coordinated fashion. In our case, the modelling is done by means of a workflow-based representation of the intended interactions and synchronizations between services. Such workflows represent specific business processes and will be handled by the BP Enactor.

Services can be referenced in both abstract and concrete ways, the former meaning only service's semantic information are used along with the discovery process and the latter meaning services can be directly referenced, thus without passing through the discovery process. Ideally, every business process published in Akogrimo will be stored, as a workflow template representing a specification of the interaction between abstract services, in a Workflow Registry.

Architecturally, the orchestration process is started and managed by the Workflow Manager component of the BP Enactor. During the orchestration process, workflow templates will be "resolved" to concrete workflows by discovering each composing abstract service. From an architectural point of view it is important to note that this "resolution" step will be accomplished in a lazy way, thus resolving and enacting pieces of workflow whenever they become "available". Moreover, the instantiation of workflow parts will entail the necessity for the Workflow Manager to interact with the OpVO Manager for creating and maintaining the OpVO related to the submitted business process. The Workflow Manager will successively submit such resolved workflow parts for enactment.

An Enactment Engine, capable of creating an instance of a workflow and successively steering and managing the run-time execution arising from the interaction and synchronization steps between services, then carries out the enactment process. The engine is thus responsible of invoking service operations, moving I/O data between services and fulfilling the temporal and control dependencies described by the submitted workflow piece.

During this enactment process, the Workflow Manager will be responsible of adapting the workflow to particular events. Such monitoring capability is necessary for managing context and SLA changes in user and service state respectively. Note also that this monitoring process, carried on by the Monitoring Daemon sub-component of the BP Enactor, entails a tight communication with external components such as the SLA Enforcement and the Context Manager.

3.2.2.1.1. *Functional requirements*

From a purely functional point of view the BP Enactor is responsible of intercepting user requests of services execution, matching them to some published and available services and enacting them while providing the necessary feedback to the user.

In a more formal way, this means the BP Enactor is required to provide the following functional characteristics:

- **User Interaction:** the first and foremost functional requirement is to provide ways for users to communicate their requests of service execution. The user will be able to select specific services for enactment or to specify a set of annotations describing, semantically, the type of service requested.
- **Service Matching:** the BP Enactor does not directly provide service matching. However, it must filter the information gathered from the user request and forward the important ones to the service in charge of matchmaking such information to appropriate services
- **Workflow Selection:** the returned service matching the user requirements must be associated, when necessary, to appropriate workflow templates, stored in workflow registries. Such templates will describe the interactions between abstract services, i.e. services without any binding to real web-services. Abstract services will be described by the means of semantic annotations that will describe their functional and non-functional capabilities and requirements. Moreover, because several workflows can correspond to one published service, selection of the best workflow should also be addressed by associating usage-related statistical information to stored workflows
- **Workflow Enactment & Monitoring:** the BP Enactor is responsible of providing the final enactment of the whole workflow. Such enactment entails the need for functional characteristics that make the Enactor able to execute and monitor the workflow. Moreover, the enactor should provide mechanisms for managing and steering such execution.

3.2.2.1.2. *Non-functional requirements*

The BP Enactor has the task of managing, steering and monitoring the execution of workflows. Such tasks need to be done in a flexible and adaptive manner and it is necessary to have these characteristics assured while creating the BP Enactor architecture. The following is thus a list of the principal non-functional characteristics the BP Enactor is supposed to own:

- **Service oriented sub-architecture:** all sub-components making up the BP Enactor should be exposed to the AkoGrimo environment as normal services, thus entailing the use of standards such as WSDL, SOAP and HTTP as protocols for interactions among internal and external components

- **Adaptive Enactment:** flexible enactment of a workflow requires the possibility of making quasi real-time changes to its control flow. Such adaptability can be achieved by applying enactment techniques that enables rollback, run-time substitutions and suspension of workflow executions
- **Lazy Evaluation:** evaluation of a workflow's nodes, i.e. resolution of abstract services into concrete ones, in a lazy way is indispensable for obtaining a flexible enactment process. Lazy evaluation gives the Enactor the ability to evaluate and successively execute services of a workflow with a high degree of control over how to deal with failures, with flow branches and with monitoring tasks
- **Real-time Monitoring:** real-time monitoring is obviously another fundamental characteristic required by the Enactor. Adaptive enactment of a workflow can only be done if real-time monitoring characteristics can be provided for the Enactor to take run-time decisions. Such monitoring needs to be effective not only on services execution but also on most of the workflow execution parameters that could influence the enactment process.

3.2.2.2. Use case view

The use case view proposed in the two following sections has been extracted from the previously described example scenario. Its purpose is to depict more clearly what are the interactions between the actors taking part into the whole process of workflow enactment. Note that to accomplish such a goal we are using entities, precisely the sub-components of the BP Enactor briefly introduced in section 3.2.2.1, which are clearly described only in section 3.2.2.3.

Moreover, the following is a figure depicting a sample workflow that will be used as a reference to describe the use case extracted from the example scenario. Note that, to be coherent, this workflow too has been modelled on the example scenario.

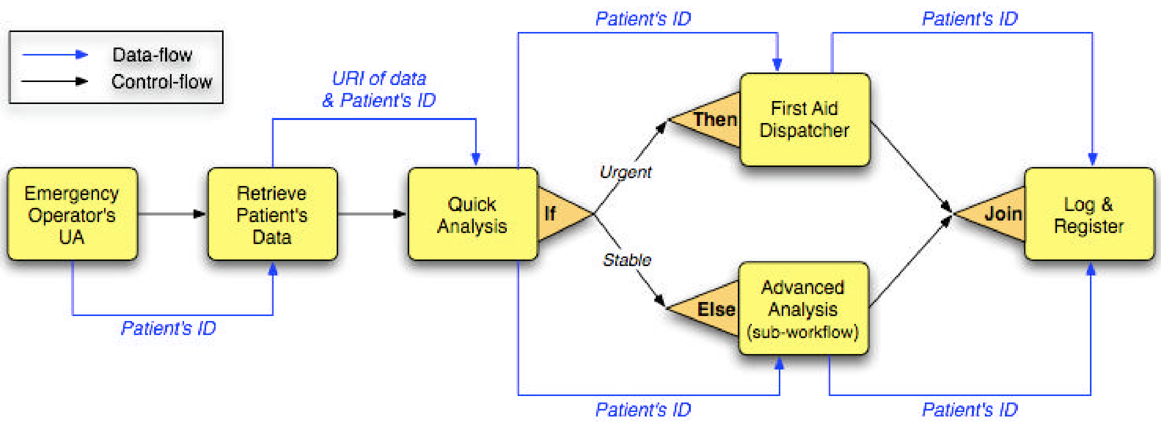


Figure 21: A sample workflow depicting a possible business process for the example scenario

As depicted in Figure 21, the workflow sample describes the entities, the control-flows and the data-flows involved in a possible e-health business process.

3.2.2.2.1. Actors

By considering the example scenario, the sub-components of the BP Enactor, the previously described interactions between Akogrimo entities and, finally, by recalling Figure 21 we can list the followings as the potential user and service actors of a workflow enactment process:

- **The patient's UA:** This is the entity inside Akogrimo and inside the OpVO that works on behalf of the user.
- **The emergency operator's UA:** This is the entity that actually triggered the workflow request to the BP Enactor.
- **The involved SAs:** These are all the services that have become available and have been selected for fulfilling the workflow enactment process. These are the actual concrete services the BP Enactor decides to use during the emergency workflow's evaluation.

Moreover, the followings are the identified actors of a workflow enactment from a BP Enactor's components point of view:

- **The Workflow Manager:** This component is the actual central unit for processing all the necessary decision on the workflow enactment process. The Workflow Manager is the decision-making component of the BP Enactor.
- **The Enactment Engine:** This component is directly responsible of the service methods invocation, of managing the I/O data, and of submitting all the necessary requests through the classical communication protocols, namely HTTP and SOAP.
- **The Workflow Registry:** This component is just a repository of the description, or implementation, files of the workflows published in the Akogrimo environment. Some sort of database manager will serve such registry.
- **The Monitoring Daemon:** This component is in charge of subscribing and polling events of entities involved in the workflow enactment. Such monitoring will be necessary for checking run-time SLA failures and for updating user's context information changes, due to the user's possible mobility for instance.

Finally, the followings are the actors of an enactment process from the point of view of the Akogrimo's Application Support and Infrastructure services:

- **The OpVO Manager:** This entity will manage all the aspects of the OpVO life-cycle. It is its duty to create, add, remove and destroy OpVO related to single workflow enactment processes. Addition and removal of entities from an OpVO must be supported even at run-time, so to enable a high level of adaptivity. The OpVO Manager will thus be also responsible of finding the good security tokens to create or pass enabling all participants in an OpVO to interact with each other.
- **The OpVO Broker:** This is the entity in charge of finding appropriate services for the workflow. It will ask the discovery server the list of concrete potential services and successively will interact with the SLA High Level services for agreeing on SLA contract. The selected service will then be announced to the Workflow Manager.
- **The GrSDS:** As stated just above, this is the entity involved in the service discovery process. Such process will be carried-on by matching semantic information of the requested functionalities with semantic information of the capabilities of published services, found in service registries.
- **The Context Manager:** This entity is one of the two entities involved in the monitoring problem. The Context Manager will interact with the Monitoring Daemon previously

introduced for constantly checking triggered events stating some context changes in the user status.

- **The SLA Enforcement:** This is the second entity involved in the monitoring. Similarly to the Context Manager, the SLA Enforcement will interact with the Monitoring Daemon for having run-time information about SLA contract failures and QoS-level status of subscribed services.

3.2.2.2.2. Use Case

Figure 22 shows a detailed view of how all the actors introduced in the above section interact with each other. This sequence diagram shows the steps done by the BP Enactor's components for the enactment of part of the workflow sample of Figure 21. Note that this sequence diagram completes the previously shown diagram of Figure 20, which depicted the initial high-level steps of the interactions between parts of all the involved actors.

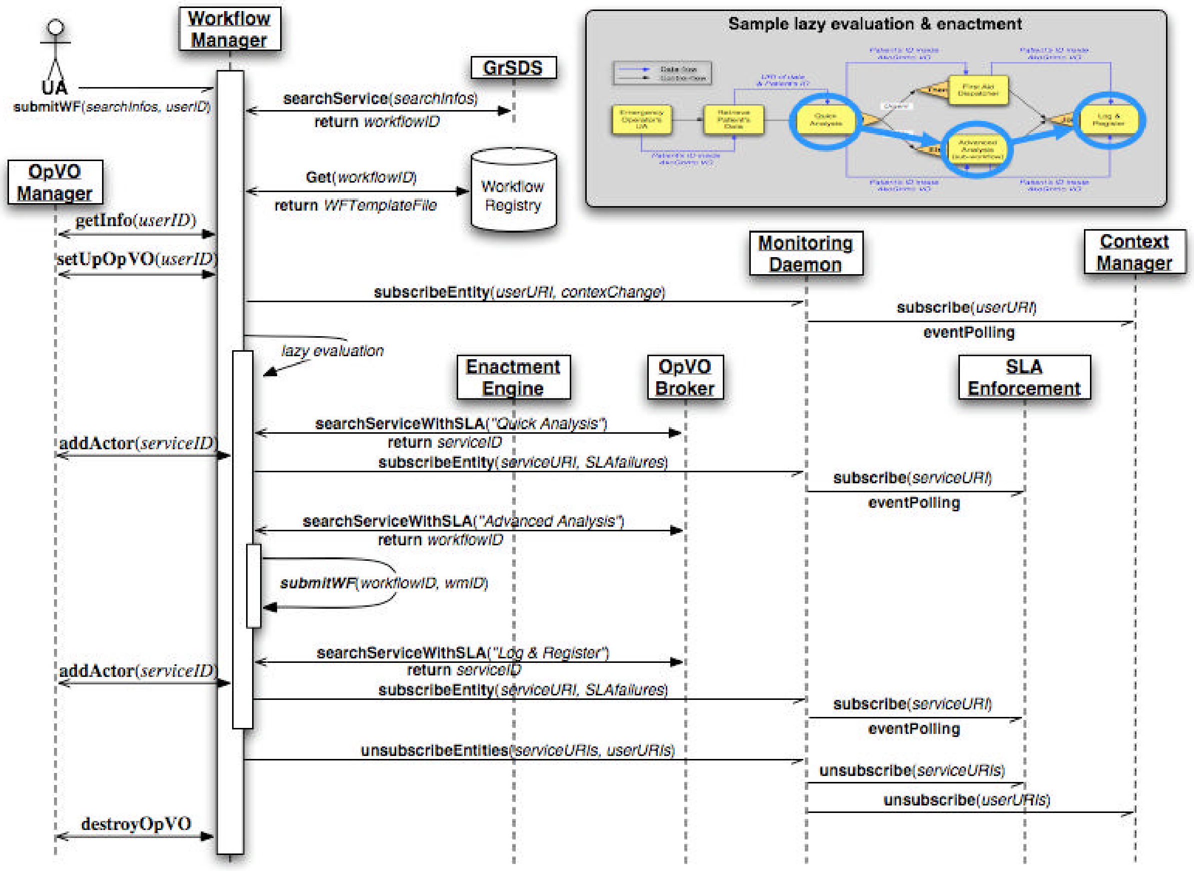


Figure 22: A sequence diagram of the workflow enactment process

For the sake of simplicity, but without loss of details, the sequence diagram only depicts the enactment process starting from the *Quick Analysis* task, as depicted by the blue circles and arrows of Figure 22.

It is worth noting how the BP Enactor copes with task in the workflow, namely the *Advanced Analysis* one. When asking for this service to the OpVO Broker, the Workflow Manager is answered with a *workflowID* meaning that the service is a sub-workflow and thus needs to be searched and enacted as a nested workflow. That is the reason for the following self-submitWF done by the Workflow Manager. By "looping back" to itself this request the Workflow Manager will start a nested workflow enactment process for that *Advanced Analysis* workflow with a sequence similar to the one depicted here.

3.2.2.3. Conceptual architecture description

As stated in section 3.2.2.1, the orchestration process is handled by the BP Enactor, which is composed of several subcomponents. These components will be necessary for storing and selecting workflows, managing context and SLA run-time changes, and maintaining instanced OpVO.

Figure below shows an overview of the internal architecture of the BP Enactor. The subcomponents are depicted in red/dark colour. Dashed arrows represent the interactions with internal components. For completeness, the figure also depicts external components, in blue/bright colour, representing the other entities with which some sub-components of the BP Enactor need to interact.

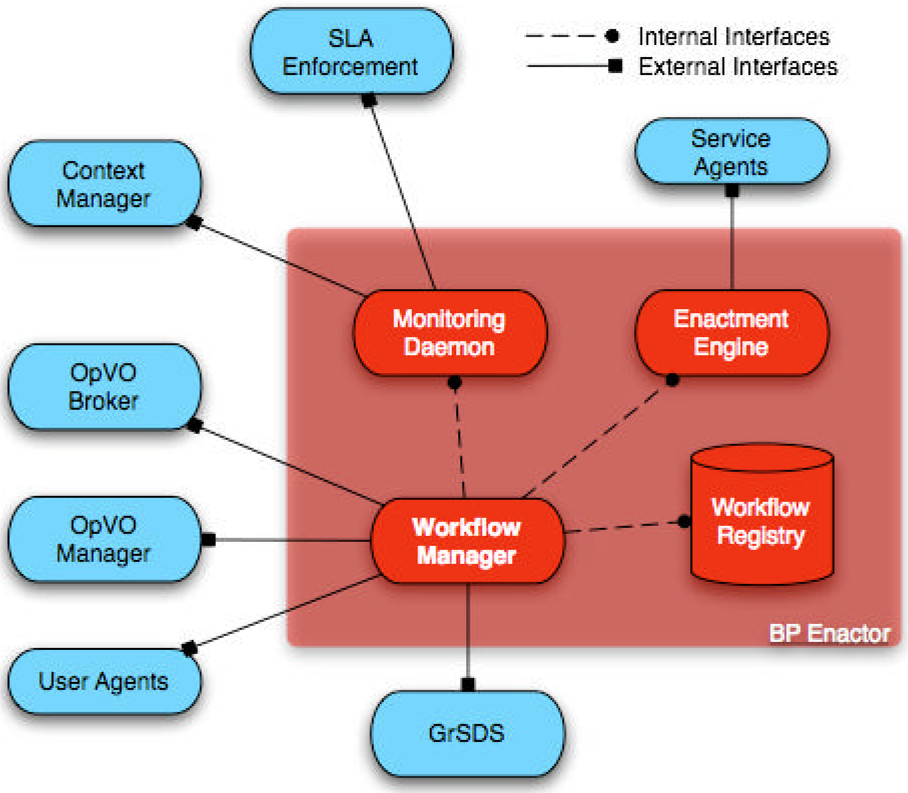


Figure 23: Internal and external interfaces between BP Enactor components and external AkoGrimo components

3.2.2.3.1. Workflow Manager

This component is at the core of the BP Enactor. Its purpose is to manage and steer the whole workflow enactment process, starting from the selection of the appropriate workflow and ending with the results transfer to the user who requested the workflow instantiation. The Workflow Manager is thus responsible of maintaining a repository of workflow, of selecting the most appropriate one based on the users semantic requests, and of starting, monitoring and adapting the workflow execution based on the real-time evolution of services and Akogrimo resources.

The Workflow Manager can be seen as the central processing unit of the whole workflow enactment process. It is in charge of receiving and processing user-side requests of service execution, of managing SLA and user context event notification and of submitting and monitoring the workflow enactment process.

The Workflow Manager interfaces externally with the Service Discovery Server to find services that match the user-submitted criteria. Successively, when the GrSDS answers with semantic keywords corresponding to a complex service, the Workflow Manager will check the workflow registry for retrieving the appropriate workflow implementation file. For an overview of invocation methods required by the Workflow Manager see section 3.2.2.5.1.

The Workflow Manager interfaces externally with the OpVO Broker for gathering the necessary information concerning the set of concrete services to be used for the workflow enactment process. The Workflow Manager will thus ask the OpVO Broker to discover and select appropriate services based on the semantic functional and non-functional requirements specified in the workflow implementation file. Note that it is the Workflow Manager responsibility to decide whether the whole workflow has to be discovered before execution or whether it can be enacted in a more lazy fashion. For an overview of the invocation methods required by the Workflow Manager see section 3.2.2.5.1.

The Workflow Manager interfaces externally with the OpVO Manager for gathering information on both users and services authenticated into the Akogrimo environment. Moreover, this interface is needed also for enabling the Workflow Manager to ask the creation, management and destruction of OpVOs. For an overview of the invocation methods required by the Workflow Manager see section 3.2.2.5.1.

The Workflow Manager interfaces externally with the User Agent, which invoked the workflow instantiation. Service methods should be provided, by the latter for enabling the former to notify it of requested interactions, ending of the workflow enactment, availability of results, location and persistency of I/O data, etc… For a more accurate description of the invocation methods provided by the Workflow Manager to the user agents see section 3.2.2.5.1.

The Workflow Manager interfaces internally with the Workflow Registry for retrieving and storing workflow implementation files. Such files use a particular language, BPEL for instance, for describing the abstract services' orchestration process to be enacted. File transfer between the two entities will be accomplished by the means of secure and reliable protocols such as SFTP. For a detailed description of the Workflow Registry see section 3.2.2.3.2.

The Workflow Manager interfaces internally with the Enactment Engine for submitting, managing, steering and inspecting a workflow's execution. It will thus be the Workflow Manager duty to check the execution state of a workflow, to cancel or stop workflow execution and to inspect internal state of a workflow's enactment process. For a detailed description of the Enactment Engine see section 3.2.2.3.3.

The Workflow Manager interfaces internally with the Monitoring Daemon for retrieving the necessary information related to on-line context changes or SLA failures of services involved in the orchestration process. Thus, the Workflow Manager will constantly interact with the monitoring daemon to retrieve events that entail specific workflow management procedures. For a detailed description of the Monitoring Daemon see section 3.2.2.3.4.

3.2.2.3.2. Workflow Registry

The workflow Registry is the component in charge of storing implementation files of published workflows. Such files will be stored in some relational database, MySQL for instance, along with annotations necessary for semantically identifying workflows. A unique key will identify each workflow and its semantic annotations. Semantic annotations will be stored either as a secondary database table or as files written in semantic languages such as OWL-S.

Note that such annotations can also be used to store "usage" information useful to the Workflow Manager for taking decisions on which workflow to use for satisfying a specific user-request. Thus, the workflow registry will be update by the Workflow Manager each time a workflow has been enacted, so to trace statistical usage parameters such as execution errors, SLA failures, execution successes, requests, etc…

Note that this workflow registry is only used by the Workflow Manager, thus is not considered a service belonging to the Akogrimo basic VO. Thus, the workflow registry interfaces, internally, solely with Workflow Manager and it is the Workflow Manager duty to query, remove or store workflows in this registry. For a detailed description of the APIs provided by the Workflow Registry to the Workflow Manager see section 3.2.2.4.2.

3.2.2.3.3. Enactment Engine

The Enactment Engine is the component in charge of invoking the necessary operations on the discovered and selected services, to collect data outputs when necessary and to steer the workflow enactment based upon the control flow specifications given in the workflow implementation file.

The Enactment Engine interfaces externally with Service agents for invoking the necessary service methods that progress the workflow enactment. For an overview of the invocation methods required by the Enactment Engine see section 3.2.2.5.3.

The Enactment Engine interfaces internally with the Workflow Manager. The latter, after resolving a workflow's abstract services into concrete ones will submit the concrete workflow to the engine for enactment. During this enactment process, the Engine can interact with the Workflow Manager to communicate it how is the submitted concrete workflow execution evolving. Service failures, branch selections, loop's iteration number and intermediate variable values are all tracked by the enactment engine and communicated to the Workflow Manager, if necessary. For a detailed description of the invocation methods provided by the Enactment Engine see section 3.2.2.4.3.

3.2.2.3.4. Monitoring Daemon

This is the component in charge of monitoring and tracing all events related either to context changes of an entity involved in the workflow enactment process, or to the SLA failures deriving from the inability of a service of fulfilling the initially agreed Service Level Contract.

The Monitoring Daemon interfaces externally with the Context Manager. Such interaction is necessary for this component, to know what are the changes in context of entities involved in the workflow enactment process. Thus, the Monitoring Daemon needs to subscribe to the context Manager all the necessary entities involved in the enactment process. Upon receiving a context change event, the Daemon will store it in a priority queue for retrieval by the Workflow Manager.

Thus, the Monitoring Daemon duty is to trace such events and manage the queue in which they are stored so to always be able to inform the Workflow Manager, when it requests for events, of the context changes that occurred during the enactment of those entities. Note that, for the Monitoring Daemon to be effective, the priority and management policies related to the event queue are fundamental. For an overview of invocation methods required by the Monitoring Daemon see section 3.2.2.5.4.

The Monitoring Daemon interfaces externally with SLA Enforcement. For this interaction, the same concepts and ideas stated for the Context Manager interaction apply. For an overview of invocation methods required by the Monitoring Daemon see section 3.2.2.5.4.

The monitoring Daemon interfaces internally with Workflow Manager. The former needs to expose methods that can enable the latter to manage the queues of entities to be monitored for context changes or SLA failures. Moreover, the Workflow Manager obviously needs methods to query for the state of such queues and maybe to select the priority and management policies to be used. For a detailed description of the invocation methods provided by the Monitoring Daemon see section 3.2.2.4.4.

3.2.2.4. Logical architecture description

3.2.2.4.1. Workflow Manager

The Workflow Manager interfaces with the user agent. To it, the Manager should provide service methods for enabling the users to access workflow basic enactment functionalities. The following is thus the list of service methods provided by the Workflow Manager to user agents:

- workflowID **submitWorkflow**(annotations[])

  It returns the identifier associated with the workflow selected by the Workflow Manager upon successful selection from the registry.

  The parameter is the user-side annotations used to semantically identify the service.

- resultURI **getResults**(*workflowID*)

  It returns an URI to where results have been stored. Note that in case of workflow enactment failure the function should return some error value (null for instance).

  The parameter is the workflow identifier returned by the Workflow Manager with the **submitWorkflow** call.

- **cancelWorkflow**(*workflowID*)

  This function enables the user to cancel execution of a workflow.

  The parameter is the workflow identifier returned by the Workflow Manager with the **submitWorkflow** call.

- **stopWorkflow**(*workflowID*)

  This function enables the user to temporarily suspend the execution an already submitted workflow for later resume. Such function requires the BP Enactor to provide mechanisms for suspending a workflow execution, thus mechanisms for maintaining all necessary information related to workflow status and services I/O.

  The parameter is the workflow identifier returned by the Workflow Manager with the **submitWorkflow** call.

- **startWorkflow**(*workflowID*)

  This function enables the user to restart an already submitted workflow or to resume it after a **stopWorkflow** call.

  The parameter is the workflow identifier returned by the Workflow Manager with the **submitWorkflow** call.

- wf_infos **getStatus**(*workflowID*)

  It returns the workflow enactment process information, along with the actual service being executed and the variable values of the control and data flow.

  The parameter is the workflow primary key returned by the Workflow Manager with the **submitWorkflow** call.

3.2.2.4.2. Workflow Registry

The Workflow Registry interfaces, internally, solely with Workflow Manager. It is the Workflow Manager duty to query, remove or store workflows in the registry. Thus, basic APIs to be provided by the Workflow Registry for making the Manager interact with it are the following:

- workflowID **Store**(wfTemplateFile, annotations[])

  It returns a workflow identifier assigned by Database upon success.

  The parameter is the workflow implementation file and the associated semantic annotation array.

- WFTemplateFile **Get**(*workflowID*)

  It returns the workflow implementation file upon success.

  The parameter is the workflow identifier eventually returned by the **searchService** request sent to the GrSDS.

- **Update**(workflowID, annotations[], values)

  It returns success or failure.

  The parameters are the workflow identifier, the semantic annotations to be updated and their new values

- **Remove**(workflowID)

  It returns success or failure. This function removes the specified workflow from the database.

  The parameter is the workflow identifier.

3.2.2.4.3. Enactment Engine

The enactment engine interfaces internally with the Workflow Manager. It will be the Workflow Manager duty to submit and query state of workflows to the enactment engine. Basic service methods to be provided by the Enactment Engine are thus the following:

- workflowRef **Submit**(*wfTemplateFile*)

  It returns an internal workflow reference if it has been correctly parsed and accepted by the engine.

The parameter is a workflow (or part of it) described through a BPEL, OWL-S or data structure specification.

- **Start**(workflowRef)

  This function starts the execution of a workflow.

  The parameter is the engine's workflow internal reference returned by the **Submit** function.

- **Cancel**(workflowRef)

  This function cancels the execution of the workflow and all related intermediate data.

  The parameter is the engine's workflow internal reference returned by the **Submit** function.

- **Stop**(workflowRef)

  This function suspends the workflow execution by saving all intermediate control flow data and services I/O (note that suspension is made upon successful completion of the service executing at the moment of the suspension request).

  The parameter is the engine's workflow internal reference returned by the **Submit** function. Information related to the state of the workflow (control flow variables, services I/O values and service reached in the workflow) can be retrieved with the **InspectVar** and **InspectFlow** functions.

- wf_state **GetWFState**(*workflowRef*)

  It returns the status of a workflow. Possible suggested states are NO_WORKFLOW, SUBMITTED, STOPPED, EXECUTING, SUSPENDED, FLOW_ERROR, SUCCESS.

  The parameter is the engine's workflow internal reference returned by the **Submit** function.

- wf_variable **InspectVar**(*worfklowRef*)

  It returns the list of a workflow's intermediate variables and services I/O values.

  The parameter is the engine's workflow internal reference returned by the **Submit** function.

- wf_construct **InspectFlow**(*workflowRef*)

  It returns the list of workflow's control flow constructs along with their state and value. Moreover, this function returns the state, DONE/FAILED/RUNNING, of each service involved in the workflow.

  The parameter is the engine's workflow internal reference returned by the **Submit** function.

These are the basic, static, functions that makes the Workflow Manager able to control and query the execution of a submitted workflow. With such functions the Workflow Manager is able to "monitor" the execution of a workflow with a simple looping procedure in which execution information are asked to the engine. Below an example of such an "event" loop is shown:

```
while( GetWFState(workflowRef) == EXECUTING ){
        wf_variable variables = InspectVar(worfklowRef);
        wf_construct flowInfos = InspectFlow(workflowRef);
        /* Do what is needed to do */
}
```

This simple code is an example of how the Workflow Manager could monitor the workflow enactment done by the engine. Moreover, note that this loop can be nested into a switch statement providing cases for all the wf_state values returned by the **GetWFState** function.

3.2.2.4.4. *Monitoring Daemon*

The Monitoring Daemon interfaces internally with the Workflow Manager. It will be the Workflow Manager duty to update, add, remove and query the Monitoring Daemon about pending events related to context changes and/or SLA failures. Basic service methods to be provided by the Monitoring Daemon are thus the following:

- **subscribeEntity**(entityURI, monitorType)

  It returns a success or failure.

  The parameters are the entity's URI to be monitored and the type of monitoring to be done, i.e. user context or SLA.

- unsubscribeEntity(*entityURI*)

  It returns a success or failure.

  The parameter is the entity's URI.

- event **pollEvent**(entityURI, monitorType)

  It returns the first event found in the priority queue related to a specific entity and monitoring type.

  The parameters are the entity's URI to be monitored and the type of monitoring to be done, i.e. user context or SLA.

- **getQueueLength**(entityURI, monitorType)

  It returns the length of the queue of pending events.

  The parameters are the entity's URI to be monitored and the type of monitoring to be done, i.e. user context or SLA.

Note that, even though the service methods provided by the Monitoring Daemon are few and simple, it is of greatest importance that it manages appropriately the necessary queues of events related to the entities of a workflow being enacted. Such queues represent the only way for the Workflow Manager to know, thus to adapt the execution, how the state of resources and services in us has changed form a context and SLA point of view.

3.2.2.5. *Logical External Interfaces*

3.2.2.5.1. *Workflow Manager*

The Workflow Manager, core component of the BP Enactor, interfaces externally with the Service Discovery Server, with the OpVO Broker, the OpVO Manager and with User Agents. Such interactions need service methods specification, but such specification should be provided and described by the two latter components whereas the Workflow Manager will use such methods to access functionalities of these AkoGrimo services.

However, for completeness, we provide hereunder a list of functionalities that should be exposed as service methods, the Workflow Manager require from these two external components:

- **GrSDS – searchService:** some APIs should be provided for the Workflow Manager to be able to ask for the discovery of services based on semantic annotations
- **OpVOBroker – searchServiceWithSLA:** some APIs should be provided for the Workflow Manager to be able to ask for the discovery of concrete services fulfilling specific SLA requirements
- **OpVOManager – getInfo:** some APIs should be provided for the Workflow Manager to be able to request information about users and services that should get involved in a workflow enactment. Such information will probably be retrieved from Akogrimo's participant registries by the means of the VOManager.
- **OpVOManager – setUpOpVO:** some APIs should be provided for the Workflow Manager to be able to request the creation, and thus the dissemination among initial participants, of appropriate credentials (maybe tokens) for setting up an OpVO within which the workflow enactment process can be started
- **OpVOManager – addActor/removeActor:** some APIs should be provided for the Workflow Manager to be able to add new actors, at run-time, to the OpVO containing the actual workflow enactment process and its participants
- **OpVOManager – destroyOpVO:** some APIs should be provided for the Workflow Manager to be able to request the destruction of an OpVO. Usually, such event happens after workflow enactment completion and can thus be triggered only by the Workflow Manager (being the only entity knowing if the workflow enactment has ended)

3.2.2.5.2. Workflow Registry

As already stated, the Workflow Registry doesn't interact with services external to the BP Enactor. The workflow registry just provides APIs to the Workflow Manager subcomponent.

3.2.2.5.3. Enactment Engine

The Enactment Engine interfaces externally with the agents of the services involved in the workflow enactment process. The methods required by the Engine to interface with such agents, thus with their respective services, are stored and specified in the workflow. Thus, the Engine requires no specific method. The latter will find all the necessary information about method invocation syntax and I/O data type in the workflow specification submitted with to it with the **Submit** method.

3.2.2.5.4. Monitoring Daemon

This component interfaces externally with two AkoGrimo core services: the context manager and the SLA High Level. With both of them, required functionalities they should expose as service methods are the following:

- **Subscription/Unsubscription:** methods should be provided by the two AkoGrimo core services for subscription and unsubscription of entities to the related events. Ability to specify the type of event, along with the entity to be monitored, should be given
- **Event Polling:** a method for gathering occurred events should be provided. Ability to specify the type of event to poll and the type of entity for which we are interested should be

provided. Note however that it is foreseeable to have, instead of a polling mechanism, a simpler event management mechanism based on event pushing.

3.2.2.6. Involved technologies

The orchestration problem described conceptually and architecturally in this chapter needs technologies that enable its real and effective implementation. Such technologies should address the two most important aspect of orchestration:

1. How is the orchestration between abstract services modelled and described
2. How is this model interpreted and executed

The first point is addressed by selecting a language that can describe the control and data flow between different entities, possibly specified in both an abstract (i.e. without referring to some real implemented piece of software of service) and a concrete (i.e. binding the entity specification to a precise service or resource) way. From such point of view, our attention has restricted its choice between two known languages, namely BPEL and OWL-S:

- The former is a more “business process” centric language. Its specification enables the description of highly complex workflows, within which parties exchange information by following the control and data flows descriptions. BPEL is a fully working orchestration language that supports abstract and concrete specification of services and that provides a very wide range of constructs for flow control, data binding and variable definition.
- The latter in, instead, a more “semantic” oriented language. Even though OWL-S supports basic control and data flow constructs that virtually provide the ability of producing any kind of complex workflow[1], its focus is on the semantic specification of entities. Ideally, OWL-S enables the description of services’ functional and non-functional characteristics at an arbitrary level of abstraction, thus making it a very flexible and adaptable language for future AkoGrimo scenarios in which semantic matching and entailment can be requested.

The second aspect of orchestration is addressed by defining and, if necessary, implementing the engine in charge of effectively parsing and executing the workflow. Such an engine needs to maintain adequate data-structures and information about a workflow so that execution can be steered, monitored and managed in a flexible way. Moreover, such an engine should also provide all the necessary APIs, languages, protocols and standards for making it possible to invoke service’s methods, passing them inputs and retrieving, when necessary, outputs.

The three following sections describe the possible combinations of these technologies to address the orchestration problem. It is not our intention to suggest a solution as the best one. Instead, we want to clearly describe what can be the advantages and disadvantages, within the AkoGrimo environment and scenarios, of all the possible solutions arising from the combination of the aforementioned technologies.

[1] We can refer to this characteristic by stating that OWL-S is a Turing-machine equivalent language.

3.2.2.6.1. *BPEL Centric Solution*

This solution is based solely on the BPEL language and its capabilities of describing both orchestration-oriented flows and abstract service capabilities and requirements. As already stated, such solution doesn't give AkoGrimo the ability to use semantic description of services, however it can still make use of keywords for service description and abstract workflow definitions.

The use of BPEL as the only language for both enactment and description provides two advantages: firstly, a single language makes it possible for all components of the BP Enactor along with external components to communicate without "translation" and/or "interpretation". This gives the whole systems a more robust, updatable and manageable architectural aspect. Secondly, there exist in literature a wide range of BPEL engines, one of which is the open source ActiveBPEL Engine.

3.2.2.6.2. *OWL-S Centric Solution*

The OWL-S based solution is in some ways similar to the previous BPEL-centric solution. Differences arise when the two languages are compared. However, a part these already described differences, it has to be noted that an OWL-S centric solution would have some interesting advantages, specifically related to AkoGrimo.

The use of OWL-S, along with some foreseeable extensions, would give AkoGrimo the necessary flexibility in semantic description of highly diverse services and resources, thus assuring the ability of adapting, through time, whatever new solution or upgrades will come to life. Moreover, the mobile world considered by AkoGrimo would highly appreciate such flexibility.

3.2.2.6.3. *BPEL/OWL-S Hybrid Solution*

The last solution considers a hybrid use of both BPEL and OWL-S. As already stated, BPEL is very well suited for business process description, whereas OWL-S is highly committed to the semantic and flexible description of resources and services.

Ideally, the use of OWL-S for services description, thus service discovery, along with the use of BPEL as the language for describing and enacting a workflow can be seen as the best solution.

Practically however, using two different languages in two different but related contexts can make some problems arise. There need to be a precise mapping between the language constructs and such mapping needs to be synchronized every time a new entity is introduced in one of the two languages. Moreover, it is often difficult to clearly trace the line that splits the use of one language from the use of the other. From the point of view of the BP Enactor it can be quite dangerous to define such line: problems could arise in the future while trying to implement specific features that would need a language more than the other.

3.2.2.7. ***Configuration and Deployment***

This section will be updated during implementation phase, because at the moment there aren't enough details to provide inputs on it.

3.3. SLA Management

3.3.1. Overview

The Service Level Agreement (SLA) subsystem, in the scope of the Akogrimo infrastructure, is a set of two main groups split between WP4.3 and WP4.4 layers. Within WP4.4 context, the SLA Management is basically involved on the negotiation phase of all the services that belong in a workflow execution, as well as the establishment of a contract as a result of this negotiation. The participants of a negotiation are the SP,, who offers his services, and the SC or better someone on behalf of him. Its goal is to check the availability of services able to carry out actions required by a user. This part of the SLA subsystem is called "**SLA High Level Services**". The other group belongs to WP4.3 layer and it is called "SLA Enforcement". This last part is more focused on the execution control of the service, and it is detailed in WP4.3 component design document, so there will not be more references here.

One of the possible purposes of the SP is to offer (and in the majority of cases to sell) several services to the SC. When a SP has developed a service and wants to publish it into Akogrimo VOs (addressed as BVO) it is also necessary to provide at least one SLA template document that will be the base of a future contract between the SP and the SC. In this way, it will be assured good service provisions to the SC. SLA subsystem offers to Akogrimo infrastructure the possibility of managing SLA templates, contracts, and to establish the negotiation of quality of service (QoS) for guaranteeing good use of the services.

In order to manage the service negotiations it is necessary for the SLA subsystem to interact with other subsystems, mainly the Policy Manager and the EMS. The main goal is to reach an agreement that will establish the conditions of usage for a concrete service in a concrete period of time. The negotiation phase will terminate when all the services have been contracted and all contracts have been saved into a repository.

The conditions of use by the SC must be accuracy defined because the service can pass through several states while it is working.

3.3.1.1. Objectives

The SLA subsystem is distributed among several places in the Akogrimo infrastructure because of its presence in different phases. One of the steps that set up a BVO is the registration of a SP and the publication of its services that are offered to Service Customers. Each service has to be associated to one or more documents called SLA-Template, that will contain service information as well as QoS offered by the SP, charging policies of use, penalties for bad provisioning, etc. Akogrimo middleware will provide a specific tool that will allow to the user to build SLA templates easily without having any know-how about how a document template is built internally.

Before one SP publishes a service, all service templates must be stored in an entity called SLA-Repository. Afterwards, during the publication phase, the SP can register the service and relate it to the templates already stored. This is the way for making the services available within the Akogrimo environment. Another phase, where SLA modules participate, is during the search of specific services. The GrSDS, in charge of managing this operation, wants to retrieve information related to all services in order to show it to the SC. For this reason a set of modules are necessary, these are SLA-Access and SLA-Translator, which will extract this section from all templates. Next step with participation of the SLA subsystem is the negotiation phase, where the SC, in agreement with the conditions that offers the SP, decides what kind of QoS he wants. In this step also participates Policy Manager and EMS subsystems with the aim of ensure the availability of the SP. Afterwards, if the negotiation has been successful, a contract is made between the SC and the SP for guaranteeing the achieved agreement. This contract must contemplate the agreed QoS together all the possible situations that could happen during the use of the service. All contracts will be saved into the SLA-ContractRepository that will be an entity analog to the SLA-Repository but storing contracts instead of templates. After that, next phases are already related to WP4.3, with the participation of the SLA on the service execution and the continuous management of the QoS fulfillment.

3.3.1.2. Example Scenario

To make services available to the overall Akogrimo environment is necessary to do two initial steps that must be carried out together. When a SP has developed and deployed a service is necessary to build a document that reflects the conditions of use, on the one hand concerning the service provision, on the other hand, essential for the service consumption. Once this document (it can be more than one) has been created it is stored into a repository together with the other services templates. Next step is to publish the service and make it visible for all the participants inside Akogrimo environment. Once all these steps are finished, the service can be requested for everyone joining the BVO where the SP is registered. When a customer requests for services, the SLA subsystem provides the functionality for extracting general info from them. This information will be showed to the user in such way he is able to know what each service is offering and to choose one of them. Once an available service is elected, several QoS parameters must be asked in order to know if the SP is able to provide them during the service execution. This is the moment when the negotiation phase starts for contracting this service. The SLA subsystem, according to several mapping metrics, will ask for the availability of the service to the SP. If the negotiation is fruitful, that means, the service is available and the SP assures that it can provide the desired QoS, there will be made a pre-reservation for the use of the service in a concrete time and with all the conditions achieved. Once the OpVOBroker, in charge of managing all the pre-reservations, confirms that all necessary services are available, there will be a confirmation of the reservation and a contract will be established where will be reflected all the conditions agreed between the customer and the provider.

3.3.1.3. Functional Capabilities

These are the set of functional capabilities identified for the SLA High Level Services:

- **To create document template**:
  - Define a document template with the QoS offered by a SP for a specific service.
- **To access documents:**
  - To offer functionalities to other modules for accessing to the data of the templates.
  - To provide a transparent way to access to the template documents without considering the interface.
- **To control the negotiation:**
  - To negotiate and achieve an agreement between a SC and a SP in relation with the use of a determinate service.
- **Create contract dynamically:**

- To establish a contract as a result of the negotiation phase where will be established all agreed conditions.

- **To store documents:**
  - To store all the templates on a repository.
  - To store all the contracts on a repository.

3.3.1.4. Position with respect to the Akogrimo infrastructure

In Akogrimo infrastructure we have addressed SLA management issues in two workpackages (4.3 and 4.4), in order to cover different aspects. SLA management, in fact, "contains" "high level" needs (e.g. provisioning of services with specific QoS and agreement on contract between service customer and service provider) as well as "low level" needs (e.g. monitoring of agreed SLA contract and violations management). In particular in this workpackage (4.4) we will present an architecture to address "SLA high level needs" such as:

- Building and manipulation of SLA-Templates. SLA templates collect service requirements and we should foresee a tool (or a service) able to write it, in the sense that the service provider should use this utility to build a right SLA template to use in Akogrimo middleware. We can foresee this tool or service as a SLA-TemplateBuilder, in this way we will be able to write a SLA without fear to make it in a wrong way. On the other hand we will foresee some additional functionality to "SLA high level" in order to access to SLA template/contract and manipulate them.
- SLA templates storing. The service provider publishes a service and associates to it one or more SLA templates, that contain information about service requirements. During the acquiring phase of a service customer requiring a business service, he asks for specific QoS. On the base of these parameters the search for one or more service providers able to provide the desired QoS takes place.
- Negotiation. A service customer needs to acquire a business service providing some input parameters, so the infrastructure services of this layer will provide functionalities in order to achieve the result. The conclusion of the negotiation will be the formalization of a contract where will be defined the requirements and conditions of use of a specific service, and also the reservation of the appropriate resources for the execution of this service.
- Final SLA contract management. At the end of negotiation phase a contract between service customer and service provider is reached. Inside it there are service requirements, QoS and policies about violation management and charging to apply. This information could be useful for the setup of the service as well as it is used for the enforcement of the SLA Management during the execution phase. This final contract is delivered to service customer and stored inside a repository.

All these functionalities are offered at this layer as grid application support services and will be detailed in next sections.

3.3.1.4.1. Interactions with Akogrimo infrastructure components

The next part shows three phases where the SLA High Level Services participates within WP4.4 layer. These phases are the publication, the negotiation and the contract establishment. For each phase there is a figure with a sequence diagram where are exposed all the interactions between the components of this layer and also WP4.2 and WP4.3 layers.

Before publishing a service the SP has to register at least one SLA-Template. Afterwards the service will be registered together some information extracted from the template. The content of this information and how could it be structured is not dear yet. The reason is that it depends of the requirements of the search phase, in concrete it depends of the requirements of the GrSDS. During the publication phase the GrSDS gets information that could be (or not) enough for the search phase. Depending on what will be the final design there are two possible ways for providing necessary information for the search phase to the GrSDS:

- During the publication phase the GrSDS retrieves from the template all the necessary information (following a fixed structure) that will be useful for the search phase.
- During the publication phase the GrSDS retrieves from the template some information that will be only useful for the user application. During the search phase the GrSDS will access to the template again in order to obtain the required information.

In this phase of the design it has been chosen the first option, which is to provide necessary information to the GrSDS during the publication of the service, so in this case while the search service operation is not necessary to access to the SLA-Template again for retrieving necessary information because the GrSDS has stored it during the publication phase.

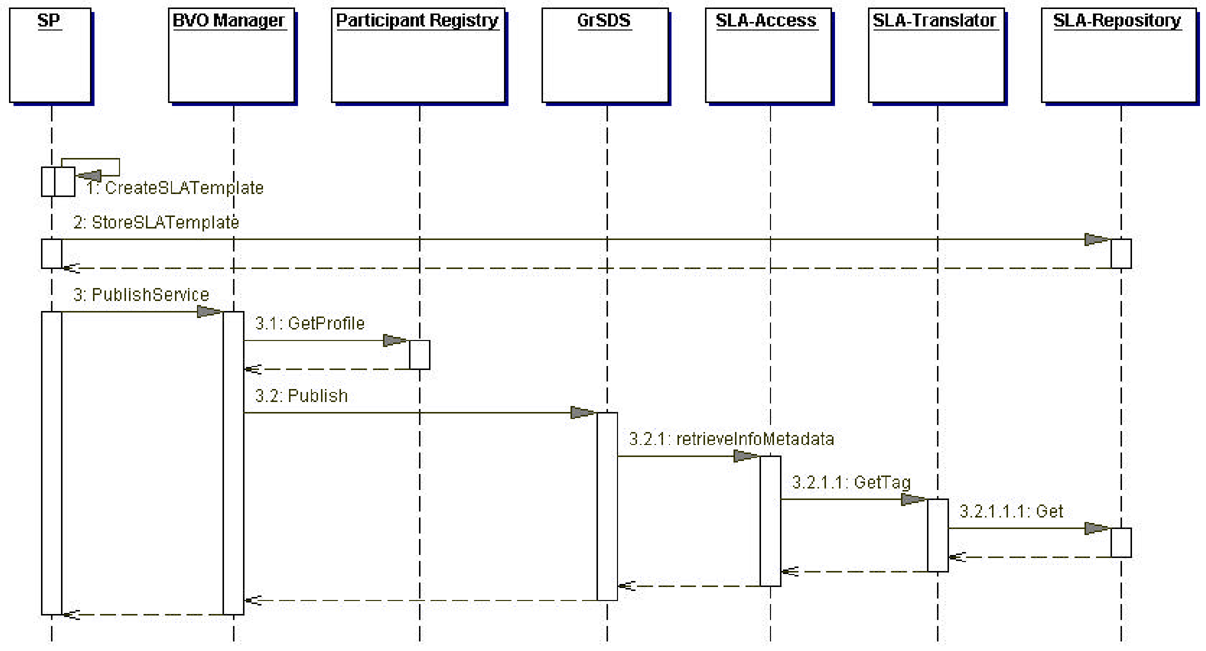


Figure 24 Sequence diagram publication

During the negotiation phase of a service the SLA-Negotiator will map the input parameters turning them into low level parameters in order to give the EMS the possibility of consulting with the SP its availability for serving this service.

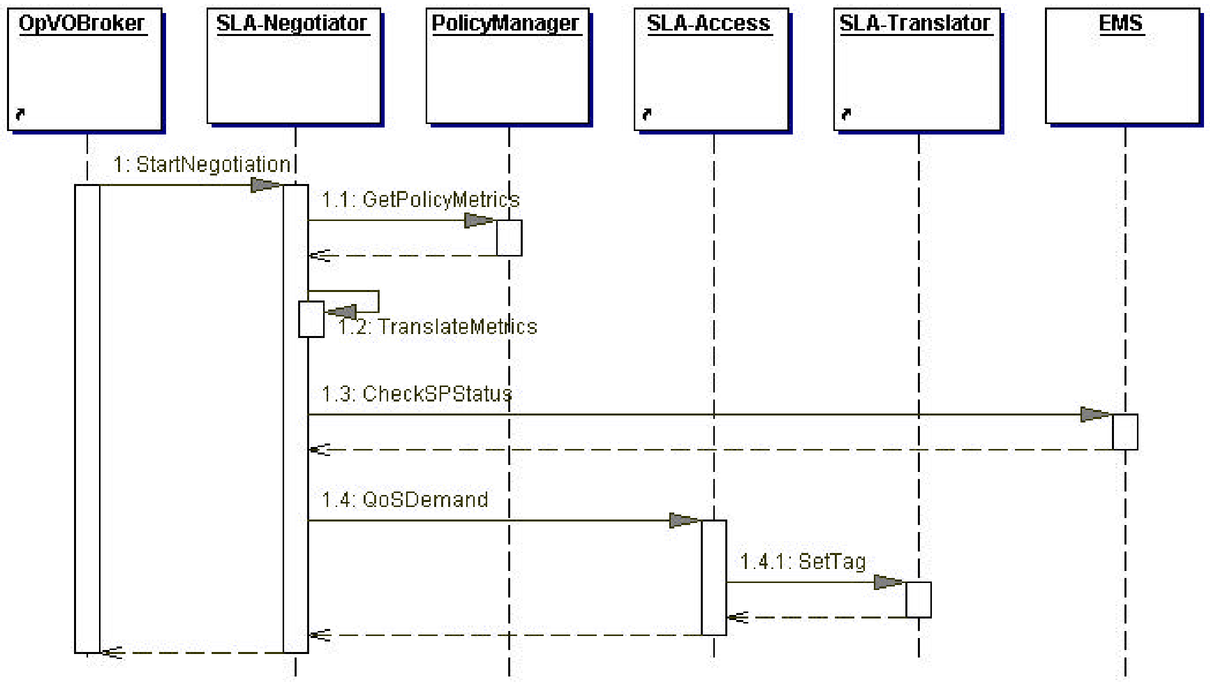


Figure 25 Sequence diagram negotiation

After the OpVOBroker confirms the use of a negotiated service the EMS will confirm the reservation in the SP. Then the SLA-Negotiator will create the contract that will contain the conditions of execution of this specific service.

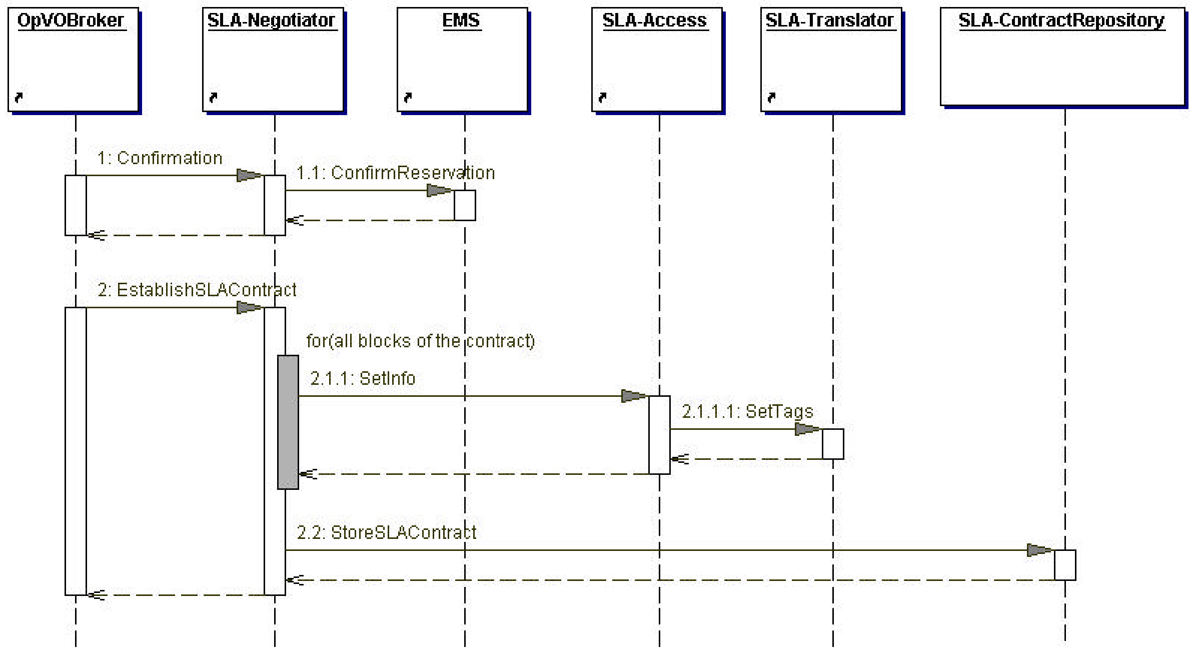


Figure 26 Sequence diagram contract establishment

This table summarize the interactions between the SLA modules and the other components of this layer:

Identified interaction	Rationale for interaction	Issues to be addressed
OpVOBroker – SLA-Negotiator	Start negotiation	
OpVOBroker – SLA-Negotiator	Confirmation	
OpVOBroker – SLA-Negotiator	Establish contract	
SLA-Negotiator – SLA-Access	Set QoS from user	TBD if this step will be finally done or it is not necessary
SLA-Access – SLA-Translator	Set QoS at the document (a pre-contract)	TBD if this step will be finally done or it is not necessary
SLA-Negotiator – SLA-Access	Set Data	
SLA-Access – SLA-Translator	Set Data to the contract	TBC if this action will be done by using the SLA-TemplateBuilder
SLA-Negotiator – SLA-ContractRepository	Store agreed SLA contract	

Table 8 Summary interactions between SLA Management and internal WP4.4 components

This table summarize the interactions between the SLA modules and the other components outside the layer:

Identified interaction	Rationale for interaction	Issues to be addressed
GrSDS – SLA-Access	Get Info in publication	TBD how and which information will be extracted from the template.
SLA-Negotiator – Policy Manager	Get policies on mapping metrics	
SLA-Negotiator – EMS	Service availability	
SLA-Negotiator – EMS	Confirm reservation	

Table 9 Summary interactions between SLA Management and components outside WP4.4

3.3.2. Architecture description

3.3.2.1. Architecture overview

The architecture of SLA High Level Services is a classical architecture service oriented, in the sense that each module is implemented as grid service. In order to implement all functionalities we will use an underlying support framework that implements WSRF specification. All entities belonging at this layer are in charge to conclude the negotiation phase and manage SLA templates and contracts. Indeed we will find functional blocks able to:

- Store SLA templates and contracts.
- Access to SLA documents in order to retrieve info from them and manipulate their contents (add, remove and update existing nodes with SLA information), in a way totally transparent to the invoker.
- Conclude negotiation phase between a service customer and a service provider.
- Request resources reservation for acquired services.
- Build a final agreement contract between who wants to acquire a service and who wants to sell it.
- Define charging policies between purchaser and seller.
- Remove expired contracts.
- Provide a tool that is in charge to help service provider in building phase of SLA template to be published when a service will be available.

With the proposed architecture we would provide all functionalities useful to realize a negotiation phase and management tasks related to SLA. We would highlight that the negotiation phase couldn’t be executed for all services requested, in the sense that some of them will be provided for free. In any case we imagine that for each business service should exist at least one SLA template published, where is specified which are the services functionalities and under which conditions the service provider can offer them. For each service requested, if the resources reservation phase is completed, then will exist a final contract. Now, for services offered by free there are specific rules and particular SLA templates (missing in some parts like charging policies for example) that won’t be negotiated. For this kind of services (offered free) the acquiring phase will be concluded in a shorter time respect to services that aren’t free, because will skip some steps and the customer should accept the services with its QoS availability. An overview of SLA High Level Services entities and a more detailed description of them is provided at 3.3.2.3.

3.3.2.1.1. Functional requirements

These are the requirements found for the SLA Management:

- A SLA-Template will be built strictly by using the SLA-TemplateBuilder tool.
- Before publishing a service is mandatory to have published at least one document template related to it. This document will be an XML language based document.
- Each SP has also to indicate policy metrics related to mapping between High Level parameters and Low Level parameters.

- No outside modules will be able to access to these documents. All the interactions must be done only by the SLA-Translator.
- The SLA-Negotiator will be a bridge between the SC and SP and it will be the one able to establish the final contract.
- The contract will be similar than the template, with adding all conditions agreed and some details/information added at runtime (e.g. unique identifier reference of grid service).

3.3.2.1.2. Non-functional requirements

These are identified several non-functional capabilities:

- **Availability:** Availability of SLA Management is bound to the availability of general Akogrimo platform. No special requirement is expected for SLA negotiation.
- **Interoperability:** SLA design is independent of a concrete technology. The use of generic specifications like WS-Agreement and XML messages interface would guarantee the interoperability with other components by design. However as a baseline, .NET technology is considered. The invocation of SLA services from client based on other technologies is envisaged as basic test.
- **Performance:** This aspect is not considered as a main issue in the first prototype. Implementation will follow main well-know software mechanisms as patterns to address the issue according to best practises and experience of design team in order to optimise the development.
- **Portability:** SLA Management components will be developed for a specific deployment case. However, the design is generic enough (decoupled from technology) so that it is foreseen that it can be ported to other platforms.
- **Reliability:** SLA-Negotiator subsystem must take into account this issue carefully since it is a critical component for the platform to run properly. Reliability issue must be framed in a general fault-tolerant strategy coordinated for all Akogrimo key components involved in this phase. This aspect should be thoroughly considered in second cycle of the project since isolated strategy would not give this feature to SLA components.
- **Scalability:** This topic is difficult to address and even more difficult to measure. Scalability is a continuum, not a binary variable, in such a way that someone can simply say: this platform is scalable or it is not. SLA Management design takes this aspect into account in order to reach a solution minimizing possible bottlenecks in a large deployment. For scalability measurement it is necessary to establish reference cases in order to verify the scalability of the solution against concrete parameters. For SLA Management services of negotiation phase two cases may be foreseen: a) scalability when number of customers requests for a QoS negotiation increases. b) scalability when deployment infrastructure increases (for instance when double size).

3.3.2.2. Use case view

3.3.2.2.1. Actors

Following actors have been identified with some participation with any of the entities that make up the SLA High Level Services block. Furthermore whatever entity, service, others that need to access to the contract or the document template, must follow the right procedure, by using the SLA modules described before for this purpose.

- **GrSDS:** The Grid Service Discovery System will require information from the SLA-Template during the publication phase of the service templates. To do this it will ask to the SLA-Access module for providing this information.
- **OpVOBroker:** This actor will only interact with the SLA-Negotiator module throughout all the time the negotiation phase spends. Its role is to be in charge of managing the negotiation for all the services that are demanded by the consumer in order to concrete at what time and which will be the service providers that will host the service execution. Once the OpVOBroker is aware of all required services will be executed on any service provider it begins to order the SLA-Negotiator that the contract has to be carried out. After the contracts are saved in the repository the negotiation cycle is completed.
- **Policy Manager:** SLA High Level Services also interacts with Policy Manager in the negotiation phase, when it is necessary to know the metrics that define the mapping from high to low level parameters. These policies are defined by the application designer and are stored in a policy db. When the negotiation starts the SLA-Negotiator will require them.
- **EMS:** During negotiation the SLA-Negotiator will ask the Execution Management Service for concrete service availability in a time scheduled. It is responsible for checking the service provider status and establishing a pre-reservation of the service use for this period in case of it is feasible. Once the OpVOBroker realizes that all the negotiation has been successful EMS must confirm the reservation of the services.

3.3.2.2.2. Use Cases

Next figure shows the use case packages that are involved on SLA. Only the "Package Publication", "Package Negotiation" and "Package Contract" will be explained below.

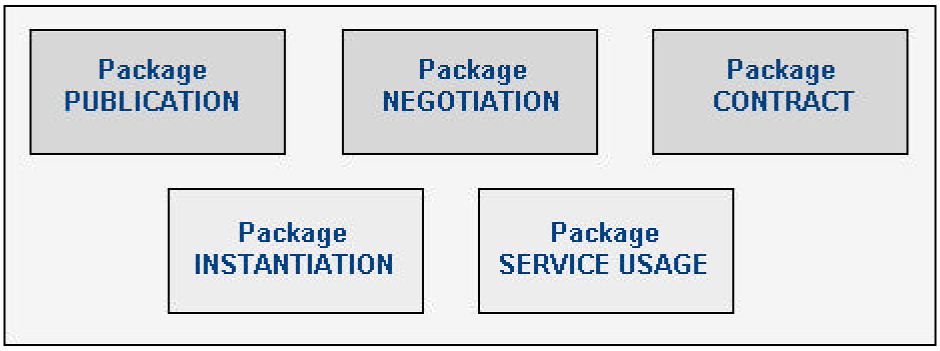


Figure 27 Use case packages

Figure 27 shows all SLA packages related to WP4.4 layer. Each package is composed by a generic use case and after this use case is split in other use cases. From now on, each time the word "service" appears in the text, it will refer to a concrete instance of any kind of service.

The publication of any service has relation with the SLA infrastructure because this includes the publication of one or more SLA-Templates associated to it. Furthermore, these templates contain information that will be useful later for the GrSDS.

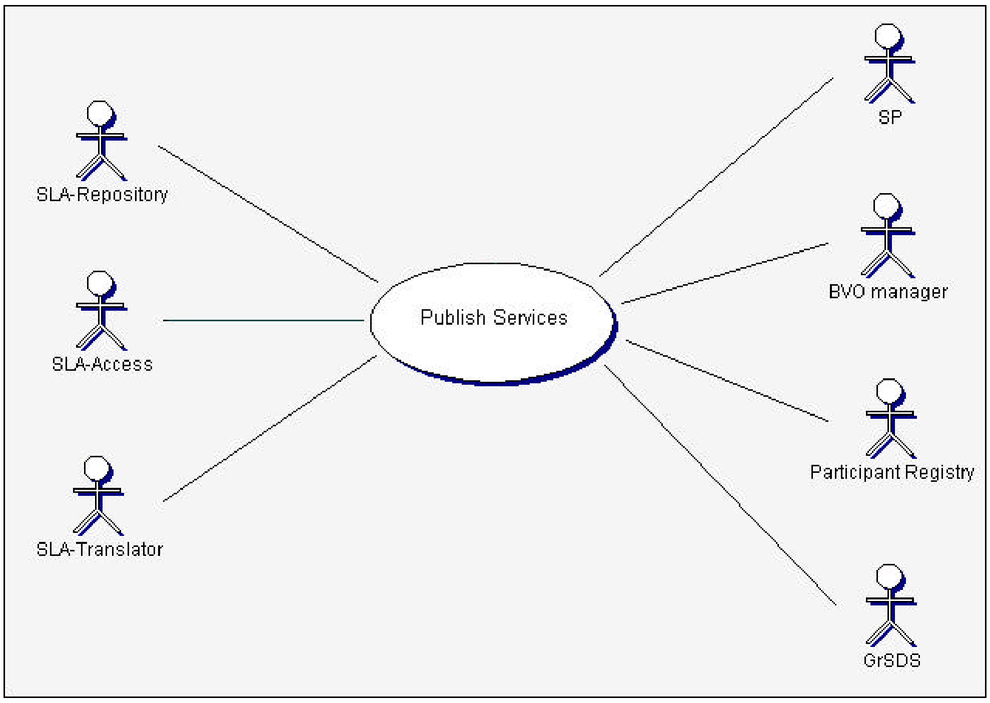


Figure 28 Use Case “Publish Services”

Next figure depicts the breakdown of this generic use case:

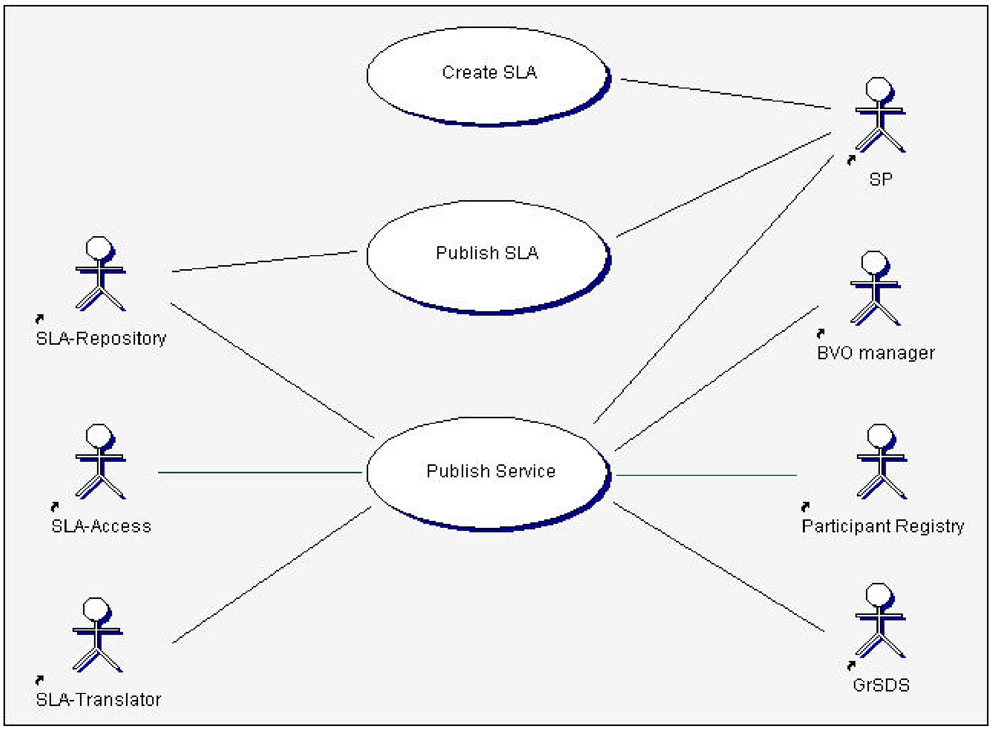


Figure 29 Use Case “Publish Services” composition

Use Case	**Create SLA**
Description	Create a SLA-Template related to a service with specific metrics. A SLA-Template is a document that establishes all the features that is composed a service. This action is responsibility of the owner of the service. Flow of events: • A SP builds the SLA-Template of the service on the basis of its capabilities and functionalities (i.e. a SP decide to offer a service with specific features on the base of resources that it has).
Actor	SP
Preconditions	The SP has to be able to guarantee what is described inside SLA-Template
Postconditions	SLA-Template document related to one service

Table 10: Use Case "Create SLA"

Use Case	**Publish SLA**
Description	The first step of the publication phase is associated with the storage of the SLA-Template into the repository. Flow of events: • SLA-Template is stored into the SLA-Repository.
Actor	SP, SLA-Repository
Preconditions	One or more SLA-Template documents exist for this service. Indeed a SP can offer the same kind of service with different QoS.
Postconditions	The SLA-Templates of the service are saved into a repository

Table 11 Use Case "Publish SLA"

Use Case	**Publish Service**
Description	The last step of the publication phase is related to the registration of the service. Flow of events: • SP starts the registration service action • GrSDS gets some information stored into the SLA-Templates from the SLA-Repository. • The service is registered
Actor	SP, GrSDS, SLA-Access, SLA-Translator, SLA-Repository

Preconditions	SP has published the SLA-Templates related to the service.
Postconditions	The service is registered together some information extracted from the SLA-Templates

Table 12 Use Case "Publish Service"

The final goal of Package Negotiation is to achieve an agreement between the SC and the SP. The main actor is the SLA-Negotiator that will manage all the cycle of negotiation between all components involved in this phase. Is necessary to assure that an agreement is reached to guarantee a concrete QoS.

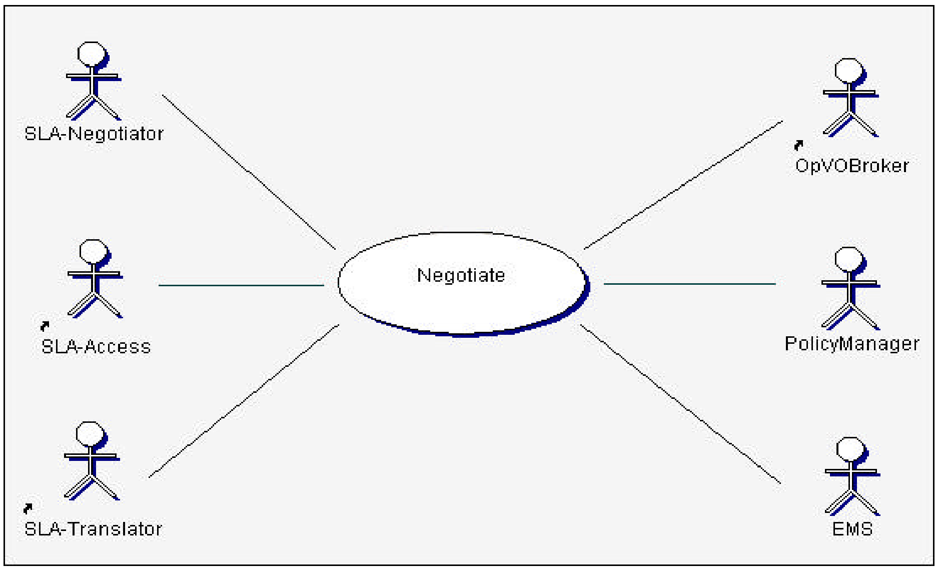


Figure 30 Use Case "Negotiate"

Next figure depicts the breakdown of this generic use case:

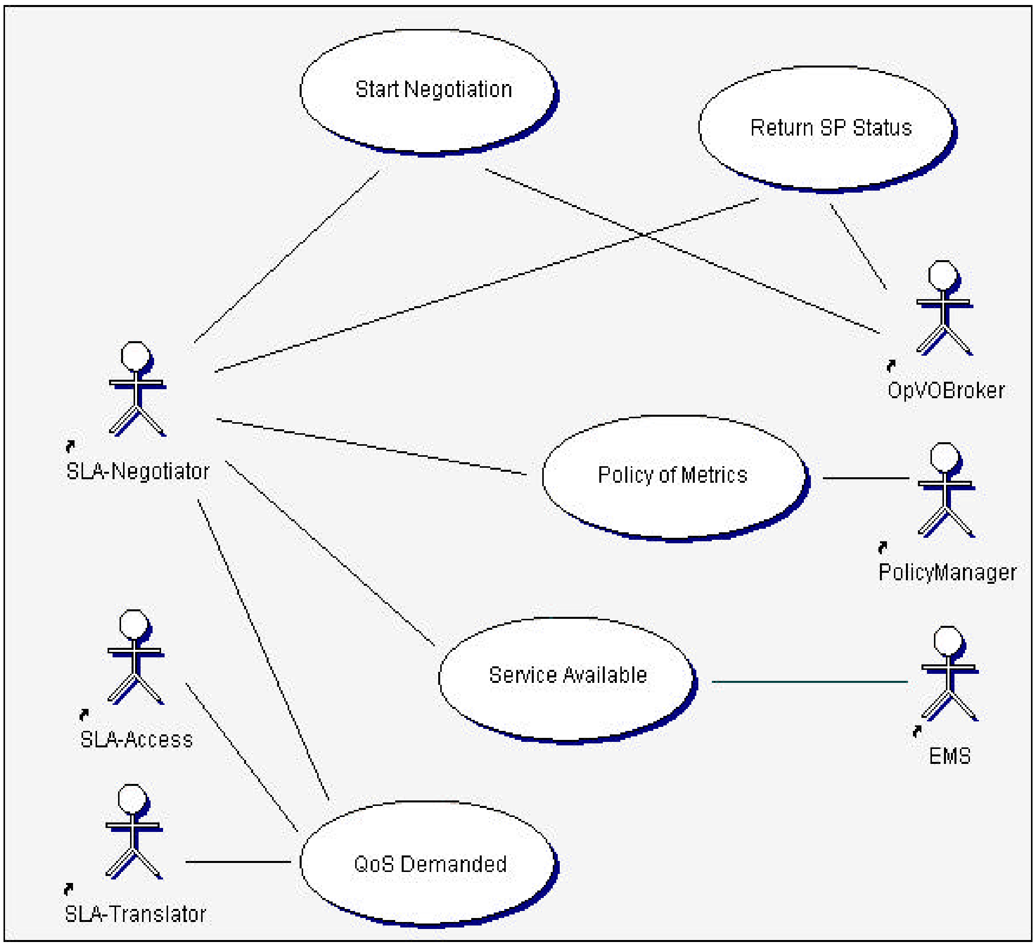


Figure 31 Use Case "Negotiate" composition

Use Case	**Start Negotiation**
Description	The OpVOBroker acts on behalf of SC and determinates values for the QoS that it is asked for. After, it starts a negotiation to check if the SP is able to fulfil what it wants. Flow of events: • The user sets the application input parameters. • These parameters are sent to the OpVOBroker who communicates with SLA-Negotiator.
Actor	OpVOBroker, SLA-Negotiator
Preconditions	The user has selected one SP and it has fulfilled the required input parameters
Postconditions	The SLA-Negotiator receives the parameters from the OpVOBroker and now is able to check the availability of the chosen SP

Table 13 Use Case "Start Negotiation"

Use Case	Policy of Metrics
Description	SLA-Negotiator asks Policy Manager for mapping policies from high to low level parameters Flow of events: • SLA-Negotiator asks for the metric policies to the Policy Manager • SLA-Negotiator translate the metrics received into QoS parameters understandable from the EMS
Actor	SLA-Negotiator, Policy Manager
Preconditions	There are policies defined with the mapping from high to low level parameters
Postconditions	SLA-Negotiator has obtained the metric policies and it has mapped the user parameters

Table 14 Use Case "Policy of Metrics"

Use Case	Service Available
Description	SLA-Negotiator passes to EMS subsystem the QoS parameters and asks for the availability of service instantiation in the SP chosen for a given time (it is a fix date for a future execution) Flow of events: • EMS receives the SP and QoS from the SLA-Negotiator • EMS asks for the availability of the SP to offer a service with this QoS • EMS returns to the SLA-Negotiator the result of this availability
Actor	SLA-Negotiator, EMS
Preconditions	There is a SP chosen and a QoS required
Postconditions	Returns the availability for this future execution request. If it is possible to host this request, all is right. Otherwise, other actions must be carried out.

Table 15 Use Case "Service Available"

Use Case	QoS Demanded
Description	SLA-Negotiator communicates to SLA-Access the QoS selected by the user Flow of events: • SLA-Access receives the selected QoS • SLA-Access orders SLA-Translator to set this information on the template.

	• SLA-Translator accesses to the template and set the QoS section. This template would be converted to a contract if the agreement were confirmed.
Actor	SLA-Negotiator, SLA-Access, SLA-Translator
Preconditions	SLA-Negotiator receives a list with the QoS values demanded by the SC
Postconditions	The template is set with the QoS communicated by SLA-Negotiator

Table 16 Use Case "QoS Demanded"

Use Case	**Return SP Status**
Description	SLA-Negotiator returns the availability of the chosen SP. There are several possible situations: • SP is available: (EMS has done a pre-reservation of the appropriate resources, only until contract is created) • SP is not available: SC will receive again the previous list. After it ought to start a new negotiation but changing QoS parameters or choosing another SP Flow of events: • SLA-Negotiator communicates to the OpVOBroker the availability of the chosen SP.
Actor	SLA-Negotiator, OpVOBroker
Preconditions	The SLA-Negotiator knows the availability of the SP
Postconditions	Show to SC the results of the negotiation and in case SP is not available return to SC the previous list with all candidates

Table 17 Use Case "Return SP Status"

The Package Contract is the result of a successful negotiation and the establishment of contracts signed between SC and SP.

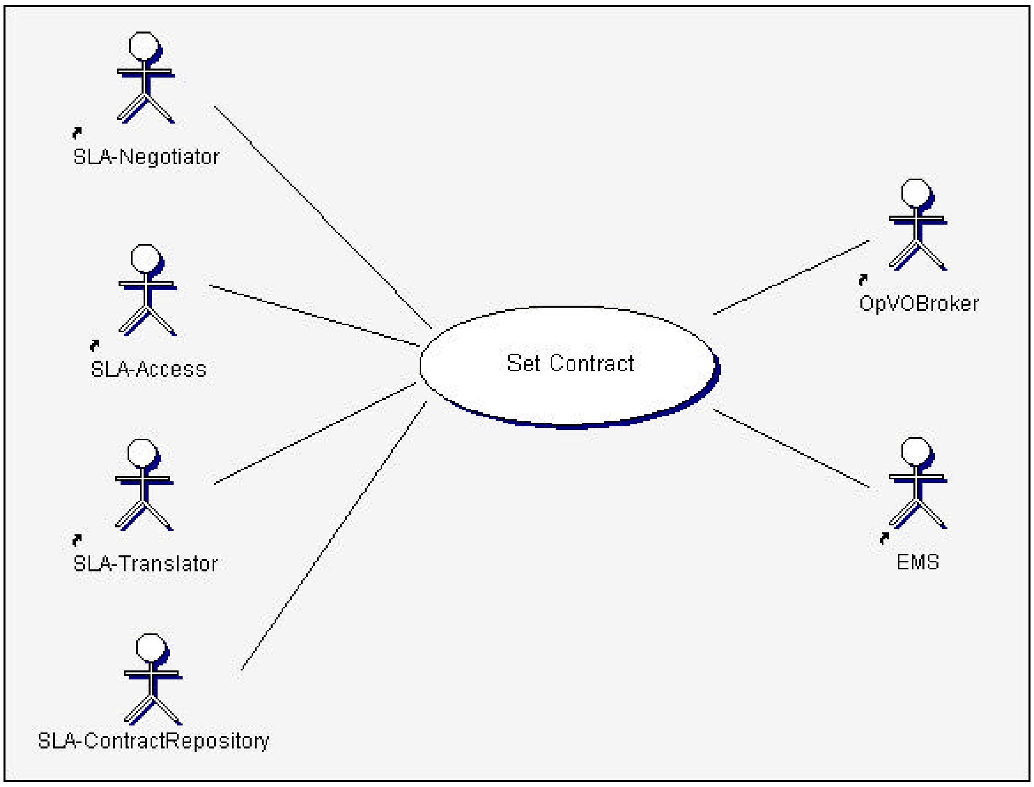


Figure 32 Use Case "Set Contract"

Next figure depicts the breakdown of this generic use case:

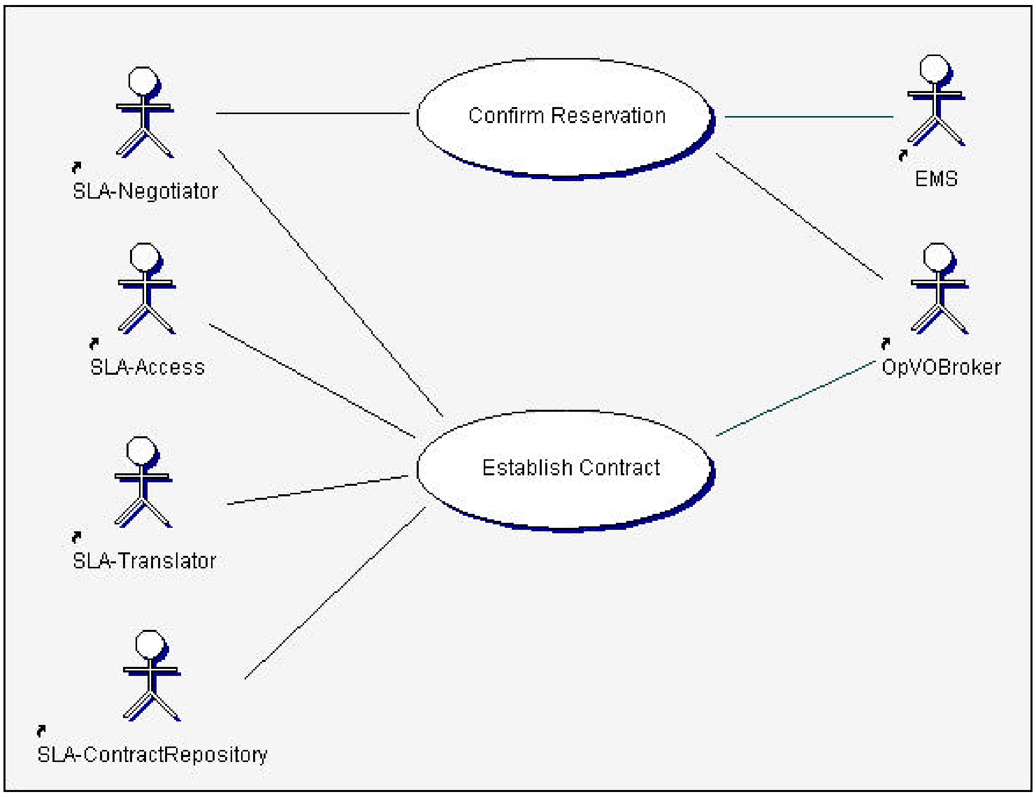


Figure 33 Use Case "Set Contract" composition

Use Case	Confirm Reservation
Description	EMS is notified to confirm the pre-reservation of a service usage for a concrete SP during the negotiation phase. Flow of events: • OpVOBroker communicates to SLA-Negotiator that the agreement has been reached and confirmed. • SLA-Negotiator notifies to EMS that the pre-reservation must be now a reservation.
Actor	OpVOBroker, SLA-Negotiator, EMS
Preconditions	The SP has a pre-reservation and OpVOBroker confirms that the agreement can be closed.
Postconditions	EMS confirms the reservation for the service use on that SP.

Table 18 Use Case "Confirm Reservation"

Use Case	Establish Contract
Description	After the reservation is confirmed, an SLA-Contract is created. It will be the base of the fulfilment of the required QoS. This contract will be stored in a repository Flow of events: • OpVOBroker orders the contract establishment • SLA-Negotiator creates the contract by using SLA modules • The contract is saved in a repository
Actor	OpVOBroker, SLA-Negotiator, SLA-Access, SLA-Translator, SLA-ContractRepository
Preconditions	There is a reservation in EMS for a service use in a concrete SP.
Postconditions	SLA-Contract document is created and stored in a repository.

Table 19 Use Case "Store Contract"

3.3.2.3. Conceptual architecture description

In below picture (Figure 34) are highlighted entities belonging to SLA High Level Services architecture and main interactions between inter and intra WPs.

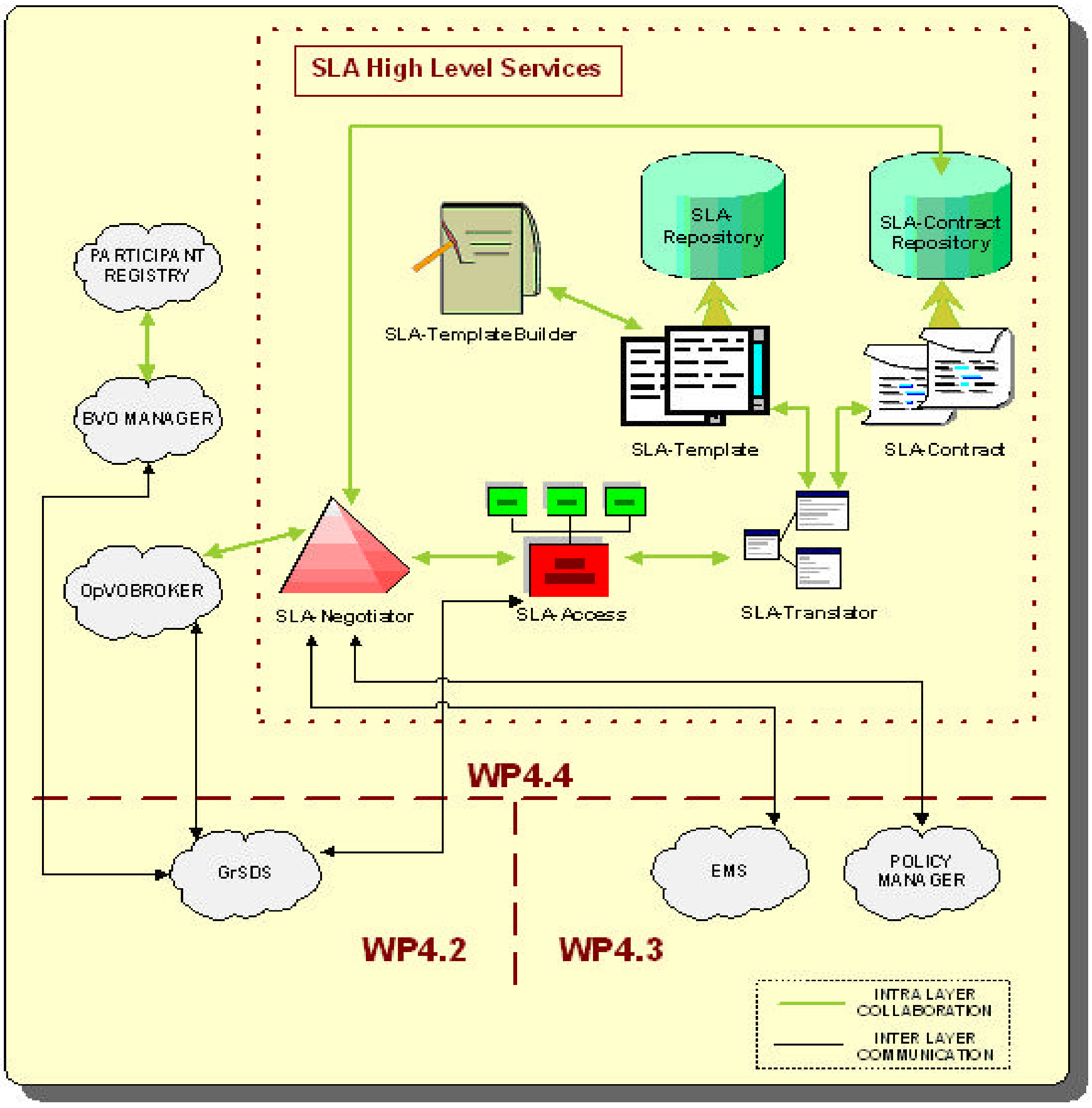


Figure 34 SLA High Level Services architecture

Following modules and tools have been identified in the design of SLA High Level Services architecture:

- **SLA-Access**. This module provides all the functionalities for getting info from/setting info on any SLA document (SLA-Template or SLA-Contract). All the interactions between the other modules and the documents will be managed by the SLA-Access, because these modules won't be allowed to access directly to them.
- **SLA-Translator**: This module is used by the SLA-Access as a way to reach to the SLA document templates. It is the one able to make read/write actions against the templates and thus to make them transparent for the SLA-Access module.
- **SLA-Negotiator**: This subsystem is in charge to carry out the negotiation phase between a service consumer and a service provider to reach an agreement.
- **SLA-TemplateBuilder**. It represents a tool that helps the service provider to produce a SLA template for the service to be published.
- **SLA-Contract**. This is the formal contract established between a SC and a SP.

- **SLA-ContractRepository**. Repository that stores all established (live) SLA-Contracts.
- **SLA-Template**. Document based on XML language that describes the SLA metrics, penalties and charging policies associated to a service.
- **SLA-Repository**. Repository where the SLA-Templates of all published services are stored.

3.3.2.3.1. *External interfaces*

The external interactions of SLA High Level Services identified involve entities belonging to WP4.3 and WP4.2. In details we will have:

- GrSDS – SLA-Access. The method *objectMetadata retrieveInfoMetadata(string serviceID, string SLATemplateID)* allows GrSDS to retrieve some metadata info from published SLA-Template and associate them to the interface of service published in GrSDS.
- SLA-Negotiator – Policy Manager. The method *objectPolicy getPolicyMetrics(object contextRequestor)* get policies on mapping metrics.
- SLA-Negotiator – EMS. The methods *object checkSPStatus(string SP, objectQoS QoS)* and *boolean confirmReservation(string SP, string serviceID, objectQoS QoS)* allow, respectively, to verify if the SP is able to provide a requested QoS for a specific service and to confirm resources reservation at the end of negotiation phase.

3.3.2.4. *Logical architecture description*

In below picture (Figure 35) the internal communications within WP4.4 entities are described for all the phases in which SLA entities are involved.

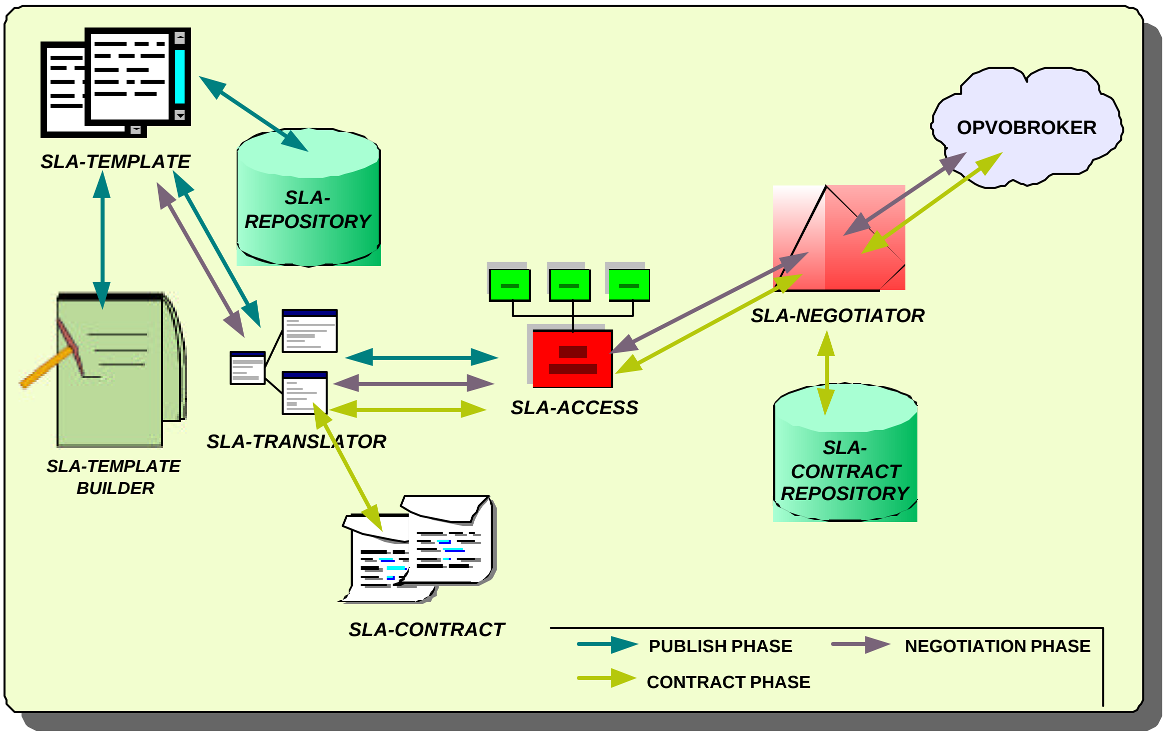


Figure 35 Steps cycle for each phase

3.3.2.4.1. Internal interfaces

The internal interactions between SLA High Level Services with themselves and with OpVOBroker component are fully detailed below:

- OpVOBroker – SLA-Negotiator. The method *boolean StartNegotiation(string SP, objectQoS QoS)* enables the negotiation of a service in the SP. The method *Confirm(string SP, string serviceID, objectQoS QoS)* notifies to the negotiator to carry out the reservation. The method *boolean EstablishSLAContract(string SP, objectQoS QoS)* starts the mechanism for elaborating the contract between SC and SP.
- SLA-Negotiator – SLA-Access. The method *QoSDemand(objectQoS QoS)* creates a document based on the template of the service that will contain the QoS requested for the user. The method *SetInfo(XMLDoc template, objectInfo info)* fills a concrete section of the contract with the information coming from the negotiator.
- SLA-Access – SLA-Translator. The methods *objectInfo GetTag(XMLdoc doc, string section)* and *SetTag(XMLdoc doc, string section, objectInfo info)* order to the translator module to extract or fill in respectively a specific section from a document.
- SLA-Negotiatior - SLA-ContractRepository. The method *boolean StoreSLAContract(string SC, string SP, XMLdoc contract)* stores the contract established by SC and SP.

3.3.2.5. Involved technologies

SLA High Level Services will be deployed over the WSRF.NET middleware [41] and all the modules will be written in c# language (with the Microsoft Visual Studio .NET 2003 tool). The SLA-Template and the SLA-Contract will be XML documents and they will follow the standard WS-Agreement. A new tool will be created for building document templates with the right structure. This tool will be also developed in c#.

SLA-Repository and SLA-ContractRepository will be organized for the first prototype as a typical structure of file system. Later the repositories could be managed through XML-Native databases.

This table summarizes the interactions (from the protocol communication point of view) between the SLA High Level Services entities and the other components:

	SLA High Level Services entities	GrSDS	OpVOBroker	Policy Manager	EMS
SLA-Negotiator	WS requests (SOAP over HTTP)		WS requests (SOAP over HTTP)	WS requests (SOAP over HTTP)	WS requests (SOAP over HTTP)
SLA-Access	WS requests (SOAP over HTTP)	WS requests (SOAP over HTTP)			
SLA-Translator	WS requests (SOAP over HTTP)				

Table 20 Summary communication protocols

3.3.2.6. Configuration and Deployment

Next components will be deployed within Akogrimo environment as is exposed below:

- SLA-Negotiator will be deployed together the OpVOBroker module because they need to maintain direct communications during the negotiation phase for each service that is asked for be used. The SLA-Negotiator must be visible for the other components that have interaction with it already commented.

Next four modules will be deployed once for administrative domain of the whole VO:

- SLA-Repository will store all SLA-Templates. On a first implementation the templates (and after the contracts) will be stored in a File System structure, each one identified with a concrete name. The way to find each document will be to query directly through its name. In a second implementation all these documents will be stored in XML-Native databases where the xml documents will be added.
- SLA-ContractRepository will store formal contracts between SC and SP and it will be deployed and structured in the same way as SLA-Repository.
- Since the role of SLA-Access is to provide access to all SLA templates and contracts, it’s not necessary to distribute this component for too many machines. In this way, it will be deployed together both repositories.
- SLA-Translator is able to make read/write actions on both (SLA-Template and SLA-Contract) SLA documents, so it should be also deployed together both repositories.

This is a simple possible distribution of all the modules that compose SLA High Level Services along the Akogrimo infrastructure.

Further update on this section could be done during implementation phase.

3.4. Security

Emerging Web services security specifications address the expression of Web service security policy (WS-Policy [43], XACML [44]), standard formats for security token exchange (WS-Trust [12], SAML [46]), standard methods for authentication and establishment of security contexts and trust relationships (WS-SecureConversation [47], WS-Security [45], SAML [46]), mechanisms and procedures for mapping trusted information about users from one domain into authentication and authorization information required by resource or service provider from other domain (WS-Federation [48]), and standard mechanisms describing how access policies are specified and managed. These specifications may be exploited to create standard, interoperable methods for these features in Grid security. But they may, in some cases, also need to be extended to address the Akogrimo security requirements.

Some delegation mechanisms should be defined in order to support the users and services interaction at VO level by leveraging the SAML standard and its possible extension for delegation purposes [11]

3.4.1. Overview

The Grid Application Support Services layer requires supporting security mechanism enabling communication between involved parties by satisfying the security policies between requestors and requested services. At this layer the security purposes are to address identity and access constraints of BVO and OpVO participants and to define a secure mechanism to allow users and services belonging to diverse organizations to access VO services. The VO is hence viewed as a 'large' (virtual) administration domain owning its security services for managing security among VO entities and bridging the VO members' security domains.

Regarding the Authentication process, using some tokens (SAML) generated at network layer, we will identify the entities (i.e. services [10]) involved at VO level.

3.4.1.1. General Concepts

In a dynamic environment, such as in a VO, each entity is bound to its own administration domain where it is identified. The VO is built upon sharing of resources, capabilities and information derived from several organizations or groups owning security mechanisms and policies in order to achieve the common objective.

At this layer Akogrimo aims to define a common mechanism to ensure a secure relationship between organizations involved within a VO; then, a general and common security mechanism must be defined to support the cross-domain interactions. Starting from a general security problem inside VO we will address, in particular, security in base VO and operative VO (the difference between them has been highlighted in previous paragraphs, we will indicate them respectively as BVO and OpVO).

Before starting with description of services involved for each administration domain and relevant to BVO, some considerations must be taken into account:

- VO entities: with the term 'entity' we refer to a resource or user visible and interoperable at VO level. Following the OGSA aims, both resources and user are virtualized as Grid services and at VO level a user is represented by the 'User Agent' while a service by the 'Service Agent'. User/service agent is the way to allow these kinds of entities to interact among them in the Akogrimo BVO context.

- Each entity belongs to an administration domain: The BVO will maintain information about the most representative entities (e.g. services providers, network providers, service customers, etc.). On the other hand, each administration domain is in charge of managing its members and should provide security policies for accepting foreign mobile members.
- An entity can be, also temporarily, bound to different administration domain and thus participate in the BVO. The A4C subsystem is taken into account for authentication of entities.
- Service and User Agents: are strategic components (at the moment, used also for security purpose) which represent user/service interactions inside a BVO/OpVO. They may provide control on the invocation, for example checking if a given user may invoke a service. Each user and each service have, inside the OpVO, an associated user/service agent acting as delegate.

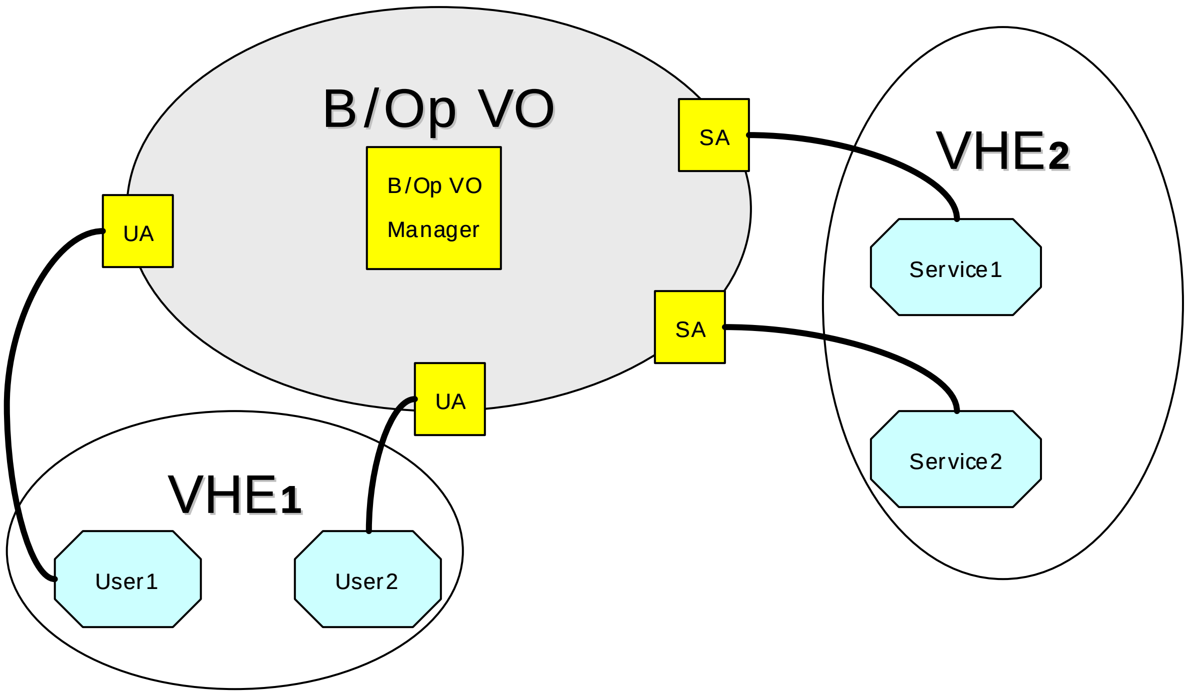


Figure 36 UA and SA overview

3.4.1.2. Example Scenarios

Security at this level involves several administration domains bound together in the common Virtual Organization. In the following we explain three main scenarios for interacting within the BVO and OpVO: membership (service or user) authentication, the buying of services together with Operative VO creation, and finally how to use these services.

3.4.1.2.1. BVO Membership authentication

User wishing to access VO (in particular BVO or OpVO) services has to be registered. By means of registration phase the user is authenticated within the BVO and a session (*BVO session*) is established enabling him to require services for a fixed time.

In Figure 37 a user who accesses a foreign domain is at first authenticated at network layer. This first step has, as result, a SAML token returned to the user. By means of this token the user requires to access to Grid services by asking his delegate UA to create (or renovate) his BVO session. To achieve this aim, the UA is able to ascertain the user credential by communicating (using BVO Manager) with A4C subsystem. At the end, the UA will forward the user request to BVO Manager for creating a new BVO session, checking the user rights at BVO level and setting up the environment for the user requests.

A trust communication already has been established between the UA and BVO Manager before the user request allows the UA to act on behalf of (and thus as a delegate of) user at BVO level.

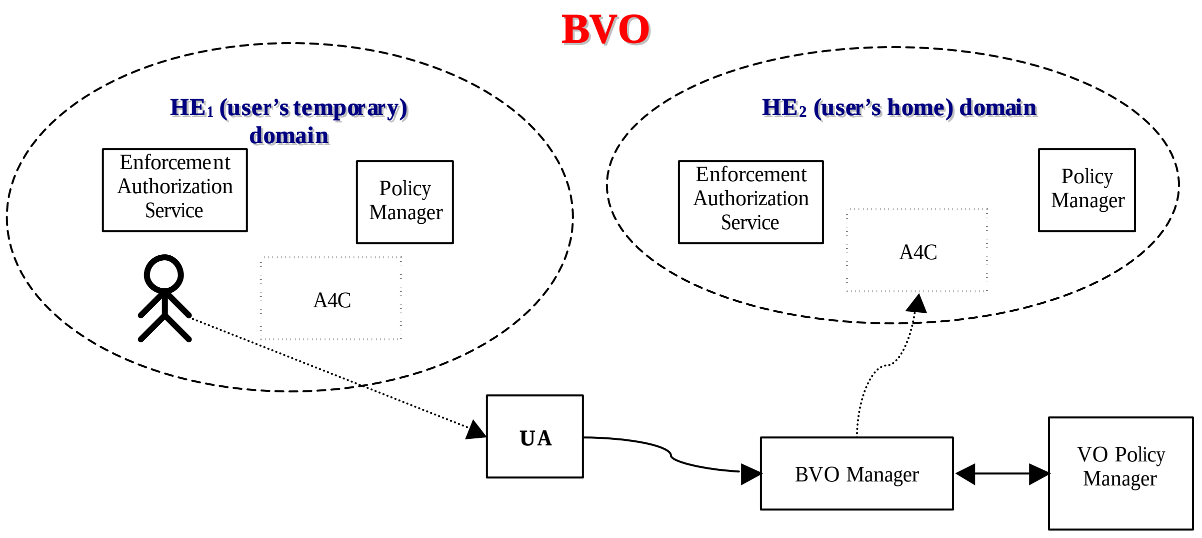

BVO
HE1 (user's temporary) domain
Enforcement Authorization Service
Policy Manager
A4C
HE2 (user's home) domain
Enforcement Authorization Service
A4C
Policy Manager
UA
BVO Manager
VO Policy Manager

Figure 37 Membership Authentication within BVO

3.4.1.2.2. OpVO creation

During the purchasing phase of a service(s) provided by third parties a new OpVO is created to ensure service usability by several users. For each user, that is going to use the service, a user agent is associated to him as well as a service agent for the requested service. Inside a BVO, a member that covers the role of SC is able to buy services. For instance (in the case of e-Learning scenario) a professor, wishing to provide the Field Trip services to his students, has to require the purchasing to his 'academic institution'. Therefore, the SC (academic institution) pays for the services, while its users (students) can pay the SC for these services, for instance, by means of university taxes.

These steps (see Figure 38) can thus be summarized in:

1. The user requires the service purchasing to his SC member by indicating his identity (SAML token), his context, what is researched and the constraints.
2. The SC will check the identity of requiring user by taking advantage of A4C subsystem at network layer and its right by interacting with Policy Manager (in this step the SC can also check how the user will pay for the service that the SC will buy).
3. The request is forwarded to the BVO Manager (BVO Manager) that will check the SC rights at BVO level.

4. The request (indicating the users that will utilize the service and user/customer profiles as well) is then sent to Workflow Manager acting for service negotiation and OpVO instantiation.
5. The OpVO is created by Workflow Manager including the UAs and SAs for the users and services involved within OpVO. These agents are supplied of security information valuable at BVO level.
6. The UAs and SAs will perform some initialization procedures in order to identify roles and users that will take part to the OpVO.

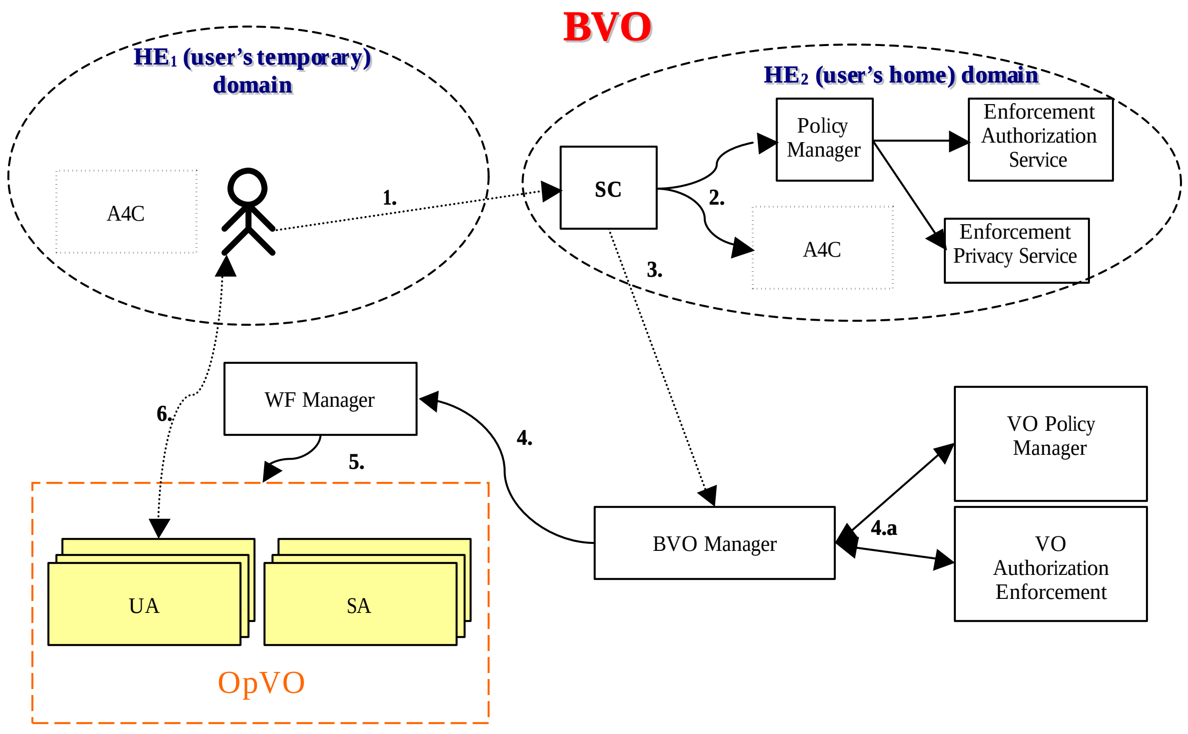

BVO
HE1 (user's temporary) domain
HE2 (user's home) domain
A4C
1.
SC
2.
Policy Manager
Enforcement Authorization Service
A4C
Enforcement Privacy Service
3.
WF Manager
6.
4.
5.
VO Policy Manager
BVO Manager
4.a
VO Authorization Enforcement
UA
SA
OpVO

Figure 38 OpVO creation

3.4.1.2.3. *Using Services*

In the distributed authorization scenario there are at least two administrative domains involved: one of the requestor subject and the second one of the requested resource. In a dynamic and mobile environment such as Akogrimo, the requestor and requested resource can temporary join another administration domain, but some of their information (e.g. identity, profile) is kept in their home domain.

Figure 39 shows a possible security model in Akogrimo for accessing third party services and for simplicity many details of security process, such as auditing, client-side authorization and privacy, are omitted. Supposing the user has been identified within the BVO after the registration phase, now the user wishes to use a service bought in previous steps and by means of his UA (operating on his behalf) makes the request to access the required functionality. Ideally the UA can be thought as a service bound to user hosting environment. But that doesn't mean the service lives in this hosting environment, just as the SA doesn't live in the related service's hosting environment. In this simple scenario the SA, UA and OpVO Manager are linked together into the OpVO meaning these entities represent the environment required for the execution of the OpVO. In general, these entities, together with supporting services at BVO level, form a common group enabled to communicate together in trusted way.

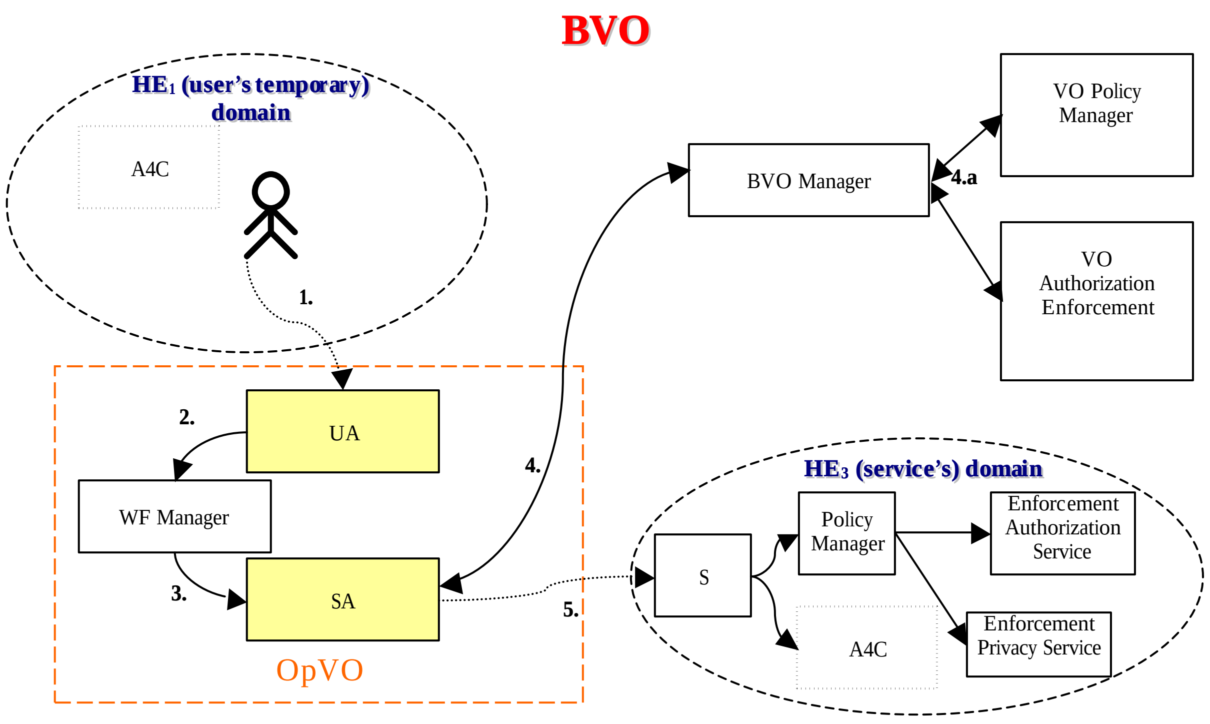

BVO
HE1 (user's temporary) domain
A4C
1.
BVO Manager
VO Policy Manager
4.a
VO Authorization Enforcement
UA
2.
WF Manager
4.
SA
3.
5.
OpVO
HE3 (service's) domain
S
Policy Manager
Enforcement Authorization Service
A4C
Enforcement Privacy Service

Figure 39 Security implications of accessing to Services

During this phase we want to enable a user identified in a hosting administration domain to benefit from a service belonging to another administration domain. Before starting it must be taken into account that the UA manages the registration session for the User and acts like a proxy for accessing BVO services. The following steps are taken to handle the security of the request:

1. The UA tests if the user session is still valid and the user profile has not changed since the last service invocation and eventually asks for an update of inconsistent info.
2. In the next step, the User Agent will communicate with WorkFlow Manager acting as delegate by asking for the requested service and explaining the user credential.
3. The Workflow Manager knows the logic of business process and will require access via a SA to a Service previously bought.
4. The Service Agent checks if the user has the authorization rights to access the service. This check involves the interaction between the SA and the BVO Manager that takes advantage of VO Policy Manager and VO Authorization Enforcement for accomplishing its purposes.
5. Then, the request is sent to the corresponding Service having the SA as proxy. Before accomplishing the request to a service belonging to the other hosting environment, some checks must be considered at this level; more details can be found in D4.3.1 [8].

3.4.1.3. Functional Capabilities

At VO level the Akogrimo's security model must address the following security capabilities:

- **Authentication**. Authentication means the capability of identifying entities. Users and services require authentication in a secure environment.
- **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. In order to deal with only the task(s) intended to be performed, some constraints (e.g. lifetime) must be bound to delegating entity. At VO level the User/Service Agent is the concrete example of delegation since they act on behalf of their users/services.
- **Authorization**. In general, service access has to be controlled in order to allow only the authorized access and under authorize conditions. The service is associated with authorization policies as well as the requestor with some invocation policies.
- **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, constraints, privacy rules etc.

3.4.1.4. Position with respect to the Akogrimo infrastructure

3.4.1.4.1. Interactions with Akogrimo infrastructure components

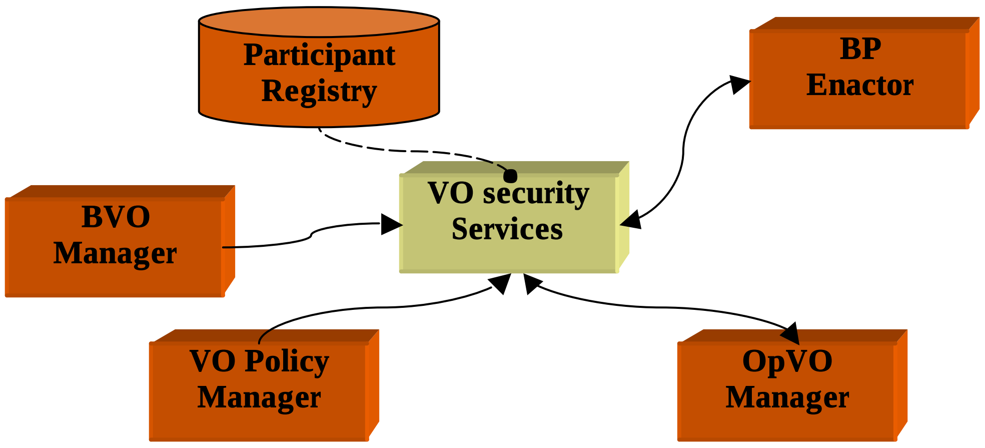


Figure 40 VO Security Services inter WP4.4 interactions

The relations with other components of Akogrimo Application Support Layer can be summarized:

Participant Registry: the UA/SA belonging to VO Security Services have to be dealt with as other services at VO level. Their information is hold in the VO Participant Registry because they represent the users and services involved in business processes.

VO Policy Manager: actions inside every VO have to be controlled in such way each participant will be subordinated to some VO and Business Process rules in order to access to VO services and this service will provide functionalities for these purposes. This component is requested for security purposes since it is able to take the current context and requested action and returns a decision to eligible VO Security Services (i.e. VO Authorization Enforcement and/or VO Privacy Service) whether the requested action (wholly or "partially") can or cannot be carried out.

Business Process Enactor (BP Enactor): interacts with the secure components representing the user and the service at VO level (i.e. User Agent and Service Agent) during the workflow enactment process.

BVO/OpVO Manager: in some cases a restricted security chain is requested for guaranteeing a secure communication among services involved within the BVO/OpVO. In fact, at VO level some interactions should be bounded to a restricted group of participant; for instance it may be required that all participants of an OpVO belongs the same secure group and no participant of other OpVO can interacts with the first.

Identified interaction	**Rationale for interaction**	**Issues to be addressed**
VO Security Services → VO Policy Manager	Set/Unset security policies	Common way to interoperate with security services, even if we should use different underlying WSRF frameworks.
Base VO Manager → VO Security Service	Participant Secure Group definition.	
VO Policy Manager→ VO Security Service	Requests for security interventions in the case of violations	
BP Enactor ←→ VO Security Services	Communication with user's/service's secure proxy	
VO Policy Manager → VO Security Services	Whether policy invalidation is triggered, the Security Services must take an enforcement action.	

Table 21 Summary interactions between VO security and internal WP4.4 components

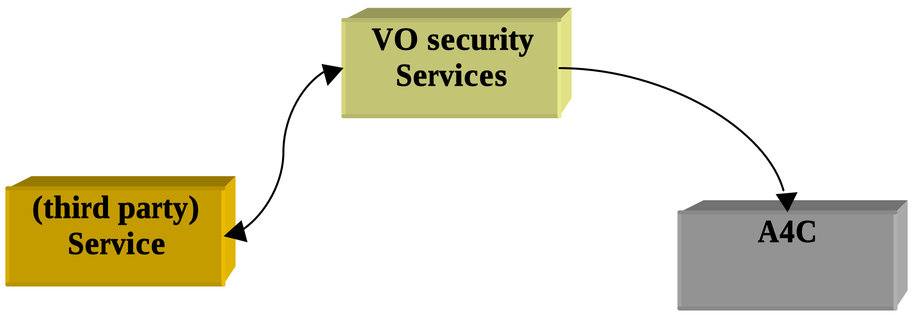


Figure 41 Security interactions with other layers

The interaction between the VO Security Services and the A4C/(third party) services can be summarized:

A4C: users and services have to be identified also at VO level in order to support the Authentication process.

Third party service: includes services like Business Services and infrastructural services (e.g. EMS)

Identified interaction	**Rationale for interaction**	**Issues to be addressed**
VO Security Services → A4C	For authentication purposes	Common way to interoperate with security services, even if we should use different underlying WSRF frameworks and security features.
VO Security Services ←→ (third party) service	For allowing a secure access to/from services belonging to an administration domain	

Table 22 Summary interactions between VO security and components outside WP4.4

3.4.2. Architecture description

3.4.2.1. Architecture overview

In our first analysis we refer to a static structural model in order to show the sub-modules (that are being developed as separate units) for the security framework defined at Grid Application Support Layer level.

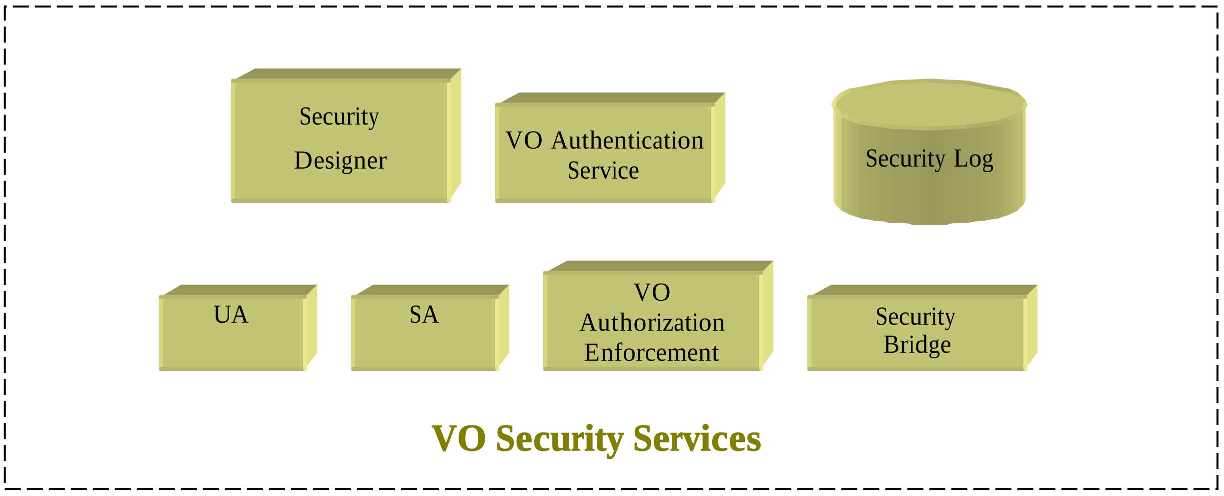


Figure 42 Security architecture overview

The entities outlined above want to be the main Akogrimo security services required at VO level and they define different behaviours for security enactment. These main security entities and their functionalities at VO level can be summarized in:

- **UA**: is in charging of checking the user's registration session and holding the user credentials for accessing to the BVO/OpVO (i.e. what are its BP roles in the BP).
- **SA**: knows which user roles are allowed to access functionalities of represented service since it knows the possible BP roles of actors that can invoke its service. The SA will check if the requestor credential corresponds to one of possible roles it manages and will verify this credential. For instance, the credential of requestor indicates the "Doctor Robinson" will access to some functionalities provided by service S. At first, the SA will check if the 'Doctor' role can require the functionality.
- **VO Authentication Service**: each Virtual Hosting Environment (i.e. we mean to refer it as "domain" that could represent generic SPs or NPs or schools or banks or hospitals, etc.) should provide means to authenticate itself within the BVO. For each domain is foreseen a VID token (a SAML token) that enable it to access VO environment. This VID token should be authenticated by VO Authentication Service taking advantage of A4C system.
- **VO Authorization Enforcement**: is the agent in charge of actualizing an action issued by VO Policy Manager. Its tasks are to stop, permit or trigger actions on a given VO service.
- **Security Designer**: is the component that allows defining security policies for services at VO level. These policies will be applied both for regulating service use within the VO and for regulating the accesses from VHE
- **Security Log**: is the repository in charge of maintaining security information.
- **Security Bridge**: in charge of mapping/translating different security mechanisms and credentials in order to enable the interoperability among different administration domains. WS-Trust specifications and in particular the STS (Security Token Service) component are going to be taken into account in order to realize the Security Bridge.

3.4.2.1.1. Group Token and Trust Relationships

A trust relationship is made when one entity is willing to rely upon a second entity to execute a set of actions; a direct trust communication between parties is thus established enabling them to accept as true all (or a subset of) the claims in the token sent by the requestor [12]. At VO level a solid trust relationship among VO members must be defined in order to allow a secure but lightweight communication. In the example scenarios (see section 3.4.1.2) the main interactions between BVO/OpVO services are depicted assuming that their communication is secure, that is the entities involved are brought together in a restricted membership group enabling them the secure communication: each entity of the group has to communicate with other entities of the group. To achieve this goal, the WS-Trust [12] model can be used for defining a framework enabling the Grid services to securely interoperate: each Grid service interoperating in a group must provide a security token in each of its requests and one way to demonstrate the authorized use of this security token is also to include a digital signature using the associated secret key. The security token represents a collection of claims that may include name, identity, key, group, privilege, capability, attribute, etc. Moreover, WS-Security, that describes enhancements to SOAP messaging to provide quality of protection, also provides a general purpose mechanism for associating security token with SOAP messages and describe how to encode binary security token (e.g. X.509 certificates [51], Kerberos tickets [50], SAML assertions, etc.). Lastly, WS-Trust [12] provides a mechanism by which the level of trust to claims and assertions presented by others/entities is defined and expressed in the Policies (i.e. WS-Policy [43], WS-PolicyAttachment [49], WS-SecurityPolicy [52]).

In general we need tokens to communicate in a secure environment, and then we can foresee different kind of tokens as listed below:

- **GT** is a security token that allows a user/service to access in a restricted secure environment (called group). The group claim the requestor belongs to and we distinguish two kinds of GT (detailed in next bullet points) enabling the communication with the 'static' services present within the BVO and others enabling the communication with 'dynamic' services within the OpVO.
- **MGT** enables the UA/SA to interoperate with services for VO management (i.e. BVO Manager, Workflow Manager, etc.) or among services for VO management itself.
- **SGT** enables the interoperability among short-lived services related to the creation of an OpVO, for instance between UA and SA.

.

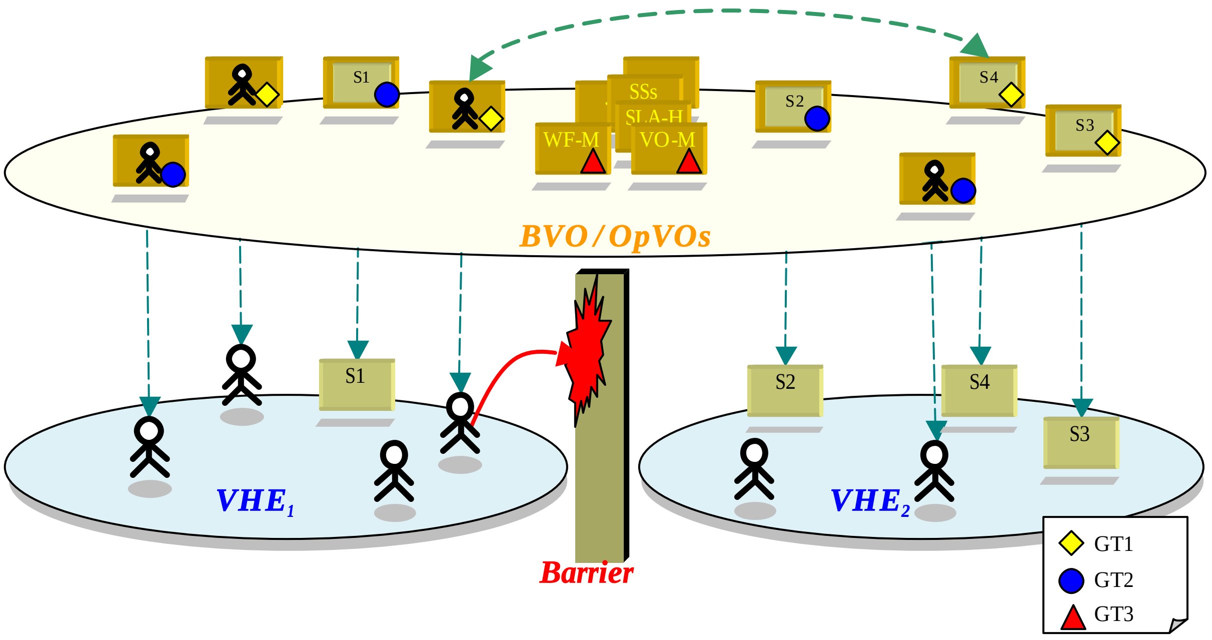


Figure 43 Trust relationships

Figure 43 illustrates relationship between two VHEs and their VO where each VHE represents an organization providing users and services to the VO. To achieve this aim, the two levels of trust that are established between VHE and VO (user/service and its agent), and within the VO itself (among VO services, UA/SA included). The first one represents a peer-to-peer relationship between parties (realizable for instance by means of WS-SecureConversation [47]), while the trust relation among parties within VO is achievable by means of GTs (for instance, all services belonging to an OpVO have the same GT). From an implementation point of view, these trust relations could be built on the base of a security context described in WS-SecureConversation [47].

3.4.2.1.2. *VO Authorization*

By taking into account the authorization framework described in [6], the authorization decisions are made based on authorization information provided by authorities. These authorities at VO level must have a direct (or a delegated) relationship with both the authorization subject (e.g., the User Agent to which the authorization is issued) and with the resource that is the target of the request that prompted the authorization (e.g., Service Agent). A possible way to implement these relationships may be by using the Group Token concepts described in the paragraph above. To achieve this aim, (see example in Figure 44) for the authorization process we identify:

Subject: An entity that can request, receive, own, transfer, present or delegate an electronic authorization as to exercise a certain right. At this level, the subject may be identified as the UA or some other VO entity requiring services. The subject may define a set of policies that determine how its authorization is used, taking into account some privacy rules as well.

Resource*:* A component of the system that provides or hosts services and may enforce access to these services based on a set of rules and policies defined by entities that are authoritative for the particular resource. Access to resources may be enforced by a Resource itself or by some entity (a policy enforcement point or gateway) that is located between a resource and the requestor and thus protecting the resource from being accessed in an unauthorized fashion. Typical resources at Grid VO level might be Service Agents and other BVO/OpVO entities if their functionalities are required by a subject.

Authority*:* An administrative entity that is capable of and is authoritative for issuing, validating and revoking an electronic means of proof such that the named subject of the issued electronic means is authorized to exercise a certain right or assert a certain attribute. Right(s) may be implicitly or explicitly present in the electronic proof. A set of policies may determine how authorizations are issued, verified, etc. based on the contractual relationships the Authority has established.

The Authority so defined can be also specialized as a policy authority, attribute authority and an identity authority [6]. In Figure 43 at the VO Manager is assigned the role of Attribute Authority because it is able to assign the credential to a UA. This credential enables and constraints the UA to access only some services and functionalities. The VO Manager is also in charge of setting the service access policies by issuing authorization policies to the UA (or some of its dependent entities). Finally, the VO Manager is viewed to assume the identity authority role because it is able to issue some certificates (e.g. GT, etc.) enabling a group of VO services to communicate among them, as discussed in the previous section.

During the execution time some checks are performed by the SA in order to verify the claims derive from a secure and trusted entity and conform to the SA policy.

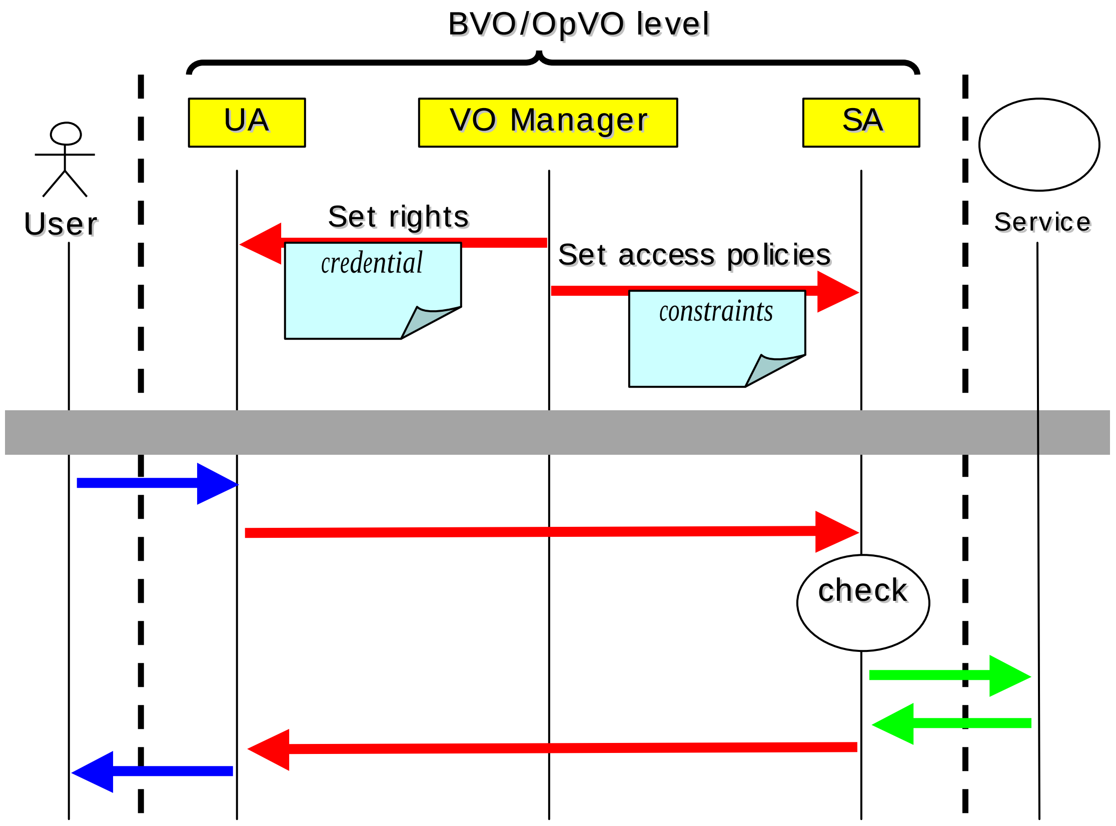


Figure 44, Authorization for VO

3.4.2.2. Use case view

3.4.2.2.1. Actors

In the framework of Akogrimo the following actors have been identified with respect to the security considerations that will be applied:

- **Users** which are allowed to use/consume the services provided.
- **Service Providers** which are as well certified resource/service providers in the framework of the VO.
- **VO Manager** allows the participation of people to be part of the VO and to assign the roles (enable the authorization for accessing services) in the administrative domain of the VO.
- **Certificate Authorities (SAML Authority)** providing tokens that are going to be used for the authentication in the framework of Akogrimo.

The system's infrastructure should provide distinct WS authentication and authorization capabilities (built on the same base of X.509 standard) which are used to identify persistent entities such as users and servers and to support the temporary delegation of privileges to other entities, respectively.

The security framework that will be used to manage WS-Resources will include:

- **Message-level Security mechanisms** which implement the WS-Security [45]standard and the WS-SecureConversation [47] specification to provide message protection for the system's SOAP messages.
- **Transport-level Security mechanisms** which use TLS [53] mechanisms; and
- an **Authorization Framework** that allows for a variety of authorization schemes, including a "grid-mapfile" access control list , an access control list defined by a service, a custom authorization handler, and access to an authorization service via the SAML protocol.

3.4.2.2.2. Use Cases

The following use cases have been identified for the cases of Akogrimo. However, the list might be enhanced in the next phase of the project by the time that the technologies will be clearly identified.

Use Case	**VO Registration**
Description	• A **user** (alternatively service provider) requests from the **VO Manager** to join the organization
Preconditions	N/A
Input	The **user** provides its personal data and preferences (determining which services and resources of the virtual organization he wants to be authorized to access).
Effects – Output	The user becomes a member of the virtual organization and gets a (unique) Id and assigned role.
Actor	Users, VO Managers.

Table 23 Use case "Join VO"

Use Case	**Request for a user/SP token**
Description	A **user** (or SP) subscribed in the VO requires a certificate for its authentication. This could happen either if the user has newly joined in the VO or if his previous certificate has expired.
Preconditions	The user requesting the certificate has requested to join the VO.
Input	The identifier of the user in this VO.
Effects – Output	A new certificate (SAML token) is generated and sent to the user.
Actors	Users, Certificate Authorities, VO Manager.

Table 24 Use case "Request for a user/SP token"

Use Case	**Authorization change request**
Description	A **user** asks the **VO Manager** to change the set of services and resources of the virtual organization that is authorized to access.

Preconditions	The user has a valid certificate and a concrete role (authorization scheme) in the VO
Input	The user's id and new role.
Effects – Output	The user is notified by the virtual organization manager whether its request is fulfilled. A new role (authorization) is valid for the user.
Actors	Users, VO Managers.

Table 25 Use case "Authorization change request"

Use Case	**Task submission**
Description	A user submits a task to the Akogrimo system.
Preconditions	The user submitting the task has been subscribed to the service requested which means he has a valid role and a token for using the service.
Input	The new task.
Effects – Output	The system undertakes the requested task and sends it for execution in the SP.
Actors	Users, SPs.

Table 26 Use case "Task submission"

Use Case	**Receive result**
Description	A user receives the results of a submitted task.
Preconditions	The user has submitted a task requesting services in which he has subscribed.
Input	A notification from the SP that the task has finished.
Effects – Output	The results of the submitted task are returned to the user.
Actors	Users, SPs.

Table 27 Use case "Receive result"

3.4.2.3. Conceptual architecture description

As discussed in previous sections there is a VO hierarchy present within Akogrimo, this is illustrated with the BVO's relationship with the OpVO's and can also be seen in the relationship between the core VO services and the VO Manager. It could be argued that this would make the implementation of a security policy easier to visualise.

However the distributed nature of the services in the Akogrimo Grid makes it important to also consider security at the end nodes of the Akogrimo Grid. These end points can be seen as the services called on in the OpVO. Not only does the BVO have to implement adequate security infrastructure for these services and act as a central policy point for security within Akogrimo, but these services have to provide and tailor their security infrastructure so it can work securely in a bi-directional fashion with the BVO of Akogrimo. The main elements for the Authentication process can be seen in the diagram below.

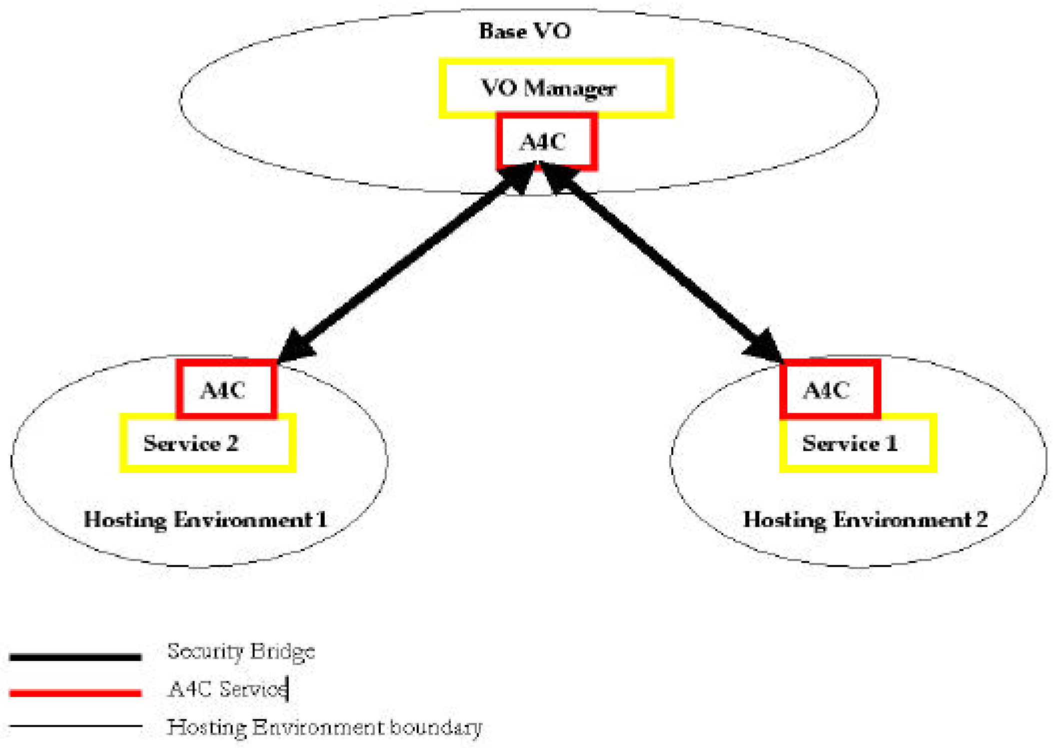


Figure 45 VO Authentication

Authentication in Akogrimo Virtual Organization

In order to connect / register or request a service from a VO a service has to be authenticated. Within Akogrimo this Authentication will be conducted via a network of A4C servers. A trust relationship between the SP domains and the domain of the BVO will be established by the VO Owner before or during the operation of the BVO.

The repository of information stored in the VO Manager will also include a repository of trusted domains which external services can be activated from. This repository then will link to an A4C server on the external domain to authenticate the service. See diagram in Figure 46.

In the figure below the main interaction between service and VO manager are depicted in order to identify the data flow exchange during the authentication process. The authentication ensures that a user is who he or she claims to be enabling them to “bypass” the identity recognition subsequently.

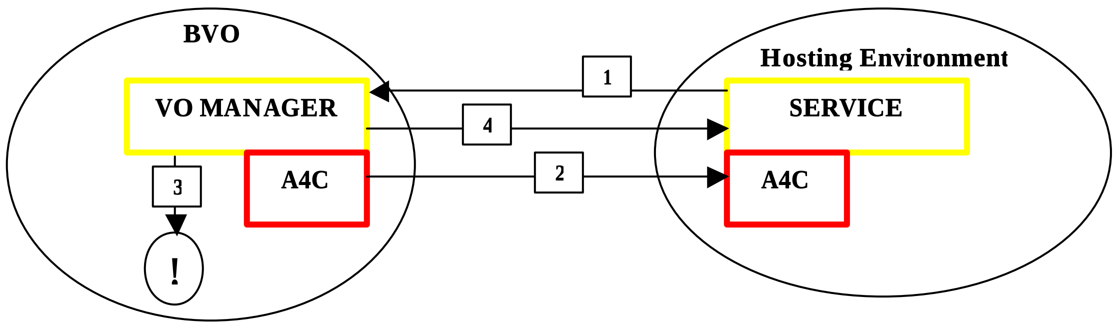


Figure 46 Authentication using A4C

1. Service sends credentials and the requests actions to Base VO.
2. Base VO sends credentials to appropriate A4C server for authentication.
3. Service's identity checked against the VO Participant Registry
4. Results of Authentication are sent back to the Service.

Once authenticated the service becomes part of the Base VO and is given a token so it can be passed down to an Operative VO for execution. The A4C hence becomes a fundamental component in the Akogrimo context for the authentication process, since it allows checking the identity of the entity (service or user) as a VHE member, before checking it as a VO member. After the Authentication phase the user/service will get credentials enabling him to operate within the Base/Operative VO without passing for A4Cs components in future requests.

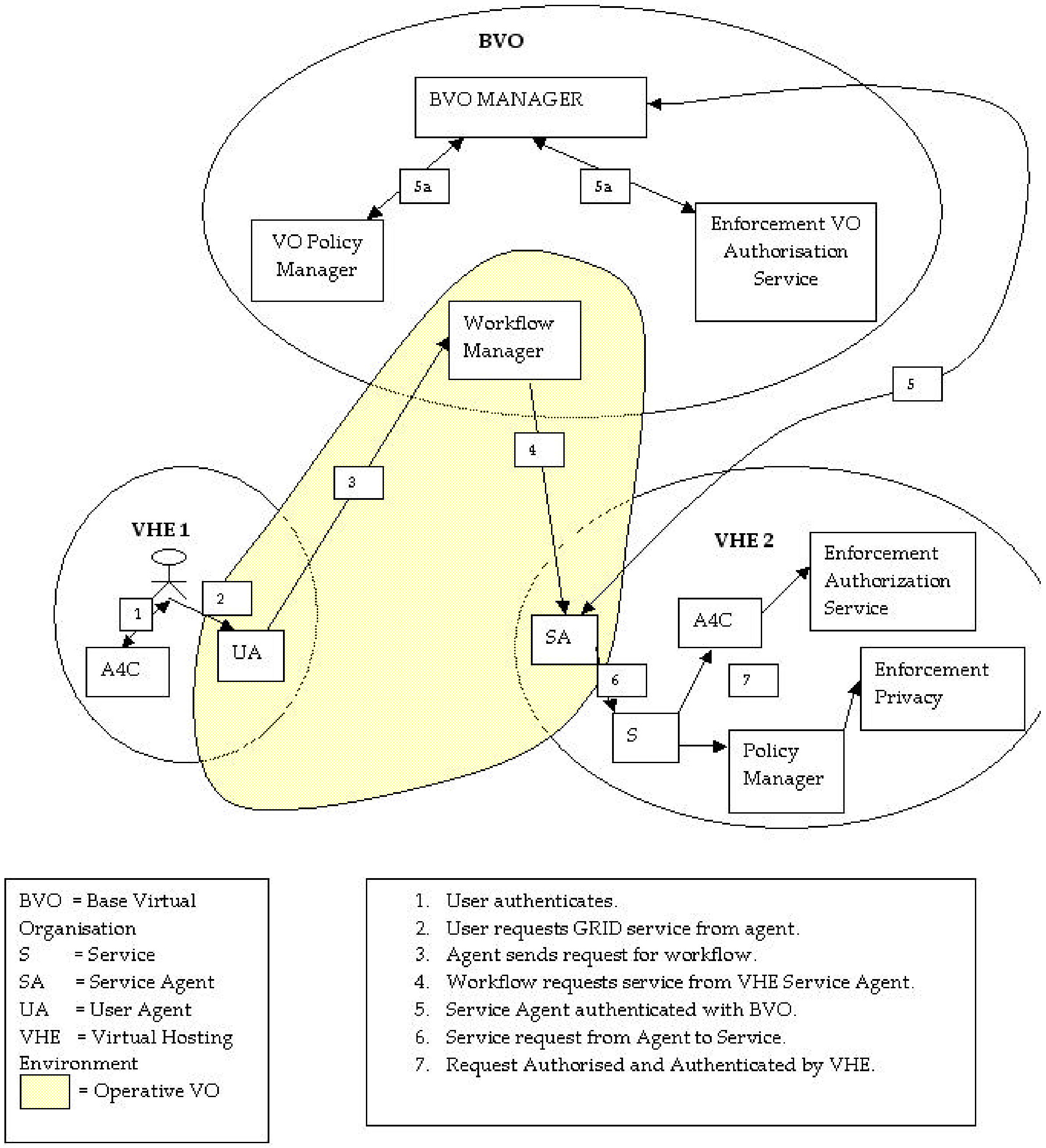
Finally, within Akogrimo all security tokens and messaging will be conducted using WS secure tunnelling.

Authorization in Akogrimo Virtual Organization

Authorization is the next step from Authentication, which as discussed using A4C servers is a pre-requisite. At VO layer, Authentication is the process of granting or denying credentials to access Grid resources. This second stage allows the user access to various resources based on their credentials.

After the Authentication credentials are received, at the point of policy for the application (in our case VO Manager) a decision is made in relation to Authorization. This decision can be seen as the enforcement process of the rules of policy to make an authorisation decision. In general, we have an Authorization process for both the VO level and VHE level but with different goals. The Authorization enforcement points shown in Figure 47 represents the Authorization process of a user request attaining a service in a VHE.

At this level the check can be related to rules defined by VHE administrator, for instance, to constrain the foreign requests. Instead, at VO level the authorization checking are related to more business/application specific rules, for instance in the workflow context if a doctor wants to access patient data (and hence to the data managing service), his request is checked in order to evaluate his role and what he can do on the requested service.



BVO	= Base Virtual Organisation
S	= Service
SA	= Service Agent
UA	= User Agent
VHE	= Virtual Hosting Environment
🟨	= Operative VO

1. User authenticates.
2. User requests GRID service from agent.
3. Agent sends request for workflow.
4. Workflow requests service from VHE Service Agent.
5. Service Agent authenticated with BVO.
6. Service request from Agent to Service.
7. Request Authorised and Authenticated by VHE.

Figure 47: Security Policy Enforcement Points in an A4C environment

As mentioned within the Akogrimo BVO, the authorization process would always be preset at the VO management level. The VO Manager has the power to create and destroy OpVO's and therefore the tasks being run in the VO. It is therefore at this point where enforcement is best applied.

In converse to the model, the enforcement of Authorization applied by the VHE to requests received from the BVO or OpVO would also be applied at an appropriate security management level (as in Figure 47).

Once Authentication is completed within the BVO a token is generated and associated with the task requestor which is used to bind with its OpVOs. As the OpVOs are created services from

VHEs are reserved to populate the OpVOs, the VHEs authorise this action based on the security credentials provided via the token given to the task requestor. . Within Akogrimo all security tokens and messaging will be conducted using WS secure tunnelling.

Internally the VO, the BVO Manager will grant security tokens for the use of services and UA/SA in order to maintain workflows and movement of data through the VO without the need to authenticate and in some cases to re-authorise (for instance, when the requests are similar).

SA and UAs.

SA and UA are essentially wrappers exposed as WS-Resources that enable services and users to interact within Akogrimo VO. In order to maintain security within Akogrimo the communication and also the wrappers need to be secured and that could be done by taking advantage of both binding/network secure communication protocols and policies defined within the VO.

It is envisaged within Akogrimo that the communication between the VO's and services will be via WS-SecureConversation [47]. This technology will ensure that the information passed between nodes (like A4C credentials) will be encrypted from end to end. Furthermore all security tokens and messaging will be conducted using WS secure tunnelling.

Once authenticated, the SA and UAs will receive tokens that they can use to interact with the VO's, without the need for further authentication. This is the same concept as the tokens received within the BVO as discussed in the authorization process above.

This use of Base VO authenticated tokens outside the immediate remit of the BVO and BVO owner could pose a potential chance of misuse. In order to combat this, the build up of agents will have to be designed in relation to specific security constraints and policy provided by the BVO owner.

Security Log

Within the whole operation of the VO's, logging will play an essential role in the security of the system. This is particularly true in the initial testing of the system. There is therefore an immediate requirement within the system for a logging tool for both security issues in the system and general behaviour. Essentially in the early days of development of the system everything should be logged.

David Chadwick of Kent University [54] has developed a secure logging tool designed for Grid applications which is capable of this job, and it is therefore suggested that this tool be investigated for use in Akogrimo.

Security Bridge

These elements in the architecture have been touched on before in relation to A4C and specifically relate to both trust and secure communication. Bridges will exist between security domains with the security policy within those domains being determined by the security root, in the BVO case this is the BVO Manager. The bridges are essentially the term given to the secure channels that will be used to pass data between services within VO's and also credentials between services without needing to know the different derivation nature of these credentials.

As discussed, this network of bridges will be built on trust relationships among the different administration domains and by taking advantage of A4C system, as well. Encryption will be provided by WS secure conversation maybe using XML digital signature and XML digital encryption to encrypt SAML tokens.

Policy Service / Security Designer.

These elements of the security phase can essentially be seen as provided by the security policy in force within the VO's. The security designer would be a tool in which security policy can be changed and developed.

3.4.2.4. Logical architecture description

The logical architecture described below aims to define a general overview of functionalities and interfaces that the security components should provide.

VO Authentication Service

The VO Authentication service will be present at VO level and also within the hosting environments that external services are picked from to become part of the OpVO's.

- **request** (*serviceID, Credentials*)

At the beginning of a business process within Akogrimo a request will be received at the VO manager that can be linked to a specific workflow. This request will be linked to the workflow ID (service ID) and also will be authenticated via credentials provided by the service sending the request.

VO Authorization Enforcement

- *reject* **Credentials**(*ErrorMessage*)

If A4C fails to authenticate user, the enforcement VO authorisation service rejects the request for service.

SAs and UAs

- **Authenticate**(*Credentials*)

The Agent should be able to authenticate its corresponding user/service and perform some initialization process for enabling the requestor's further requests.

- **RequestService**(*requestorCredentials, ServiceID,requestedFunctionality*)

The Agent must be able to send a user/service's request message to a service.

- **ReceiveToken** (*Token*)

Agent must be able to receive and store tokens.

- *Credentials* **request** (*serviceID*)

If an external service requests a service from the agents it must be able to authenticate the requesting body.

- **transmit data** (*UserID, TargetID,Data*)

Once a workflow is in progress the agent must be able to transmit data.

- **StopService**(*ServiceID*)

In the case of a user / workflow ID loosing its authorization permissions or completing its task the agent must be able to stop the service from running.

Security Log.

VO-ID **Record Events** (time/date, event)

Must be able to record the activity in a VO at a suitable abstract level.

VO-ID **GetActivity** (argument)

Must be open to interrogation

Security Bridge.

The Security Bridge need arises when we have user and resources located in different administration domain. We need interoperability cross-domain that takes into account the different mechanisms and credentials:

- X509 V.S. Kerberos
- X509 V.S. X.509 in different domain
- X509 attribute certificates V.S. SAML assertions

The Security Bridge is related to credential/identity conversion and mapping enabling the seamless single sign-on and federated session management.

Credential **convert**(*credential, targetType)*

Converts the credential of specified type into the desiderate type target.

Id **mapping**(*Id, targetID*)

Maps the id from one type to other type.

Security Designer.

Operating from a base level of security there must be a system or tool in place where the user can make simple changes to the VO's security policy.

block (*component, howlong*)

Must be able to create access control based on blocking of VO's or services.

Restrict (*component, data*)

Must be able to restrict types of data present on specific VO's or services.

Grant (accessType, targetComponent, sourceID)

It allows defining some bind policies for an entity (e.g. service, user) on accessing a target service.

Trust (VO-ID, VHE-ID)

Establish basic trust relationships in the form of mappings between VO's and service environments.

3.4.2.5. Involved technologies

On the base of Liberty Alliance Project results [42] we are going to develop our security system by taking advantage of SAML standard [46] and its possible application for WS-Resources. By means of SAML support provided at Network layer, each entity involved at Grid Application Support Layer level is identified and the message interchange between these entities will be protect by combining SAML with network and binding security technologies and specifications such as XML digital signature, XML encryption, WS-Security, etc.

3.4.2.6. Configuration and Deployment

This section will be updated during implementation phase, because at the moment there aren't enough details to provide inputs on it.

3.5. Application Specific Services

3.5.1. Overview

3.5.1.1. Objectives

The application specific services section will describe the envisioned working of architectural middleware components in the context of an eHealth environment. It is intended to highlight and detail important aspects of the components from all levels.

3.5.1.2. Example eHealth Environment

In the following we describe the organizational background of the eHealth scenarios we will detail later. The presented organizational setup is just one simplified example and does not cover many business details as the focus is on the architecture. It is intended to provide a rough idea of who the actors are and what their relation is.

The diagram in Figure 48 shows the basic structure of a regional health network. The yellow boxes are the main organizational actors. The health network consists of hospitals, medical specialists, general practitioners, an emergency dispatch centre and possibly other members. In order to build an IT infrastructure for eHealth services the regional health network cooperates with a network operator (NO) and an IT specialist, the health service provider (HSP). Together they form a virtual organization (VO). In our example, the network operator controls the network infrastructure and also the IT infrastructure of the VO. The health service provider is a software and consulting specialist in the eHealth domain. Regarding the IT infrastructure, the network operator is responsible for the generic part, i.e. the part that does not have any domain specific elements, and the health service provider implements and adopts all the domain specific services. Among the responsibilities of the HSP are:

- Provide mobile equipment, in our case only wearable ECG devices.
- Integrate the IT infrastructure of the involved medical institutions and practitioners
- Integrate external eHealth information systems like patient record databases
- Integrate external information systems like geographical information systems. This is used to find best way for the ambulance to the patient, or the hospital that is closest to the emergency location.
- Implement eHealth domain specific grid services like a medical analysis service
- Define the business process in cooperation with the regional health network and the network operator.
- Define the workflow templates that make up the business process.
- Define policies in compliance with the regional health network and the network operator.
- Define SLA templates.

Network Operator QoS Bundles are adjusted to the needs of the VO.

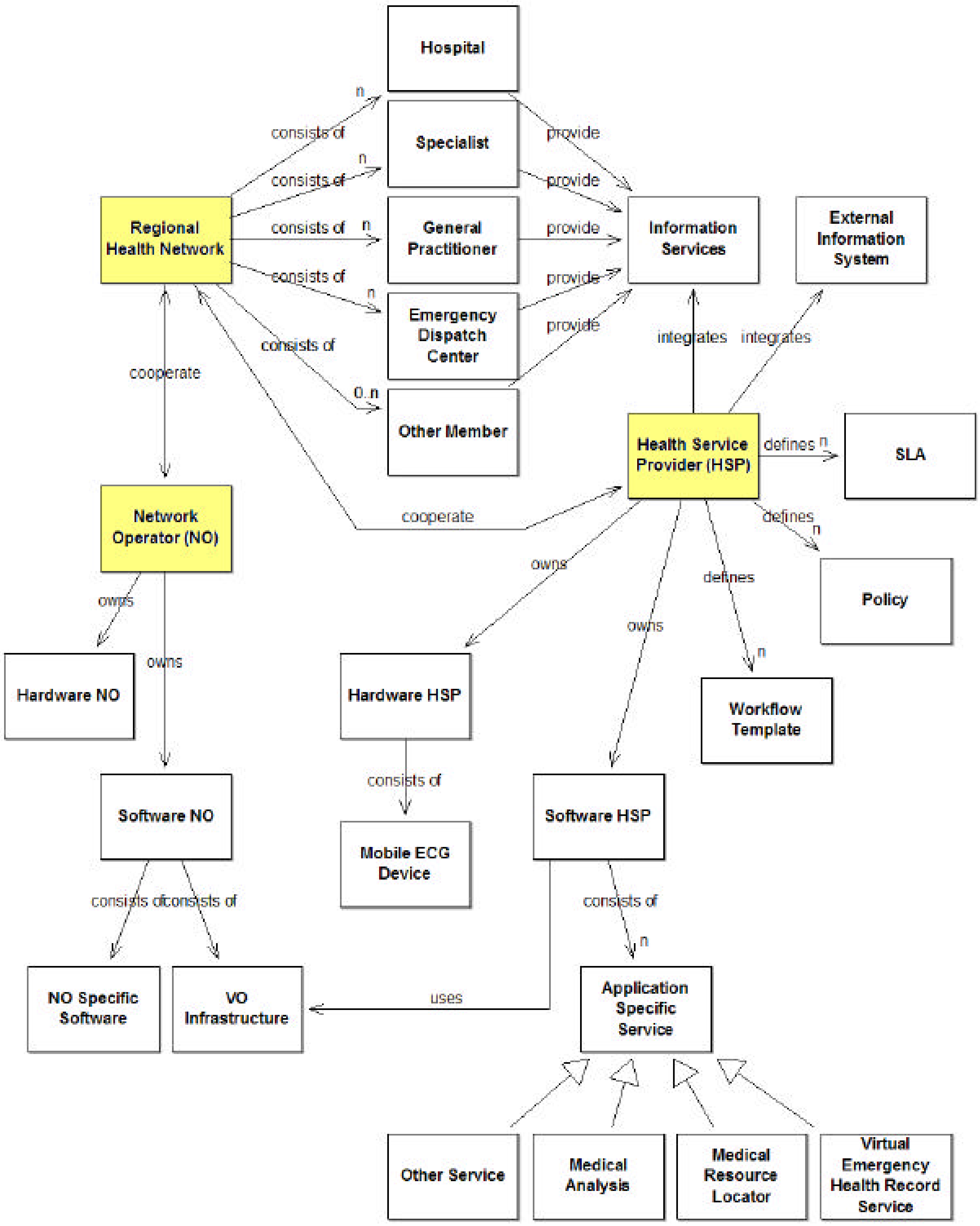


Figure 48 eHealth Scenario Organizational Overview

3.5.2. Interface Descriptions

In this section we will describe the interfaces of the grid services that are implemented by the health service provider (HSP). The interfaces give a rough idea about the application logic that is provided by the services. A detailed description will follow in the service descriptions section 3.5.3.

3.5.2.1. General Purpose Interfaces

3.5.2.1.1. IAkogrimoService

We assume the existence of an interface definition, the IAkogrimoService interface, that ensures the interoperability of services. IAkogrimoService contains the functionality that is common to all web-services that are to be used in the Akogrimo environment. Common functionalities are e.g. policy distribution and querying, security and SLA management support. Especially the BP Enactor component would depend on such an interface for service invocation and meta-data distribution and collection.

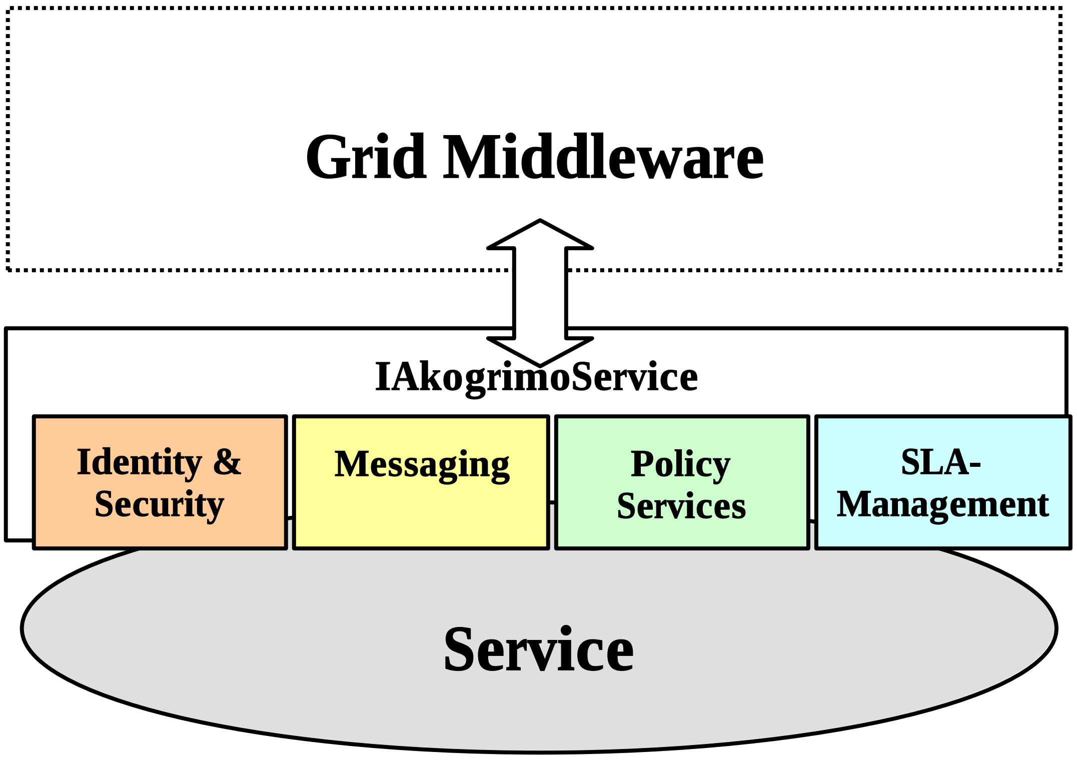


Figure 49 IAkogrimoService

Interface:	IAkogrimoService
Description:	The interface provides general purpose functionality for: • Messaging • Identity and Security • Policy distribution and querying • SLA Management

3.5.2.1.2. *IRegisterService*

It is assumed that the Service Discovery Server (SDS) provides the IRegisterService interface to allow services to be registered along with their meta-data.

Interface:	IRegisterService
Description:	Provides functionality to register a service plus its meta-data with the Service Discovery Server (SDS).

3.5.2.1.3. *IContextProvider*

We consider the context information to be provided via the IContextProvider interface. This is a functional assumption and may be replaced by a general publish-subscribe mechanism in the real implementation.

Interface:	IContextProvider
Description:	Used to retrieve context information about user from the Context Manager.

3.5.2.2. *eHealth Specific Interfaces*

3.5.2.2.1. *IMedicalResourceInformation*

Interface:	IMedicalResourceInformation
Description:	• Allows to query the geographical location of the medical resource • Allows to query up-to-date information about the medical resource. E.g. for a hospital the current available personnel with specific skills could be requested.

3.5.2.2.2. *IPatientInformation*

Interface:	IPatientInformation
Description:	• Administrative data of a patient can be requested through this interface.

3.5.2.2.3. *IECG_Data_Source*

Interface:	IECG_Data_Source
Description:	• A service implementing this interface provides access to ECG data either as a stream of data or by sending the data in specified time intervals (e.g. once a minute). The endpoint reference of the data sink is a parameter of the service.

3.5.2.2.4. IElectronicHealthRecord

Interface:	IElectronicHealthRecord
Description:	• Allows to query and to retrieve electronic health records.

3.5.2.2.5. IPatientMonitoringRecord

Interface:	IPatientMonitoringRecord
Description:	• Allows to query and to retrieve monitoring information from a Medical Data Logger.

3.5.2.2.6. IVirtualEmergencyHealthRecord

Interface:	IVirtualEmergencyHealthRecord
Description:	• Allows to query and to retrieve a patient record that contains patient information relevant for the emergency case.

3.5.2.2.7. IAlertReceiver

Interface:	IAlertReceiver
Description:	• Receives alerts from various sources. Automatic action or human intervention can be triggered.

3.5.2.2.8. IEmergencyOperator

Interface:	IEmergencyOperator
Description:	• This interface is used to query the SIP identifier of an emergency operator that is assigned to a specific patient. A phone conversation between emergency operator and patient may be initiated by the Enactment Engine.

3.5.3. eHealth Specific Services

This section describes a selected set of application specific services from the eHealth domain. The described services are intended to fit into the general overview shown in 3.5.2.1 and to provide the application layer building blocks that the use case descriptions are based upon. The design decisions made in this section are intended to provide one example of how to define eHealth specific services.

3.5.3.1. Common Interfaces

The following interfaces are provided or consumed by all services. They won't be mentioned in each description of a service.

Provided Interfaces:	• IAkogrimoService
Consumed Interfaces:	• IRegisterService to register a service with the SDS

3.5.3.2. Medical Resource Service

Real world entities like hospitals, ambulance cars or helicopters are referred to as *medical resources*. A medical resource implements the IAkogrimoService interface and registers with the SDS. A medical resource provides a specific interface, the IMedicalResourceInformation interface, through which current up to date information can be requested.

Provided Interfaces:	• IMedicalResourceInformation
Consumed Interfaces:	

3.5.3.3. Medical Resource Locator Service

The Medical Resource Locator Service uses the SDS to find potentially useful medical services. Additional application specific logic allows the Medical Resource Locator Service to understand the information provided by the IMedicalResourceInformation interface and make a decision about suitability of a service depending on that information.

Provided Interfaces:	• IMedicalResourceLocator
Consumed Interfaces:	• IMedicalResourceInformation

3.5.3.4. Virtual Emergency Environment (VEE) Service

The Virtual Emergency Environment (VEE) provides information and coordination to the participants involved in the handling of the emergency case. Decisions that require human interaction are taken here. An alert signal from the Medical Analyzer service is sent to the Enactment Engine. The business process definition might specify forwarding the alert signal to the VEE service in order to get a decision on how to proceed from the emergency operator. The emergency operator might automatically be connected to the patient by phone in order to support his decision making.

Provided Interfaces:	• IVirtualEmergencyHealthRecord to provide emergency case related information that is collected from different data sources. • IAlertReceiver to provide a human decision • IEmergencyOperator to request the SIP URI of the assigned emergency operator
Consumed Interfaces:	

3.5.3.5. ECG Data Source Service

In our example the ECG device that actually measures the heart signals is connected to a PDA. The PDA implements the grid service that receives the endpoint of the data sink at start-up and sends the data to that endpoint periodically. The PDA has also the capability to provide information about its geographical location.

Provided Interfaces:	• IECG_Data_Source
Consumed Interfaces:	

3.5.3.6. Medical Data Logger Service

The medical data logger consumes ECG data and stores it in a Patient Monitoring Record.

Provided Interfaces:	• IPatientMonitoringRecord
Consumed Interfaces:	• IECG_Data_Source

3.5.3.7. Medical Data Analysis Service

The Medical Data Analysis service consumes ECG data stored in a Patient Monitoring Record.

Provided Interfaces:	• IMedicalDataAnalysis
Consumed Interfaces:	• IPatientMonitoringRecord to read monitoring data • IAlertReceiver to be able to send alerts

3.5.3.8. Electronic Health Record (EHR) Service

Information systems of hospitals and doctors' offices implement this service interface to enable querying for patient records.

Provided Interfaces:	IElectronicHealthRecord
Consumed Interfaces:	

3.5.3.9. Attending Physician's EHR Service

Electronic health records (EHRs) provided by the attending physician's Electronic Health Record service can include the ECG monitoring data. So when the Virtual Emergency Health Record is prepared by the VEHR service the ECG monitoring data will be incorporated via the patient record provided by the attending physician. Also the emergency health care provider will use a Medical Data Logger.

3.5.3.10. Virtual Emergency Health Record (VEHR) Service

The Virtual Emergency Health Record service aggregates patient records from various sources. These sources are represented as EHR services. The VEHR service performs an aggregation of the electronic health records.

Provided Interfaces:	IVirtualEmergencyHealthRecord
Consumed Interfaces:	IElectronicHealthRecord

3.5.4. Application Components

3.5.4.1. ECG Viewer

The ECG Viewer can visualise real time ECG data as well as ECG data stored in a Patient Monitoring Record.

Provided Interfaces:	
Consumed Interfaces:	• IECG_Data_Source • IPatientMonitoringRecord

3.5.4.2. Map Viewer

A Map Viewer visualises the location of patients and medical resources like hospitals or ambulance cars. Information about these entities can be requested via the graphical user interface of the Map Viewer. Attributes of the medical resources that are of special interest may be visualised graphically, so the Map View becomes an important instrument in supporting human decision taking.

Provided Interfaces:	
Consumed Interfaces:	• IMedicalResourceInformation • IContextProvider • IPatientInformation • IMapService – interface to an external geo-information service to generate maps showing the location of medical resources and patients Optionally additional information can be included: • ITrafficInformation – interface to an external traffic information service • INavigationService – interface to calculate optimal routes

3.5.5. Generic Application Components

3.5.5.1. *Membership Management Application*

As will be detailed in the scenario descriptions an application to include patients into an OpVO, or in general, users into a VO should exist in a real world scenario.

3.5.6. Scenario Descriptions

3.5.6.1. *Scenario Overview*

The description of the scenario background is intended to motivate a connection to a real world scenario. Some simplifications are due to technical restrictions like focusing on WLAN and not using other wireless technologies. Others are made for clarity of the examples, e.g. inter-VO relationships are only addressed briefly. Only one BVO and a single OpVO are considered. Figure 50 shows a sketch of the mobile actor, the patient, and the entities related to the mobile grid service, the ECG Data Source service.

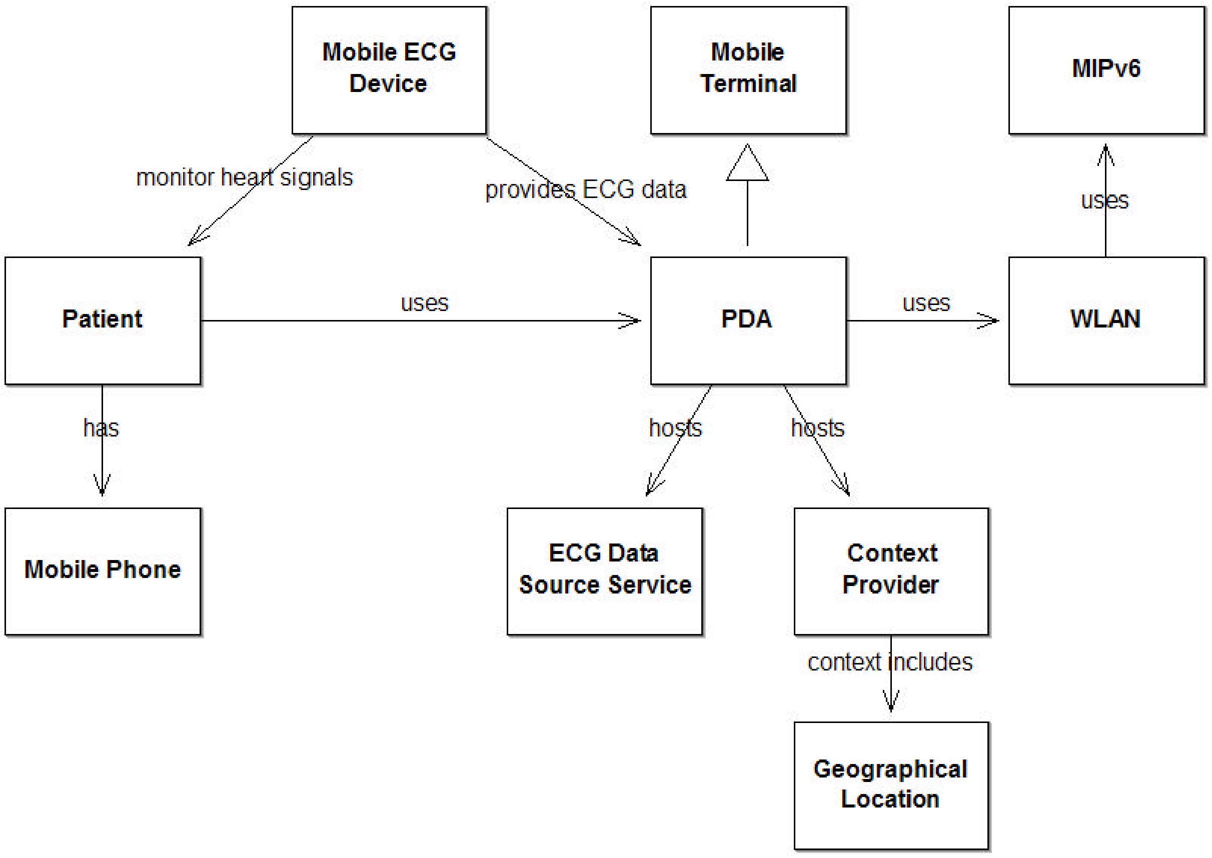


Figure 50 Patient Related Entities

A patient who's ECG is being monitored, carries a mobile ECG measuring device with him that is connected to a PDA. The PDA hosts the ECG Data Source Service that can send the ECG data via WLAN to a data sink, in our case the Medical Data Logger service. Throughout the scenarios only WLAN will be considered. At the network layer IPv6 and MIPv6 are used. The PDA is also capable of providing geographical location information.

3.5.6.2. Patient Monitoring Setup

Due to heart problems the patient's ECG is to be monitored for a specified period of time. In the doctor's office the setup of the monitoring service, the hardware equipment and the OpVO is done.

3.5.6.2.1. PDA Setup

The hardware equipment in our example scenario is provided by the HSP, it is not the patients own PDA that is used. For this reason an association of the PDA with a specific patient has to be created anew in each ECG monitoring case. There are several ways to "tell" the system that the grid services that are hosted by the PDA "belong" to the specific patient. In any case the meta-data of the gird services has to contain information that identifies the patient. E.g. the patient could become the service provider of the grid services. Or "patient identifier" field could be introduced as a specific entry of the meta-data that describes the service. Also the context providing client software has to be able to specify the identity of the user. In our example, identity information of the patient is transferred to the PDA during the setup. The identity information that is stored on the PDA can be used to spare the patient a manual login when the PDA is switched on.

Use Case	**PDA Setup**
Initiator	Doctor
Primary Actor	Doctor
Additional Actors	VO Manager, A4C, PDA, ECG Data Source service
Description	• The doctor requests an identifier for the patient from the VO Manager. The VO Manager contacts the A4C to retrieve the required token. • The doctor transfers the identifier to the PDA. The identifier is stored on the PDA for ease of use, so the patient does not have to do a manual login when he switches the PDA on. • The ECG Data Source service is configured with the identifier of the patient. This means that the identity of the user is used in the meta-data of the service to associate the service with the patient. In the service discovery process the meta-data information is used to include the right ECG Data Source service into the workflow. There may be other ECG Data Source services "provided" by other patients, so identifying the right patient is essential.
Pre-conditions	Doctor and patient are members of the BVO. The doctor can request a patient identifier. The doctor is allowed to access the PDA for setup purposes.

Post-conditions	The PDA “carries” the patient identity information, so the patient does not have to use a user name and password to identify himself. The identity information is used in the meta-data of the ECG Data Source service and in the context providing client software.

Table 28 Use case “PDA Setup”

After the PDA setup is done, the ECG Data Source service is published into the Service Discovery Server.

3.5.6.2.2. Creating the Patient Monitoring OpVO

Use Case	**OpVO Setup**
Initiator	Doctor
Primary Actor	Doctor, Workflow Manager
Additional Actors	OpVO setup application, VO Manager, OpVO Manager, OpVO Broker, Service Discovery Server, Workflow Registry
Description	• With the help of a OpVO setup application following steps are executed. • A workflow template is specified by the doctor. He may have to choose from a list of pre-selected workflow templates. E.g. ECG monitoring, blood preasure monitoring and others. Alternatively the workflow template may be discovered depending on the given parameters. • Parameters for the setup of the Patient Monitoring OpVO are specified with the OpVO setup application. These parameters are: • The human actors and their roles. Identifying the actors can be done by passing identifying data like name, date of birth, etc. to the VO Manager and receiving an identity token. Identifying data may also be taken from a patient identity card or in the case of the doctor, form an existing login. • The duration of the monitoring, which VEE should be contacted, phone number or SIP address of the patient, who is the “Attending Physician” (if not the doctor). • Also SLA parameters for the services that are to be used in the workflow can be specified. • The OpVO setup application pases the parameters to the Workflow Manager and request an OpVO to be created. OpVO creation is detailed in section 3.2.2.3.
Pre-conditions	Doctor and Patient are members of the BVO. The doctor has the right to request the creation of an OpVO. The appropriate ECG Data Source service has been registered with the Service Discovery Server, i.e. the PDA has been

	set up for the specific patient.
Post-conditions	The patient and doctor are member of the newly created OpVO. The patient can act in the role of “Monitored Patient”, i.e. he can use dedicated services of the Patient Monitoring OpVO, like sending an alert in case of emergency. The doctor can act in the role “Attending Physician”.

Table 29 Use case “OpVO Setup”

3.5.6.2.3. Viewing ECG Data

After the OpVO has been created the doctor starts the ECG Viewer application on his desktop PC. The ECG can also be view on the PDA of the patient.

Use Case	**Viewing ECG Data**
Initiator	Doctor
Primary Actor	Doctor, ECG Viewer application
Additional Actors	Medical Data Logger, Workflow Manager, Service Authentication Service.
Description	• The doctor starts the ECG Viewer application. • The viewer application uses the identity and OpVO membership information that is present on the desktop PC of the doctor. • With this information the application (via the User Agent) contacts the Workflow Manager to request a service that can provide the ECG data of the patient. • The ECG Viewer requests ECG data from the Medical Data Logger service. • The Service Authentication service of the hosting environment where the Data Logger is hosted verifies that the viewer application on behalf of the doctor (or patient) can access the ECG data.
Pre-conditions	The Patient Monitoring OpVO has been set up successfully.
Post-conditions	The working of the ECG measuring device has been verified by viewing the ECG data with the ECG Viewer application.

Table 30 Use case “Viewing ECG Data”

3.5.6.2.4. Hosting Environment Setup

The services provided by the HSP can be vitally important for the treatment of patients and handling of emergency situations. In order to ensure scalability, reliability and performance the HSP hosting environment uses grid middleware components as describe in D4.3.1 [8]. In Figure 51 we just show the Execution Manager (EMS) and the Data Manager. Monitoring of the computational and hardware resources is essential to counter service failures. Application specific services like the Medical Data Logger and the Medical Analysis services are deployed into the hosting environment of the HSP. Only a few interactions of the components are indicated in Figure 51. Multiple ongoing patient monitoring processes are indicated by the two mobile ECG devices sending data to two Medical Data Logger services. The HSP hosting environment most likely contains a variety of eHealth related services.

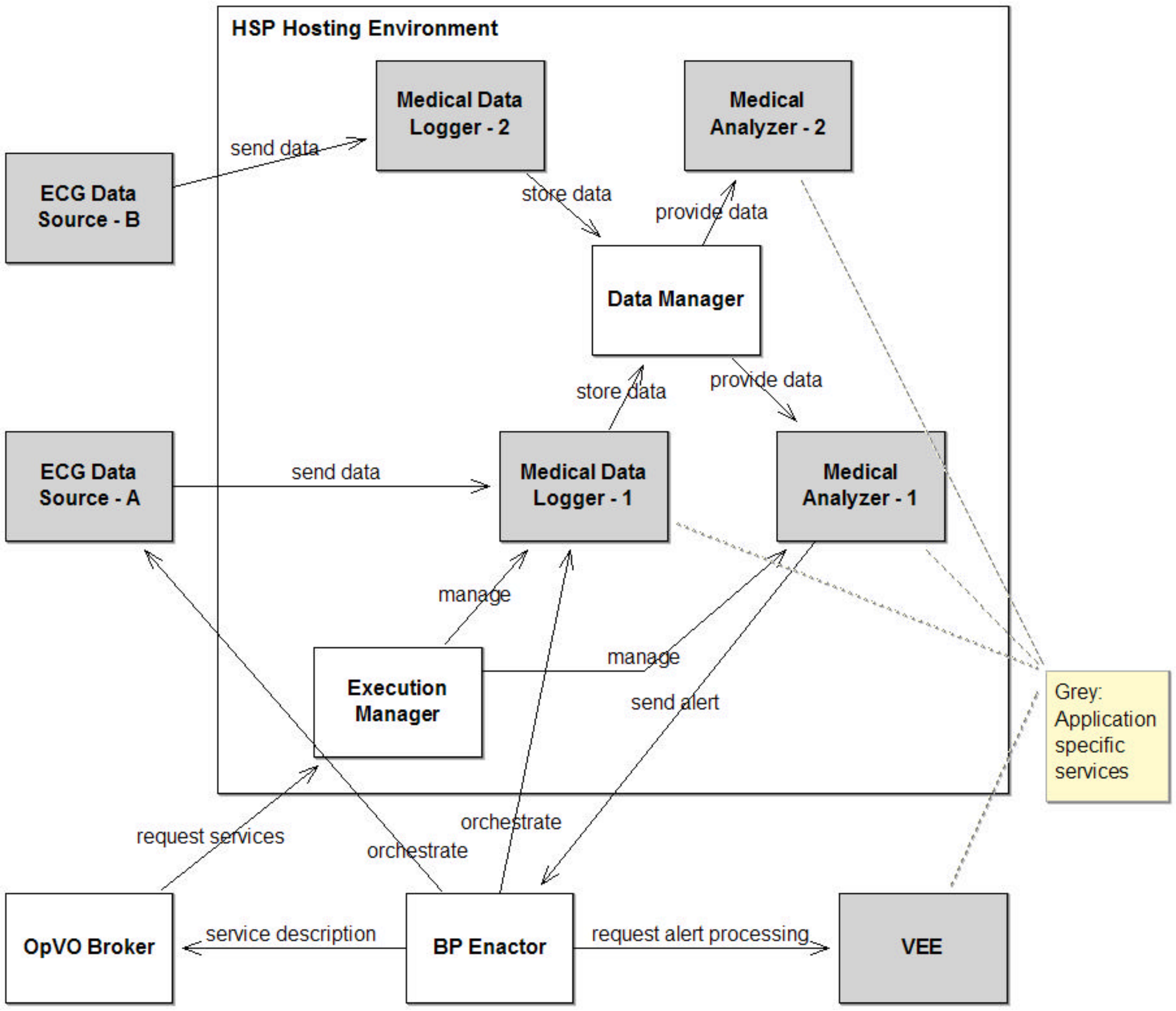


Figure 51 HSP Hosting Environment

The service descriptions of the services hosted by the HSP hosting environment are published into the Service Discovery Server of the VO. Services may be pre-allocated to specific VOs or may be allocated when needed. Interactions of the OpVO Broker and the EMS for service allocation are detailed in section 3.2.

3.5.6.3. *Emergency Response*

The ECG data of the patient is continuously logged and analysed. The Medical Data Analysis service has detected an anomaly and sends an alert message to the Enactment Engine.

Use Case	Emergency Response
Initiator	Enactment Engine
Primary Actor	Enactment Engine
Additional Actors	Medical Data Analysis service, VEE, VEE operator, patient, EHR service, SIP server Optional: Diagnosis Support service, medical expert
Description	• The Medical Analysis service has detected an anomaly in the ECG data of the patient • An alert message is sent to the enactment engine • The enactment engine requests the VEE operator to contact the patient in order to get additional information about the symptons. • After the VEE operator has signaled his availability, the Enactment Engine connects the VEE operator and the patient by phone. • The operator asks the patient for his symptoms. In this procedure the operator is guided by a Diagnosis Support service. • The operator can view the ECG data of the patient. • The operator may request a cardiologist to evaluate the anomaly of the ECG data. • An initial diagnosis of the heart problem is prepared and stored in the Electronic Health Record of the patient. • The operator decides whether the patient should be brought to a hospital or not.
Pre-conditions	The Patient monitoring process is ongoing.
Post-conditions	Information about the situation of the patient is available. A decision has been taken by the VEE operator whether the patient should be brought to a hospital or not.

Table 31 Use case "Emerging Response"

3.5.6.4. *Emergency Handling*

In case the VEE operator decides that the detected anomaly (see previous scenario) is severe, the emergency handling procedure is initiated. This emergency handling procedure might be executed within the Patient Monitoring OpVO or it might be realized by creating an Emergency Handling OpVO. We first describe the case where emergency handling is a sub-process of the existing Patient Monitoring OpVO. As emergency situations do not regularly occur during the ECG monitoring process, the necessary services would not be allocated up front, but would have to be included on demand into the existing OpVO. Existing identity information, role assignment, security and access policy settings would continue to subsist.

Use Case	Emergency Handling – As Sub-Process of Patient Monitoring
Initiator	VEE operator
Primary Actor	Enactment Engine, VEE operator
Additional Actors	Workflow Manager, OpVO Manager, OpVO Broker, Virtual Emergency Health Record (VEHR) service, several Electronic Health Record (EHR) services, Medial Data Logger, ECG Data Source service, Medical Resource Locator service, several Medical Resource services
Description	• The VEE operator has confirmed the execution of the emergency handling sub-process. The Workflow Manager has retrieved the associated workflow template from the Workflow Registry. • The Workflow Template or parts thereof are passed to the OpVO Broker to allocate the necessary services. • The services are configures with the necessary tokens and policies. Following actions are initiated in parallel by the Enactment Engine. Some of them may involve human actors: • Preparation of the Virtual Emergency Health Record (VEHR). This is done by the VEHR service. • Electronic Health Records (EHRs) from various sources, e.g. from hospitals, from the attending physician and other doctor's, are queried and combined into a Virtual Emergency Health Record. The VEHR includes information that is necessary to provide appropriate treatment in the current emergency situation. • Bringing the patient into a hospital • Find a suitable hospital (Medical Resource Locator service) o Send an ambulance o Prepare medical treatment • The VEE operator continues to talk to the patient. He get's feedback from the system about the actions that are initiated by the Enactment Engine. He may be asked to confirm certain decisions proposed automatically by the system. • Experts are consulted if needed
Pre-conditions	An emergency situation has been diagnosed. Information about the situation and location of the patient is available.
Post-conditions	Appropriate help is on the way. Detailed medical information about the patient is available.

Table 32 Use case "Emergency Handling – As Sub-Process of Patient Monitoring"

An alternative way to deal with the emergency situation is to create a new OpVO. In this case the basic OpVO setup steps have to be carried out. In the description of the creation of the Patient Monitoring OpVO it was assumed that the doctor uses an application to create the OpVO. Parameters like the duration of the ECG monitoring were entered manually. In the case of an emergency, time is precious and the OpVO creation should be automated as far as possible. A pre-selection of a suitable workflow for emergency handling could be done by the system. The final selection would be confirmed by the VEE operator.

Communication between the Patient Monitoring OpVO and the Emergency Handling OpVO would be achieved by means of the data that is stored about the patient. Usually this is enough and no direct interaction between participants or services of different OpVOs is necessary. In the patient monitoring case the ECG Data Source service that was created to be used in the Patient Monitoring OpVO provides information that is also needed in the Emergency Handling OpVO. So some sort of coordination between the OpVOs is necessary. It should be avoided that the ECG Data Source service stops operating before the ECG monitoring is transferred to the more sophisticated ECG equipment of the ambulance car.

3.5.6.5. Configuration and Deployment

Depending on their nature, application specific services can be deployed on any hosting environment that supports the implementation of web services. Configuration and deployment of a service includes the following phases:

- Deployment of the service and the service description
- Discovery of the service by a potential service consumer
- Negotiation of the terms of use

While the WSDL describes the functional behaviour of a service, the terms and conditions of use are expressed by policies and SLAs that are attached to the WSDL description. This information about the service is deployed into the Service Discovery Server. For a detailed description see [9] section 5.4. Also information about the service discovery process can be found there. The complexity of the negotiation of the terms of use depends on the service and its hosting environment. E.g. the ECG Data Source service that is hosted on a small mobile device like a PDA will either say that it is can provide ECG data or not. Services deployed into the hosting environment of the HSP will use dedicated negotiation functionality to agree upon the configuration of the SLA parameters.

Further update on this section could be done during implementation phase.

3.6. Legacy Application Integration

The goal of this section is to provide a connector that makes legacy applications (Mobile) Grid-aware and for this reason it needs a more detailed description of the testbeds that will evaluate Akogrimo Infrastructure and a final specification of Grid Application Support Services Layer. Therefore, if necessary, an update of this section will be provided in next deliverable (D4.4.2 Prototype Implementation of the Application Support Services Layer).

3.6.1. Overview

The grid environment can be treated as a federation of heterogeneous, distributed and dynamic resources which are seamlessly and transparently integrated across diverse virtual organizations and collaborate in order to deliver the desired quality of service [14]. Entities constituting the grid are not subject to centralized control and are coordinated by means of standard, open, general-purpose protocols and interfaces. The fundamental paradigms which lay the foundations of grid computing are resource virtualization and service orientation.

When combined, they yield a model in which widely varied software and hardware resources are uniformly abstracted into the high-level services designed to support interoperable machine-to-machine interaction conformant to the well-defined conventions. Services communicate by exchanging message and collectively comprise a distributed system referred to as service-oriented architecture (SOA).

In the context of grid environment, the legacy software can be considered as one that follows traditional process-based model of computation. This is in contrast to service-based processing, which is the characteristic of SOA. Both approaches differ considerably with respect to the offered level of abstraction and interoperability, as presented in Table 33.

Criterion	**Grid Services**	**Legacy Software**
Paradigm	Service-oriented	Process-oriented
Interface	Language-neutral (WSDL)	Language-dependent
Execution mode	System-neutral run-time environment (virtual machine)	System-dependent native operating-system process

Table 33: Comparative definition of legacy software

It is worth emphasizing that the foregoing classification disregards the programming language employed by implementation. A software component is considered to be a legacy one, when it is designated to be executed as a native operating system process. Thus, it is dependent on the underlying system API and architecture. Moreover, legacy libraries exhibit interfaces which are dependent on a specific programming language. The situation is totally different in case of grid services since they are run within the virtual machine provided by the hosting environment and their interfaces are language-neutral. Service-centric approach enhances flexibility and reduces maintenance expenditure thus lowering the total cost of ownership (TCO).

For the vision of the world wide grid to become reality, there is a need for adaptation of a large number of presently used legacy applications to SOA. One of the key challenges facing this process is a cost-effective migration strategy that has to be devised. Significant corporate investments in the existing high quality validated software need to be preserved during the transition to the new environment. Legacy systems are critical enterprise assets reflecting key business practices and knowledge acquired over the life of an organization. From economic perspective, it is unacceptable to discard fully operative software and re-design and re-implement it from scratch using grid-enabled technology. It is essential to ensure that legacy systems can be incorporated into the grid infrastructure in a smooth, incremental way which does not involve excessive development effort.

Another important issue is that in some cases it is not only expensive but also undesirable to port legacy software to a different technology. This is particularly true for computationally intensive applications (e.g. those using numerical libraries) that are optimized for certain native platforms. They could suffer from an intolerable degradation of efficiency when relocated into the runtime environment built on the top of a virtual machine.

For the above-mentioned reasons, it is clear that legacy software cannot be entirely substituted. Process-based computing will remain indispensable even in the face of emerging grid technology. In fact, both described models need to co-exist and seamlessly cooperate with each other. Therefore, it is necessary to design an architecture enabling for transparent bridging between legacy systems and grid services. And ideally, there should exist a framework supporting software developers in porting legacy codes to SOA.

3.6.2. Legacy System Modernization

In this chapter we familiarize ourselves with certain aspects of legacy software engineering. In particular, we should realize what kind of activity migration to a new technology is and how it relates to the whole process of system evolution. For this purpose we will now briefly describe the life cycle of a typical enterprise information system.

3.6.2.1. System Evolution

A number of activities account for the evolution of any software system. They can be roughly divided into three broad categories, i.e. system maintenance, system modernization, and system replacement [33].

Figure 52 illustrates how they relate to one another in the context of the system life cycle. The dashed line represents the growing business needs while the solid one reflects the provided functionality. As shown in the diagram, maintenance is the activity which usually suffices only for a short period of time. When the system becomes increasingly outdated, its functionality starts to fall behind the business needs. Under such circumstances, modernization effort is required. Finally, when the system can no longer be evolved, it must be replaced with a new one.

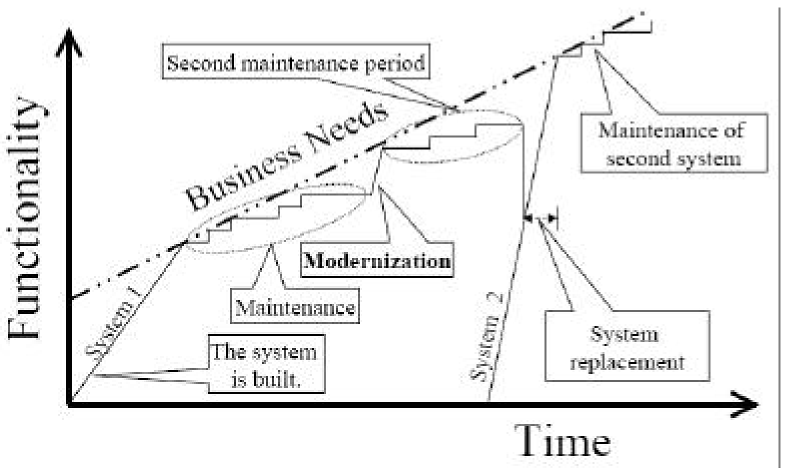


Figure 52: Enterprise information system life cycle

Maintenance is an incremental and iterative process which involves introducing minor changes to the system, like bug corrections or small functional enhancements. Architectural modifications are here out of the question. Maintenance is indispensable for the evolution of any system, but it possesses many limitations. First of all, the cumulative effect of numerous small changes is the erosion of the system conceptual integrity. Secondly, the source code continuously expands due to the fact that the unused modules are seldom systematically removed. Furthermore, the costs gradually increase since finding expertise in out-of-date technologies becomes more and more difficult in the course of time.

Replacement is appropriate for legacy systems that can not keep pace with business needs and for which modernization is not possible or cost-effective. It is normally employed in case of systems that are undocumented or not extensible. Obviously, replacement is connected with a number of risks that should be evaluated before this technique is decided upon. Most importantly, building a system from scratch is a very resource intensive task. In the majority of situations it may basically prove too expensive. What is more, there is no guarantee that the new system will be as robust or functional as the old one which is already thoroughly tested and well-tuned. Therefore, replacement may cause a period of degraded system functionality with respect to the business needs (as depicted in Figure 52).

3.6.2.2. System Modernization

Modernization is an activity which involves more extensive changes than maintenance (e.g. system restructuring, important functional enhancements), but conserves a significant portion of the existing software [17] [18]. It is employed when a legacy system requires more pervasive modifications than those possible during maintenance, yet it still presents the business value that must be preserved.

We can distinguish two basic classes of system modernization techniques:

- White-box modernization, which requires knowledge of the internals of a legacy system, and
- Black-box modernization, which requires knowledge only of the external interfaces.

3.6.2.2.1. White-box Modernization

White-box modernization requires an initial reverse engineering process to gain an understanding of the internal system operation. Components of the system and their relationships are identified, and a representation of the system at a higher level of abstraction is produced [21].

Program understanding is the principal form of reverse engineering used in white-box modernization. Program understanding involves modelling the domain, extracting information from the code using appropriate extraction mechanisms, and creating abstractions that help in the understanding of the underlying system structure [22]. Analyzing and understanding old code is a difficult task because with time, every system collapses under its own complexity [23]. Although some advances have been made in program understanding, it is still a risky and work-intensive task [24] [25].

After the code is analyzed and understood, white-box modernization often includes some system or code restructuring. Software restructuring can be defined as “the transformation from one representation form to another at the same relative abstraction level, while preserving the subject system’s external behaviour (functionality and semantics)” [21]. This transformation is typically used to augment some quality attribute of the system like maintainability or performance. Program (or code) slicing is a particularly popular technique of software restructuring. A description of a semi-automatic restructuring technique to improve cohesion of legacy procedures can be found in [26].

3.6.2.2.2. Black-box Modernization

Black-box modernization involves examining the inputs and outputs of a legacy system within an operating context to gain an understanding of the system interfaces. Although acquiring an understanding of a system interface is not an easy task, it does not reach the degree of difficulty associated with white-box modernization.

Black-box modernization is often based on wrapping. Wrapping consists of surrounding the legacy system with a software layer that hides the unwanted complexity of the old system and exports a modern interface. Wrapping is used to remove mismatches between the interface exported by a software artefact and the interfaces required by current integration practices [27] [28]. Ideally, wrapping is a “black box” reengineering task in which the legacy interface is analyzed and the legacy system internals are ignored. Unfortunately, this solution is not always practical, and often requires an understanding of software modules’ internals using white-box techniques [29].

3.6.3. State of the Art

Many large industrial and scientific applications, which are available today, are well written before Grid computing or SOA appeared. In order to ease industries in adopting the Grid technology, these legacy code programs have to be integrated into service-oriented Grid architectures with the smallest possible effort without degrading their performance. This chapter summarizes several research efforts on black-box modernization aiming at transforming legacy codes into a Grid service.

3.6.3.1. Transformation into Web Services

In [20] a conceptual architecture for adaptation of legacy applications to web services technology is presented. As illustrated in Figure 53, three components constitute the essence of the proposed solution: web service container, web service adapters, and back-end legacy servers running legacy applications. Each web service is equipped with an adapter which is responsible for connecting to the appropriate backend server(s) on behalf of the client. The role of adapters is to hide the complexity of calling backend functions which typically involves the communication through proprietary protocols.

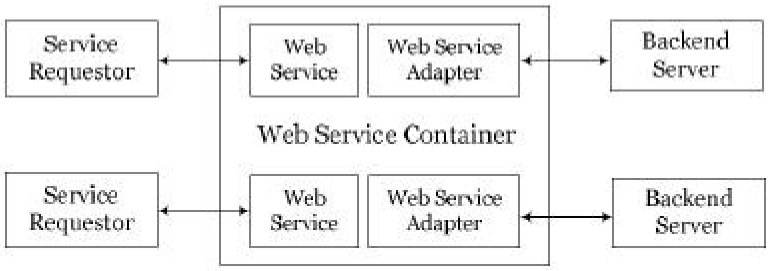


Figure 53: Architecture employing web service adapters

There are some disadvantages on this approach, i.e.:

1. Insecurity

   Each backend server demands an open port on which it can listen to the client requests. In complex installations this may introduce serious security vulnerabilities.

2. Inflexibility.

   Service adapters have to be configured statically with regard to the locations of the corresponding backend servers so that the communication can be established. In consequence the infrastructure cannot tolerate process migration between computing nodes and thereby lacks such features like automatic load-balancing and fail-over.

3. Lifetime management is not supported

   Instances of web services cannot be created and destroyed on demand. Hence, potentially dynamic and transient nature of grid services are not fulfilled using this technique.

3.6.3.2. Wrapping Legacy Codes

Currently, there are two techniques which are used to wrap the legacy codes, i.e. object-oriented wrapping and component wrapping. These techniques are explained briefly in section 3.2.1 and section 3.2.2 respectively.

3.6.3.2.1. Object-oriented Wrapping

Objects have been used to implement complex software systems successfully. Object-oriented systems can be designed and implemented in a way that closely resembles the business processes they model [23]. Additionally, the use of abstraction, encapsulation, inheritance, and other object orientation (OO) techniques make object-oriented systems easier to understand.

To support object distribution, we need a more powerful means of communication than that provided by normal inter-object communication mechanisms. Distributed object technology (DOT) is the combination of distributed technology with OO [27]. In effect, DOT extends object technology to the net-centric information systems of modern enterprises by using object middleware. The most prevalent object middleware is the Common Object Request Broker Architecture (CORBA) from OMG, with its platform-neutral object specification language, robust remote method calls, interoperable protocols, and rich set of services.

The conceptual model of object-oriented wrapping is deceptively simple: individual applications are represented as objects; common services are represented as objects; and business data is represented as objects. In reality, object-oriented wrapping is far from simple and involves several tasks including code analysis, decomposition, and abstraction of the OO model. Several difficulties exist during legacy system wrapping, i.e.:

1. The definition of appropriate object-level interfaces

   Translating the monolithic and plain semantics of the often procedural legacy system to the richly hierarchic and structured semantics of an object-oriented system can be a difficult task. A good knowledge of the domain can greatly help in the translation. For example, Stets describes an experience in which the Win32 API is translated into objects [34]. To create meaningful objects, domain-specific knowledge of the structure of an operating system is used. Unfortunately, developers who wrap a system rarely possess such deep domain knowledge. Some techniques have been developed to perform the legacy to OO mapping more automatically. For example, in one such method, every coarse-grained persistent item is mapped into an object, and services are assigned to objects with an algorithm that minimizes coupling [35]. Although these and other techniques are useful in extracting objects from legacy systems, the mapping problem is far from being solved.

2. The need for integrated infrastructure services

   The second challenge of translating an OO system using DOT is integrating infrastructure services in the OO system. Almost any DOT middleware provides some set of services such as security, transactions or persistence. However, it is often the case that to get the expected level of services the developer must integrate two or more of these middleware solutions. This integration is at least problematic due to unexpected interactions and incompatibilities between theoretically compatible products [36]. To address this need for integrated services, the industry has developed what is commonly known as application servers (AKA server-side component frameworks). An application server is a product that integrates a set of services and defines a component model. Components, which are developed conforming to this model, are able to leverage all the services provided by the application server.

A proposal of a semi-automatic technique for conversion of legacy C interfaces to their Java equivalents is presented in [19]. Two auxiliary tools, i.e. JACAW (JAva-C Automatic Wrapper) and MEDLI (MEdiation of Data and Legacy code Interface), are introduced which allow for code wrapping and data mapping, respectively. Although separate, they are intended to work in collaboration. The JACAW tool exploits JNI (Java Native Interface) in order to bridge between C and Java code. It provides facilities which enable to generate Java interfaces from the given C header files. MEDLI allows for conversion of the obtained Java code to components that can be then deployed in the Triana-based grid environment. Triana is a workflow composer which can be used to build complex grid services from pluggable components.

This approach is connected with three major limitations:

1. Most importantly, it is restricted to configurations in which legacy applications are located on the same machine as the service container. It stems from the fact that the communication is realized locally by means of JNI. In a vast array of situations this inflexibility can prove unacceptable.
2. It is constrained to Java language as far as hosting environment is concerned. Along with the emergence of different non-Java container implementations, this shortcoming may become problematic.
3. The solution is unsafe due to the fact that legacy code is executed in the same operating system process as the container's JVM. Bugs present in the legacy library can manifest themselves by crashing the whole runtime environment. In other words, there is a very poor isolation and safety provided.

3.6.3.2.2. Component Wrapping

Component wrapping is very similar to OO wrapping, but components, in contrast with objects, must conform to a component model. This constraint enables the component framework to provide the component with quality services. The component model market consists initially of numerous proprietary and mutually incompatible solutions. Fortunately, this situation is improving with the emergence of a small set of standard enterprise component models and products. Of these standards, three are of particular consequence:

- Distributed internet Architecture™ (DNA) is the Microsoft solution and a de facto standard because of Microsoft's weight in the industry.
- The CORBA 3 Component Model is the OMG approach for the enterprise that promises an enterprise component model for CORBA.
- Enterprise JavaBeans (EJB) is the Sun Microsystems solution for Java server-side computing.

According to press releases from the OMG, it is likely that the CORBA 3 Component Model and EJB will merge into a single standard leaving only two dominant players. The following discussion uses EJB to illustrate component wrapping, but does not necessarily mean that EJB is superior to other competing models or that these solutions cannot be implemented with other products.

Components in EJB are known as Enterprise JavaBeans (also referred as beans). Each bean encapsulates a piece of business logic. Enterprise JavaBeans are deployed within an application server, or EJB server, that provides the runtime environment for the bean and manages common services such as security, transactions, state management, resource pooling, distributed naming, automatic persistence, and remote invocation. This allows the EJB developer to focus on the business problem to be solved. An Enterprise JavaBeans is, in essence, a transactional and secure remote method invocation (RMI) or CORBA object, in which some of whose runtime properties are specified at deployment using special files called “deployment descriptors”.

The first step in wrapping a legacy system using EJB is to separate the interface of the legacy system into modules consisting of logical units - shown in Figure 54 as Function 1 and Function 2. The degree of difficulty in dividing the legacy system into discrete functions will vary depending on the degree to which the separation was defined in the legacy system interfaces and whether new interfaces must be built. Although a black-box approach is preferable, poor documentation and a cryptic legacy system interface may make it necessary to peer into the black-box to better understand the legacy system [37].

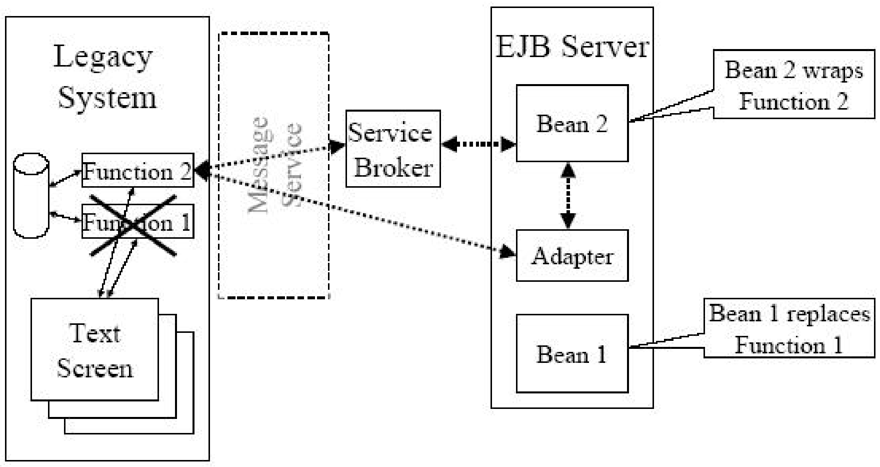


Figure 54: Wrapping Legacy Business Logic Using EJB

The next step in wrapping the legacy business logic is to build a single point of contact to the legacy system. It is a good idea to centralize all the communication knowledge in a single software artefact. The communication method used by this artefact depends on the particular situation. Options for communication between the single point of contact and the legacy system include RMI over IIOP, sockets, or even Message Oriented Middleware (MOM), which has the advantage of uncoupling the EJB server from the legacy system and allowing for asynchronous communication. This single point of contact can be implemented as a bean (called adapter) or as service broker – a software component placed external to the EJB server. Placing the point of contact inside or outside the server depends mainly on the chosen communication method and some security restrictions. For example, if you need to create a new thread or listen to a socket, the single point of contact must be outside of the application server because the EJB specification prevents Enterprise JavaBeans from being multithreaded or listening to sockets. The final step in wrapping the legacy business logic is to implement a wrapper bean for each module in the legacy system. In Figure 54, this wrapper is shown as Bean 2. These beans forward requests to the legacy system using the single contact point, in a manner similar to object wrapping.

There are several advantages to the component approach for wrapping legacy business logic. First, with relatively limited effort, the advantages of component-based systems are supported. We can, for example, build new Enterprise Java Beans that use the wrapper beans in unanticipated ways, greatly improving system flexibility. Second, wrapper beans are bona fide Enterprise Beans and can be integrated fully with all the management facilities and services included with the application server. Lastly, wrapping legacy business logic provides a roadmap to substitute the old system incrementally. After wrapping the functionality of the legacy system, we can re-implement wrapper beans one at a time (Bean 1 in Figure 54), without having to go through a “big-bang” replacement of the system. This is possible because the system and the clients would not notice any disruption as the re-implemented bean maintains the same interfaces provided by the wrapper bean. In time, it is possible to replace the old system completely.

Wrapping legacy business logic with EJB is not without risk. The EJB specification is porous and portability problems can arise between different vendor's application servers [38]. In addition, the Java programming language is considered by some to be unsuitable for critical applications.

As mentioned earlier, application servers manage the system services of deployed components (persistence, transactions, etc). However, a wrapping bean represents a piece of business logic living in an external legacy system. A mechanism is needed to coordinate the service management of both the legacy system and the EJB server. This mechanism, called connector, is being developed by Sun and promises to simplify the integration of legacy assets [39]. Connectors (shown in Figure 55) are a standard set of system level interfaces that manage transactions, security, and resource pooling between the server and the legacy resource. This architecture supports the development of a standard connector to access a legacy data source or system. The connector is plugged into an application server and provides connectivity between the legacy system and the application server. This connector, once developed, can be plugged into any application server that supports the connector mechanism. It is, of course, still necessary to implement application-level communication.

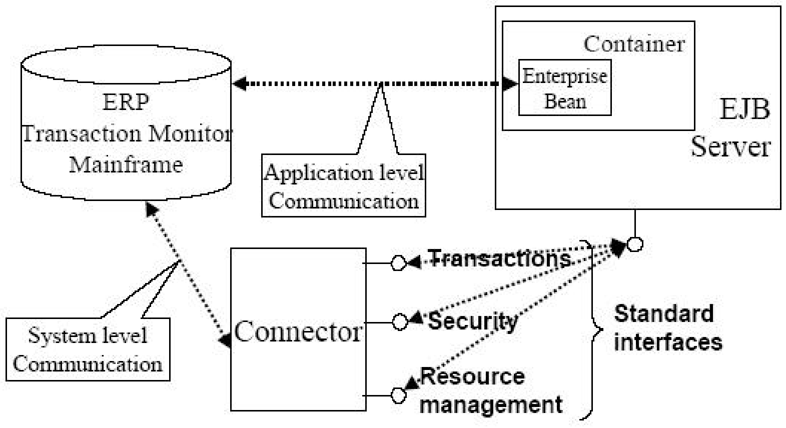


Figure 55: Integration of legacy business logic using connectors

In [31] a black-box modernization technique which is largely based on the component wrapping technique is presented. The general approach that is employed while porting a certain legacy application to grid services technology is based on the separation of its interface and implementation. Legacy interface is first extracted, then refined and finally exposed in the form of a grid service. Legacy implementation is encapsulated into a software wrapper which constitutes the necessary adaptation layer. The resulting components can be deployed on disparate hosts.

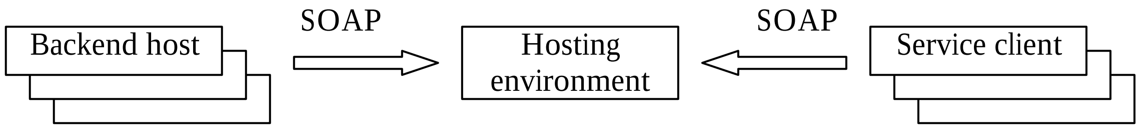


Figure 56: Wrapping legacy application with component wrapping

As described in Figure 56, the architecture which is used in [31] consists of three major components, i.e.:

1. Backend host

Backend host represents the machine on which legacy software is installed. In order to enhance performance and reliability, several instances of the same legacy application are deployed on different computing nodes which are located possibly in diverse geographic locations.

2. Hosting environment

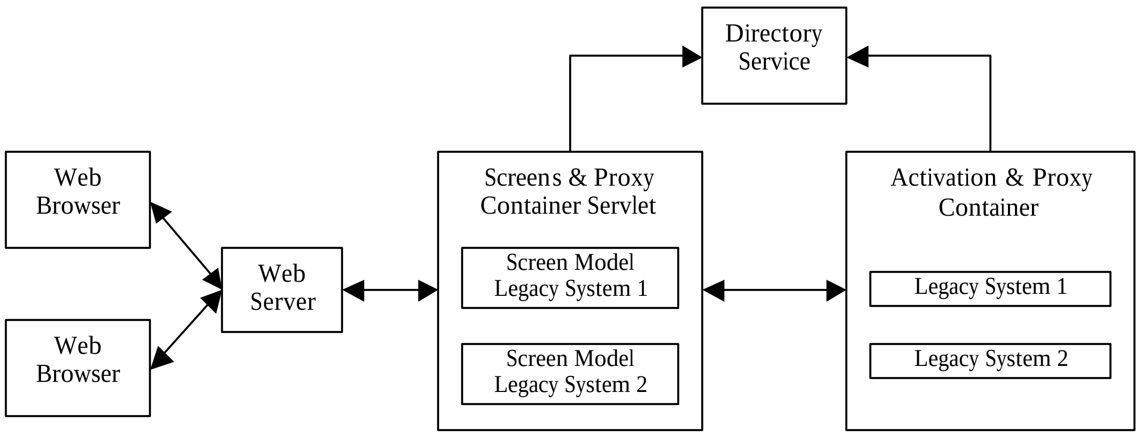
   Hosting environment is basically the container for grid services which provides the needed middleware facilities.

3. Service clients requesting grid services.

These three components are potentially located on different machines and communicate by means of SOAP messages. The only way in which information can be exchanged between the components is remote method invocation performed on grid services residing in the hosting environment.

3.6.3.3. Screen Proxies

A non-decomposable legacy program contains business logic and data model components in which their interfaces are dependent each other and cannot be separated. An approach to deal with non-decomposable legacy programs is described in [30] to integrate character-based legacy systems written in Cobol language with the Web and Grid technologies by using screen proxies and redirecting input/output calls. Figure 57 depicts legacy systems integration with web and grid technologies exploiting this technique.



Directory Service
Web Browser
Web Browser
Web Server
Screens & Proxy Container Servlet
Screen Model Legacy System 1
Screen Model Legacy System 2
Activation & Proxy Container
Legacy System 1
Legacy System 2

Figure 57: Integration of legacy systems with web and grid technologies

The Activation & Proxy Container (APC) on the right side of Figure 57 includes the information about legacy systems that either have been or may be activated. It interacts with Directory Service for registering the legacy system services that can be activated. When a web user connects to a Web Server to find and interact with a legacy system through a web user interface, the Web Server activates the Screens & Proxy Container Servlet (SPCS). The SPCS contains the references between the screens models and the associated activated legacy system. It contacts the APC associated to the legacy system that should be activated. Afterwards, the APC activates the legacy system and keeps a reference to it. The legacy system contacts the SPCS for creating its user interface which is actually done through the wrapper of the legacy system and through the proxy part of the APC. The SPCS creates an instance of a screen model corresponding to the legacy system user interface and connects them. The legacy systems continue to run and modifies its associated user interface through calls to the proxy part in the APC that redirect calls to the SPCS and then to the specific screen model. At a certain time the SPCS responds to the Web Server transforming the screen model to an HTML page. This may happen when the legacy system requests a user intervention. The Web Server forwards the answer to the web browser. After the user fills in an input field or the web browser simply asks for a refresh of the screen, the Web Server redirects the request to the SPCS. The SPCS redirects the web input to the legacy system through the APC.

3.6.3.4. Front-end OGSI Grid Service Layer

There is a clear need to deploy existing legacy code programs as OGSI Grid Services since legacy code applications have been developed and implemented to run on a single computer or on a computer cluster and do not offer a set of OGSA compliant interfaces. Two approaches can be identified to solve this problem. The first approach is to re-engineer the legacy code which in many cases implies significant efforts. The second one is to develop a front-end OGSI Grid service layer that contacts the target host environment without re-engineering the original code. The Grid Execution Management for Legacy Code Architecture (GEMLCA) supports the last approach and it is presented in [32].

The GEMLCA offers a front-end Grid service layer that communicates with the client in order to pass input and output parameters, and contacts a local job manager through Globus MMJFS (Master Managed Job Factory Service) to submit the legacy computational job. Thereby there is no need for the source code in order to deploy a legacy application as a Grid service. The user only has to describe the legacy parameters in a pre-defined XML format. The legacy code can be written in any programming languages and can be not only a sequential but also a parallel PVM or MPI code that uses a job manager like Condor and where wrapping can be difficult. The GEMLCA conceptual architecture is shown in Figure 58.

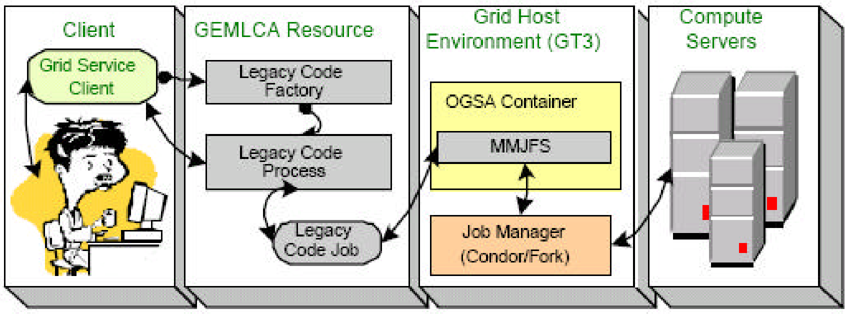


Figure 58: GEMLCA conceptual architecture

In order to access a legacy code program, the user executes the GEMLCA Grid Service client which creates a legacy code instance with the help of the legacy code factory. Following this, the GEMLCA resource submits the job to the compute server through GT3 MMJFS using a particular job manager (Condor or Fork).

A GEMLCA resource is composed of a set of Grid services that provides a number of Grid interfaces in order to control the life-cycle of the legacy code execution. This architecture can be deployed in several user containers or Tomcat application contexts.

GEMLCA has been designed as a three layer architecture: the first, front-end layer offers a set of Grid Service interfaces that any authorized Grid client can use in order to contact, run, and get the status and any result back from the legacy code. This layer hides the second, core layer section of the architecture that deals with each legacy code environment and their instances as Grid legacy code processes and jobs. The final layer, backend is related to the Grid middleware where the architecture is being deployed.

3.6.4. Integration Criteria (*to be updated*)

This chapter explains criteria in determining whether a given application qualifies for a grid solution. The criteria deal with job / application, data, usability, and non-functional perspectives.

3.6.5. Proposed Approach (*to be updated*)

This chapter will describe our approach in integrating legacy applications within Akogrimo project.

4. Conclusion

In this deliverable we have described the architecture and main functionalities provided by WP4.4 Grid Application Support Services Layer. All services described in the deliverable will be implemented using grid services and following WSRF specification [40], then we will use some support grid middleware to develop them. Inside the document there are, also, some topics that explain what application services can be useful in order to implement Akogrimo testbeds and some guidelines to provide a connector that makes legacy applications (Mobile) Grid-aware.

In next Figure 59 we will show an overall vision about interactions between entities belonging to WP4.4 in relation to a specific action. In this way we would give a general view to summarize role and relationships between entities.

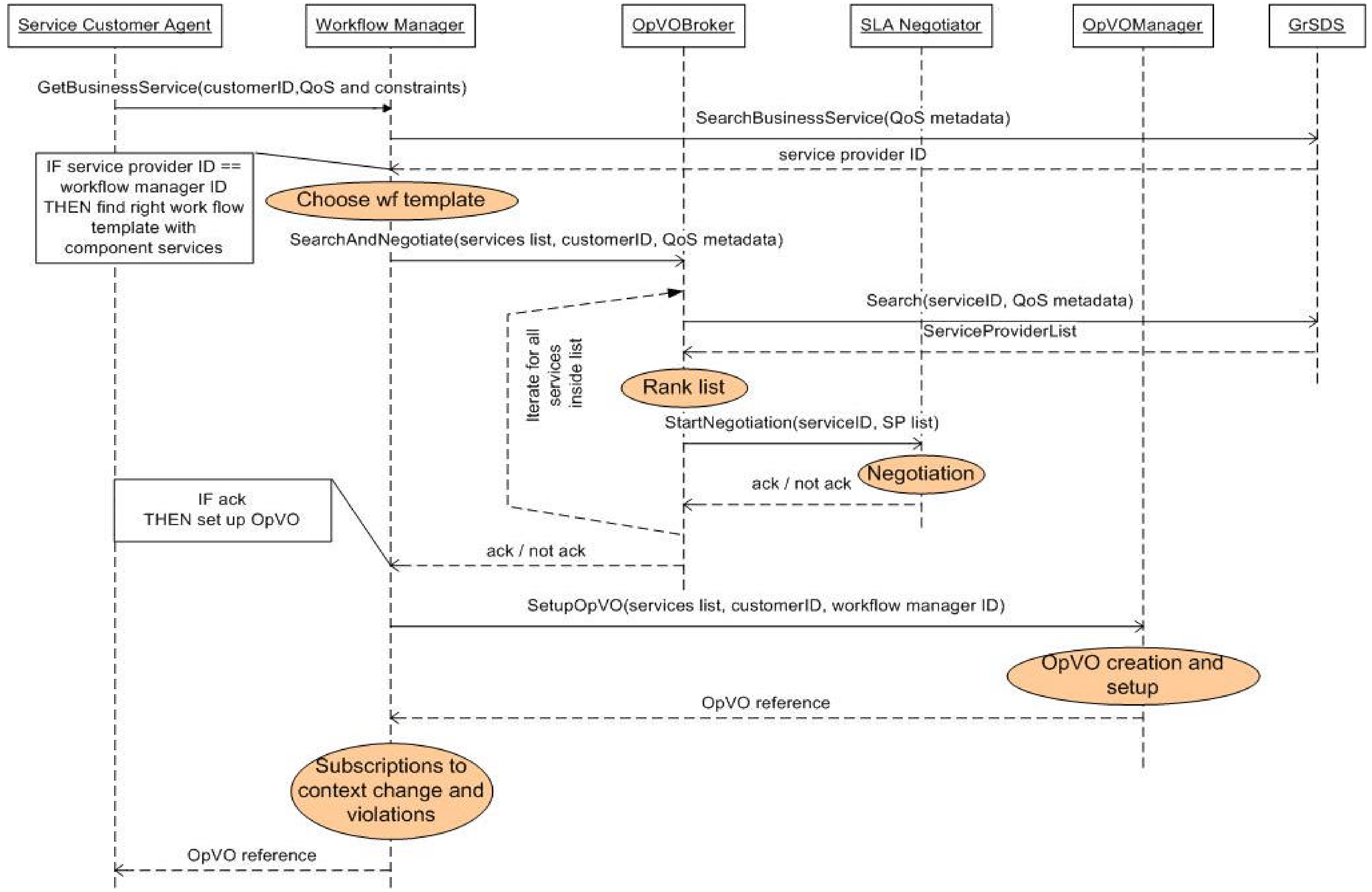

Service Customer Agent
Workflow Manager
OpVOBroker
SLA Negotiator
OpVOManager
GrSDS
GetBusinessService(customerID,QoS and constraints)
SearchBusinessService(QoS metadata)
service provider ID
IF service provider ID == workflow manager ID THEN find right work flow template with component services
Choose wf template
SearchAndNegotiate(services list, customerID, QoS metadata)
Search(serviceID, QoS metadata)
ServiceProviderList
Iterate for all services inside list
Rank list
StartNegotiation(serviceID, SP list)
ack / not ack
Negotiation
IF ack THEN set up OpVO
ack / not ack
SetupOpVO(services list, customerID, workflow manager ID)
OpVO creation and setup
OpVO reference
Subscriptions to context change and violations
OpVO reference

Figure 59 Summary of main interactions between WP4.4 components during "Acquire Service" action

References

[1] D2.3.2 Validation Scenarios, in folder http://bscw.hlrs.de/bscw/bscw.cgi/0/54606, document D2.3.2 Draft 0.3

[2] Open Science Grid, A Blueprint for the Open Science Grid, http://www.opensciencegrid.org/documents/ (2004)

[3] R. Alfieri et al, VOMS, an Authorization System for Virtual Organizations (2003 - check)

[4] C Kesselman, Grids, where next?, (2005)

[5] T. Dimitrakos, D.Golby, Paul K.earney, Towards a Trust and Contract Management Framework for dynamic Virtual Organizations (2004 - check)

[6] M. Lorch, B. Cowles, Conceptual Grid Authorization Framework and Classification, GFD-.038, 2004-11-23, http://www.ggf.org/Meetings/ggf7/drafts/authz01.pdf

[7] Web Services Trust Language, ftp://www6.software.ibm.com/software/developer/library/ws-trust.pdf

[8] D4.3.1 Grid Infrastructure Services Layer (Version 1.0)

[9] D4.2.1 Overall Network Middleware Requirements Report (Version 1.0)

[10] S. Gokul, *Authenticating Web Services with SAML*, http://www.awprofessional.com/articles/article.asp?p=28286

[11] Wang, D. Del Vecchio, M Humphrey, *Extending the Security Assertion Markup Language to Support Delegation for Web Services and Grid Services*, 2005 IEEE International Conference on Web Services (ICWS 2005), Orlando, FL, July 12-15, 2005, http://www.cs.virginia.edu/~humphrey/papers/SAML_delegation.pdf

[12] Web Services Trust Language, ftp://www6.software.ibm.com/software/developer/library/ws-trust.pdf

[13] D2.3.2 Testbed Validation Scenarios (Version 1.0)

[14] Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid

[15] Open Grid Services Infrastructure: http://www.gridforum.org/ogsi-wg/

[16] Open Grid Services Architecture: http://www.globus.org/ogsa

[17] Seacord, R. C., Comella-Dorda, S., Lewis, G., Place, P., Plakosh, D.: Legacy System Modernization Strategies, SEI, July 2001

[18] Comella-Dorda, S., Wallnau, K., Seacord, R. C., Robert, J.: A Survey of Legacy System Modernization Approaches, SEI, April 2000

[19] Huang, Y., Taylor, I. Walker, D., Davies, R.: Wrapping Legacy Codes for Grid-Based Applications, Proc. International Workshop on Java for Parallel and distributed computing, Nice, France, 2003

[20] Kuebler, D., Eibach, W.: Adapting Legacy Applications as Web Services: http://www-106.ibm.com/developerworks/webservices/library/ws-legacy

[21] Chikofsky, Elliot, J., Cross II, J.H.: Reverse Engineer and Design Recovery: A Taxonomy, IEEE Software vol. 7, 1990, p. 13-17

[22] Tilley, Scott, R., Smith, Dennis, B.: Perspectives on Legacy System Reengineering (draft), Reengineering Center, Software Engineering Institute, Carnegie Mellon University, 1995

[23] Phoenix Group: Legacy Systems Wrapping with Objects, http://www.phxgrp.com/jodewp.htm

[24] Haft, T. M., Vessey, I.: The Relevance of Application Domain Knowledge: The Case of Computer Program Comprehension, Information Systems Research vol. 6, 1995, p. 286-299

[25] Von Mayrhauser, A., Vans, A. M.: Comprehension Processes During Large Scale Maintenance, Proceedings of the International Conference of Software Engineering ICSE, Sorrento, Italy, p. 39-48, 1994

[26] Lakhotia, A., Deprez, J-C.: Restructuring Functions with Low Cohesion, Proceedings of the 6th Working Conference of Electrical and Electronics Engineers, 1998

[27] Wallnau, K., Morris, E., Feiler, P., Earl, A., Litvak, E.: Engineering Component-Based Systems with Distributed Object Technology, Lecture Notes in Computer Science, International Conference WWCA'97, Tsukuba, Japan, 1997

[28] Shaw, M.: Architecture Issues in Software Reuse: It's not just the Functionality, It's the Packaging, Proceedings IEEE Symposium on Software Reusability, 1995

[29] Plakosh, D., Hissam, S., Wallnau, K.: Into the Black Box: A Case Study in Obtaining Visibility into Commercial Software, Software Engineering Institute, Carnegie Mellon University, 1999

[30] Bodhuin, T., Tortorella, M.: Using Grid Technologies for Web-enabling Legacy Systems, Proceedings of the Software Technology and Engineering Practice (STEP), Amsterdam, 2003

[31] Balis, B., Bubak, M., Wegiel, M.: A Framework for Migration from Legacy Software to Grid Services, Cracow Grid Workshop, Cracow, 2003

[32] Delaitre, T., Goyeneche, A., Kiss, T., Winter, S. C.: Publishing and Executing Parallel Legacy code using an OGSI Grid Service, 2004

[33] Weiderman, N, Northrop, L., Smith, D., Tilley, S., Wallnau, K.: Implications of Distributed Object Technology for Reengineering, Pittsburgh, Carnegie Mellon University, 1997

[34] Stets, Robert, J., Hunt, Galen, C., Scott, Michael, L.: Component-Based APIs for Versioning and Distributed Applications, IEEE Computer, p. 54-61, 1999

[35] Cimitile, A., De Lucia, A., Di Lucca, A., Fasolino, A. R.: Identifying Objects in Legacy Systems, Proceedings of the 5th Workshop on Program Comprehension, 1997

[36] Seacord, R. C., Wallnau, K., Robert, J., Comella-Dorda, S., Hissam, S. A.: Custom vs. Off-the-Shelf Architecture, Proceedings of 3rd International Enterprise Distributed Object Computing Conference, University of Mannheim, Germany, 1999

[37] Plakosh, D., Hissam, S., Wallnau, K.: Into the Black Box: A Case Study in Obtaining Visibility into Commercial Software, Pittsburgh, Carnegie Mellon University, 1999

[38] Comella-Dorda, S., Robert, J., Seacord, R.: Theory and Practice of Enterprise JavaBean Portability, Proceedings International Society for Computers and Their Applications (ISCA) 1st International Conference on Information Reuse and Integration, Atlanta, Georgia, 1999

[39] Sun: J2EE™ Connector Architecture (JSR-000016), 1999

[40] http://www.oasis-open.org/news/oasis_news_03_17_04a.php OASIS – WSRF http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

[41] WSRF.NET framework http://www.cs.virginia.edu/~gsw2c/wsrf.net.html

[42] Liberty Alliance Project, http://www.projectliberty.org/index.php

[43] Web Service Policy Framework ftp://www6.software.ibm.com/software/developer/library/ws-policy.pdf

[44] XACML TC - http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml

[45] WS-Security TC - http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

[46] SAML - http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

[47] WS-SecureConversation - http://www-128.ibm.com/developerworks/webservices/library/specification/ws-secon/

[48] WS-Federation - http://www.verisign.com/wss/WS-Federation.pdf

[49] Web Service Policy Attachment - http://www-128.ibm.com/developerworks/webservices/library/specification/ws-polatt/

[50] Kerberos WG - http://www.ietf.org/html.charters/krb-wg-charter.html

[51] X509 WG - http://tools.ietf.org/wg/secsh/draft-ietf-secsh-x509/

[52] WS-SecurityPolicy - http://www-128.ibm.com/developerworks/webservices/library/specification/ws-secpol/

[53] Transport Layer Security - http://www.ietf.org/html.charters/tls-charter.html

[54] Information Systems Security Group at Kent University - http://www.cs.kent.ac.uk/research/groups/iss/index.html

[55] TrustCoM project - http://www.eu-trustcom.com/

Annex A. Alternative OpVO topologies

In most cases, the design of an OpVO is likely to be used in many situations. So we design an OpVO for monitoring a patient, allowing for the possibility of many patients. The whole process of monitoring a single patient can be regarded as a business case.

Two OpVO topologies can be distinguished:

- An OpVO with a single *business case* (SBC)
- An OpVO with multiple similar *business cases* (MBC)

What about

An OpVO per customer per workflow services still shared

An OpVO per workflow / similar workflows using same services with multi customers grouped in OpVO sharing services

Figure 60 and Figure 61 show a graphical representation of the two topologies. Each business case is represented by the Service Consumer. VO related information like membership tokens that each entity of the OpVO manages are represented by a circle.

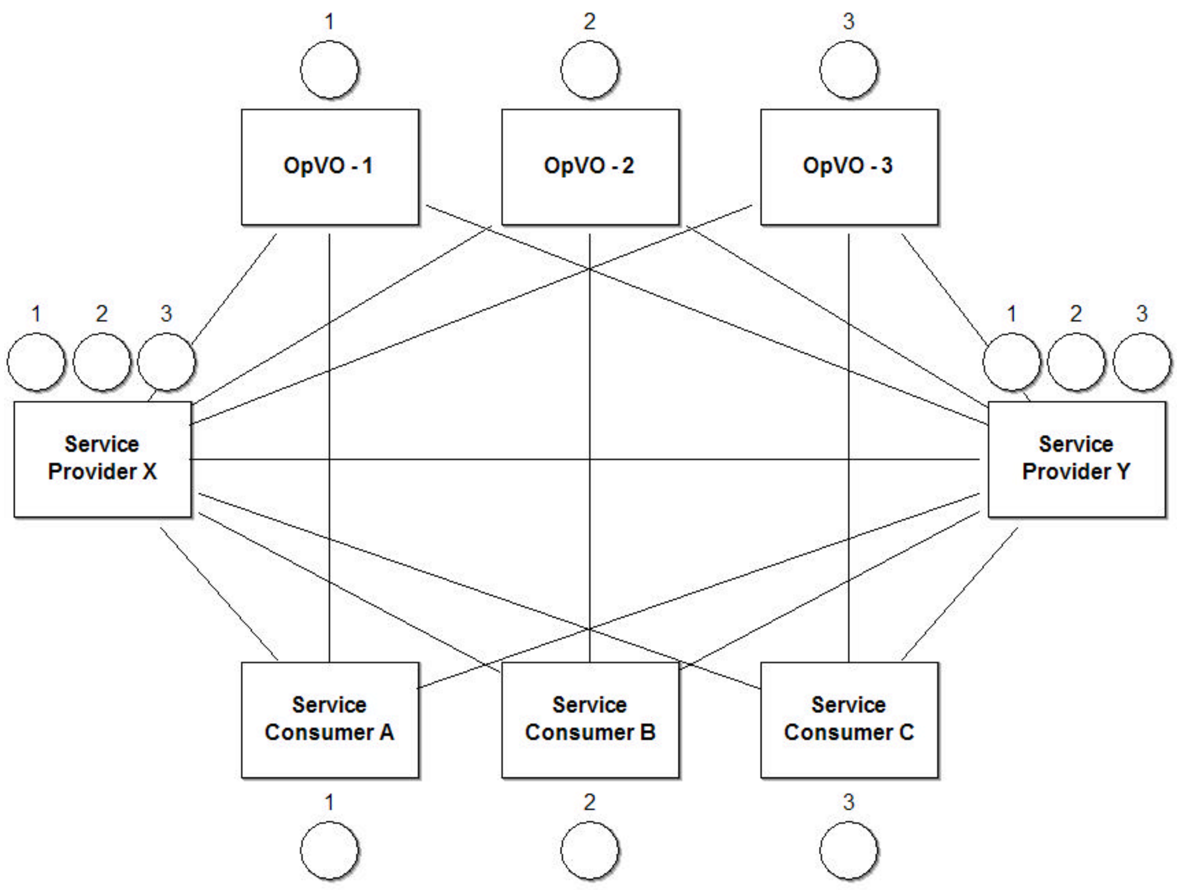


Figure 60 – Three OpVOs each with a single business case (SBC)

These two OpVO types are different in the complexity of achieving certain technical goals like:

- VO Management
- Performance
- QoS Management
- Security
- Scalability

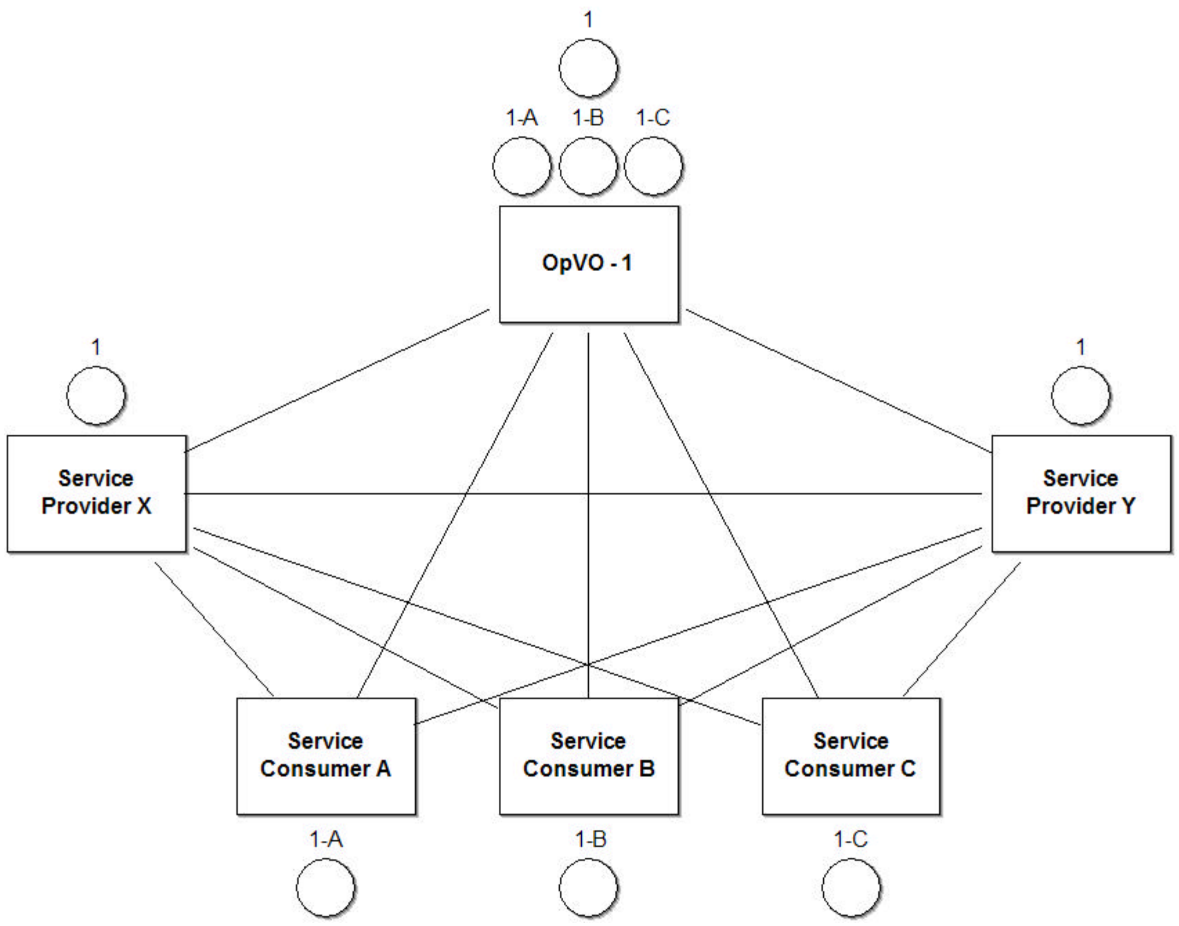


Figure 61 – A single OpVO with multiple similar business cases (MBC)

Differences in VO management:

Single business case OpVOs (SBC OpVOs) are rather short lived and focused on accomplishing one step in an overall process. The complexity in number of services is rather limited. E.g. in the eHealth domain an OpVO might be formed to provide a detailed diagnosis in preparation for a medical treatment. Specialists would be included into the OpVO and granted access to the necessary resources to perform their task. When the diagnosis (the service) has been provided the OpVO is dissolved.

Membership management is easier for the OpVO Manager in the SBC OpVO. There is no need to distinguish several unrelated customers. Service providers on the other hand may have to manage several OpVO memberships, but each service instance would only need to manage one. The Base VO services will have to support multiple VO's.

If any situation is expected to use the MBC method, a generic OpVO Manager will have to be written which handles multiple cases. This could encourage more generic core services.

The Multi OpVO services could also have to deal with membership of multiple VO's. For instance if a monitoring MBC OpVO contains all patient heart monitoring devices, if an alert is triggered that sets up separate OpVO's to deal with an emergency a response from one of the new VO’s would need to be sent back to the device to end the alert when the process is completed.

The MBC enables grouping of services in a process, rather than separate private processes. It encourages single VO per workflow type.

Differences in Performance:

From the viewpoint of the Base VO, creation and dissolution of multiple OpVO services means effort. Services have to be found and allocated, SLAs have to be agreed upon and configuration information has to be distributed. When choosing between the two alternatives, runtime behaviour has to be evaluated. Also the setup of the Base VO is an important factor when performance is required. Are there pre-arrangements between the Base VO members that make certain processes (e.g. negotiation easier)? Are certain critical services pre-reserved by the Base VO? How many actors are involved? As multi business case OpVO have a tendency to be longer lived the overhead in creating the OpVO is reduced.

Differences in QoS management:

Service providers may prefer having a contract with only one OpVO in order to make management of their services simpler. If they sell their services to multiple OpVOs the potentially different contracts regarding quality of service may require weighing multiple factors to achieve an optimal overall performance. This is true but commercially it would be appealing if a service could appeal in various types of VO.

Differences in Security:

For an MBC OpVO, special care has to be taken to keep the service consumers separate. Patient information from the one case should not interfere with patient information from another case.

Differences in Scalability:

The additional measures that have to be taken to ensure security can influence the overall performance and scalability of the system. More information has to be processed. More entities are related by the same workflow. Tight cohesion hampers scalability and makes management more complex. However scalability could be enhanced if this complexity is dealt with well, the singleOpVO model probably would present more headaches in terms of scalability.

Does the MBC alternative eventually become a bottleneck when there are a large number of cases?

Annex B. Summary tables and figures

This annex provides a table that shows in one place a summary of the major components and the constituent subcomponents. The Application Specific services and legacy application integration are not included.

Section title	**VO Management**	**Orchestration**	**SLA Management**	**Security**
Name of major component in WP4.4	VO Management	BP Enactor	SLA High Level services	VO Security Services
Sub Components	BVO Manager OpVO Manager OpVO Broker Participant Registry	Workflow Manager Monitoring Daemon Enactment Engine Workflow Registry	SLA-Access SLA-Translator SLA-Negotiator SLA-Contract SLA-ContractRepository SLA-Template SLA-Repository SLA Template Builder	VO Authentication Service VO Authorization Enforcement Security Log Security Bridge User Agent Service Agent. Security Designer

Table 34 Components and Subcomponents