D4.4.3

Report on the Implementation of the Application Support Services Layer

Version 1.0

WP WP4.4 Grid Application Support Services Layer

Dissemination Level: Public

Lead Editor: Giuseppe Laria, CRMPA

15/02/07

Status: Approved by QM

SIXTH FRAMEWORK PROGRAMME

PRIORITY IST-2002-2.3.1.18

Information Society

Grid for complex problem solving

Proposal/Contract no.: 004293

Context

Activity 4	**Detailed Architecture, Design & Implementation**
WP4.4	**Grid Application Support Services Layer**
Dependencies	**This deliverable uses specifically the input of the deliverables D4.4.2, D4.4.1 .**

Contributors:

Contributors (in alphabetical Order):	**Reviewers:**
ATOS: Section 3.2	Fredrik Solsvick
CCLRC: section 3.1, 3.3, 3.5	Nuno Inacio
CRMPA: section 1, section 2, 3.1, 3.2,3.5, Executive Summary	Antonis Litke
DATAMAT: section 3.3	
TID: section 3.2, 3.3	
USTUTT: section 3.4	

Version	**Date**	**Authors**	**Sections Affected**
0.1	11/12/06	CRMPA	Template
0.2	22/12/06	All	Section 3
0.3	12/01/07	CRMPA	Introduction, Section 2
0.4	19/01/07	All	Section 3 update
0.5	26/01/07	CRMPA, CCLRC	Section 3.5 and Executive Summary
0.9	31/01/07	CRMPA	Editing of final version for internal review
1.0	14/02/07	All	Revision applied

Table of Contents

List of Figures

List of Tables

Abbreviations

Akogrimo	Access To Knowledge through the Grid in a Mobile World
ASS	Application Specific Services
BVO	Base Virtual Organization
BVOm	BVO Manager
CM	Context Manager
ECG	Electrocardiogram
ECG_DA	Electrocardiogram Data Analyzer
ECG_DV	Electrocardiogram Data Visualizer
ECG_DG	Electrocardiogram Data Generator
EMS	Execution Management Service
EEngine	Enactment Engine (also known as WorkFlow Engine)
EPR	End Point Reference
GASS	Grid Application Support Services
GrSDS	Grid Service Discovery Service
GT4	Globus Toolkit 4
GW	Gateway
HE	Hosting Environment
MD	Monitoring Daemon
MDL	Medical Data Logger
OpVO	Operative Virtual Organization
OpVOBr	OpVO Broker
OpVOm	OpVO Manager
PR	Participant Registry
SA	Service Agent
SA_ECG_DA	Service Agent ECG data analyzer
SA_ECG_DG	Service Agent ECG data generator

SA_ECG_DV	Service Agent ECG data visualizer
SA_MDL	Medical data logger
SDFS	Service Description Fact Sheet
SIP	Session Initiation Protocol
SIP_Br WS-Int	SIP Broker WS-Interface
SLA	Service Level Agreement
SLA_A	SLA Access
SLA_N	SLA Negotiator
SLA_N_F	SLA Negotiator Factory
SLA_R	SLA Repository
SLA_T	SLA Translator
UA	User Agent
UAF	User Agent Factory
WF	Workflow
WF_R	Workflow Registry
WM	Workflow Manager
WM_F	Workflow Manager Factory
WS	Web Service
WSDL	Web Service Description Language

Executive Summary

This document assumes a general understanding of Akogrimo concepts and a high level knowledge of Grid Application Support Service layer architecture (see deliverables [1] "The Mobile Grid Reference Architecture" and for more details [2] "Architecture of Application Support Service Layer").

It describes the final release of the Akogrimo Grid Application Support Service (GASS) prototype. All the implemented components are described including interfaces and involved technologies.

Starting from the results carried out by the first prototype implementation, the WP4.4 participants have updated the existing implementation in order to provide the functionalities that were not implemented during the first project cycle, although included in the GASS architecture.

In particular, the final implementation mainly covers the OpVO creation process that in the first prototype has been performed through an offline setup that required manual pre-configuration of the execution environment.

In order to achieve a dynamic creation process the following additional components have been implemented:

- OpVO Broker
- SLA Negotiator
- SA Factory

Furthermore the existing implementations have been updated in order to support the complete process and interactions underlying the OpVO creation. Finally the security infrastructure allowing service calls authentication and authorization has been implemented as well.

The deliverable is organized as follows:

- Section 1: provides an overall introduction to the document describing its scope and goal, as well as a description of the prototype goal.
- Section 2: provides an overview of the prototype, listing the implemented components. Moreover the main high level features of this prototype are described explaining how the listed components interact to provide those features.
- Section 3: provides a detailed description of each implemented component, including interface definition and interactions with other modules. Details about involved technologies are provided as well.
- Section 4: provides final conclusions and future plans about the use of the prototype in order to run the test beds

1. Introduction

This document is the updated version of the first implementation report related to the Grid Application Support Service (GASS) layer of Akogrimo project.

It is assumed that the reader is already familiar with the Akogrimo concepts and has a high level. knowledge of GASS layer architecture (see [1] and for more details [2]).

The GASS layer represents the "front-end" of the Akogrimo platform. In fact, it provides high level services that allow the application to have an entry point to leverage on the Akogrimo platform capabilities. In particular, this layer provides two main features:

- Management of the Base VO.
- Creation and management of the Operative VO.

The design has split the architecture in four main subsystems:

- VO management: includes services to manage Base VO and Operative VO.
- SLA High Level[1]: it provides services to manage negotiation and contract definition during the OpVO creation phase
- BP enactment: it provides services to manage the instantiation, execution and monitoring of a workflow template.
- Grid Service Discovery Service (GrSDS): it is the "yellow pages" of the BVO and it can be searched for services published by a Service Provider participating in the BVO

A security infrastructure across the different subsystems has been designed and the implementation is going to be finalized according with the features described in section 3.5.

The details about the available components and related functionalities are provided in section 3.

1.1. Implementation Scope

The final prototype of the GASS layer (hereafter "Prototype") implements the architecture designed in [2] and, integrated with the prototypes implemented in the other Work Packages, will be the basic infrastructure running the test beds. Thus the goal of this prototype is to support properly the execution of the test bed scenario, during the validation phase of Akogrimo project, in particular, providing the following capabilities:

- Populating and administrating the Base VO
- Searching and negotiating services to be invoked during the execution of a workflow
- Creating and populating an OpVO. It includes:
    - Creating instances of OpVO management and monitoring services
    - Creating and running dedicated work flow instances
    - Assuring invocations from the work flow towards the services negotiated with the Service Providers

[1] The SLA management is split in two different parts: the SLA High Level (part of GASS layer) and SLA Enforcement (part of Grid Infrastructure Service Layer). The SLA High Level, mainly addresses the negotiation phase and the management of SLA templates and contracts. Once a service is being used, parts of the SLA Management have to supervise the execution phase, to ensure that the service is running as agreed (This is the role of SLA Enforcement).

- Use of the OpVO that includes:
  - Running and monitoring the work flow instance execution
  - Adapting the work flow execution in accordance with the context changes
  - Managing interactions between the user invoking the OpVO (i.e. work flow) and the OpVO (i.e. work flow) invoking services in the Service Provider administrative domain
- Preventing unauthorized access to BVO and OpVO environment

1.2. Document structure

After this introductive section explaining the scope and the goal of the Prototype, this report is organized as follows:

- *Section 2 (overview)*: it provides an overall overview of the behaviour of the Prototype describing how each capability listed in section 1.1 is made available through the interaction between the components of Akogrimo infrastructure.
- *Section 3 (GASS components details)*: reports technical details about the implementation of each component developed within the frame of GASS layer.
- *Section 4 (conclusions)*: summarizes the implementation results and provides suggestions for future implementations tasks to be performed beyond the Akogrimo project lifecycle.

2. Prototype Overview

2.1. Prototype Software Components

Table 1 summarizes all the software components implemented in the final prototype and for each component a brief description about the provided functionalities has been included. In order to have a comprehensive understanding refer to section 3 that provides a detailed description of components functionalities, technical choices and components interactions.

Table 1 – Prototype Software Components

Subsystem	Software component	Description
VO Management	User Agent Factory – UAF	*It allows to create dynamically UA for each new participant of the BVO*
	Service Agent Factory - SAF	*It allows to create dynamically the SA that will act on behalf of external services inside the OpVO*
	Operative VO Broker Factory – OpVOBr_F	*It is in charge of managing the process to search and negotiate for services to be invoked by the OpVO*
	Base VO Manager – BVOm	*The Base VO Manager is the key part of the policy and authorization enforcement at the top level of the Base VO*
	Operative VO Manager Factory – OpVOm_F	*The Operative VO Manager takes the role of the VO manager in terms of authentication and authorisation of service calls to the VO during the execution of application specific services*
	Participant Registry – PR	*The Participant Registry service is actually constituted by three services that provide different kind of information on the BVO/OpVO and their participants*
SLA Negotiation	SLA Negotiator Factory – SLA_N_F	*SLA-Negotiator Service represents the service that has to be contacted in order to lead the service negotiation process in the SP domain.*
	SLA Access – SLA_A	*This service provides all necessary functionalities that other components could require from the SLA document template and contract.*

Subsystem	Software component	Description
	SLA Translator Factory– SLA_T_F	*This service is the only responsible for providing access to SLA documents (SLA-Template and SLA-Contract) and get (or set) information of them*
	SLA Repository – SLA_R	*The SLA Template Repository allows storing and retrieving of SLA documents*
BP Enactment	Monitoring Daemon – MD	*The purpose of this service is to provide a web service interface that the Context Manager and SLA Enforcement can use to notify BP Enactment about context changes or SLA violations*
	WF Manager Factory – WM_F	*Its purpose is to transform workflow templates into workflows that can be carried out by the Enactment Engine; part of this transformation includes service instantiation and registration for context changes (with the Context Manager) and SLA violations (with SLA Enforcement).*
	Enactment Engine – E_E[2]	*The Enactment Engine is the component of the Business Process Enactor in charge of enacting specific BPEL processes submitted by the Workflow Manager component.*
	SIP Broker WS-Interface	*This service does not represent a component itself, but it aims to provide a WSRF service interface to the Sip Broker component available in the platform.*
	WF Registry - WR	*The Workflow Registry is the component in charge of storing implementation files of published workflows*
GrSDS	GrSDS proxy – GrSDS_P	*The service discovery server is divided into two parts – the service repository and the service discovery proxy. The service repository is principally replaceable, whereas the proxy stays the same. This way the proxy hides the actual service registry implementation from the search clients*
	Service Registry – S_R	

[2] Also known as Workflow Engine

Figure 1 shows the static[3] relations between the internal components of WP4.4. It includes interactions with components external to the WP as well. More in detail:

- The dotted lines in the figure show an interaction with a component developed in another Work Package (the belonging WP is indicated below the service icon)
- The continuous lines show an interaction with a component inside the same Work Package
- The external box groups all subsystem belonging to the WP4.4
- The internal boxes group components belonging to the same subsystem (e.g. VO management)
- The label associated to a line explains which communication protocol is used in that interaction.

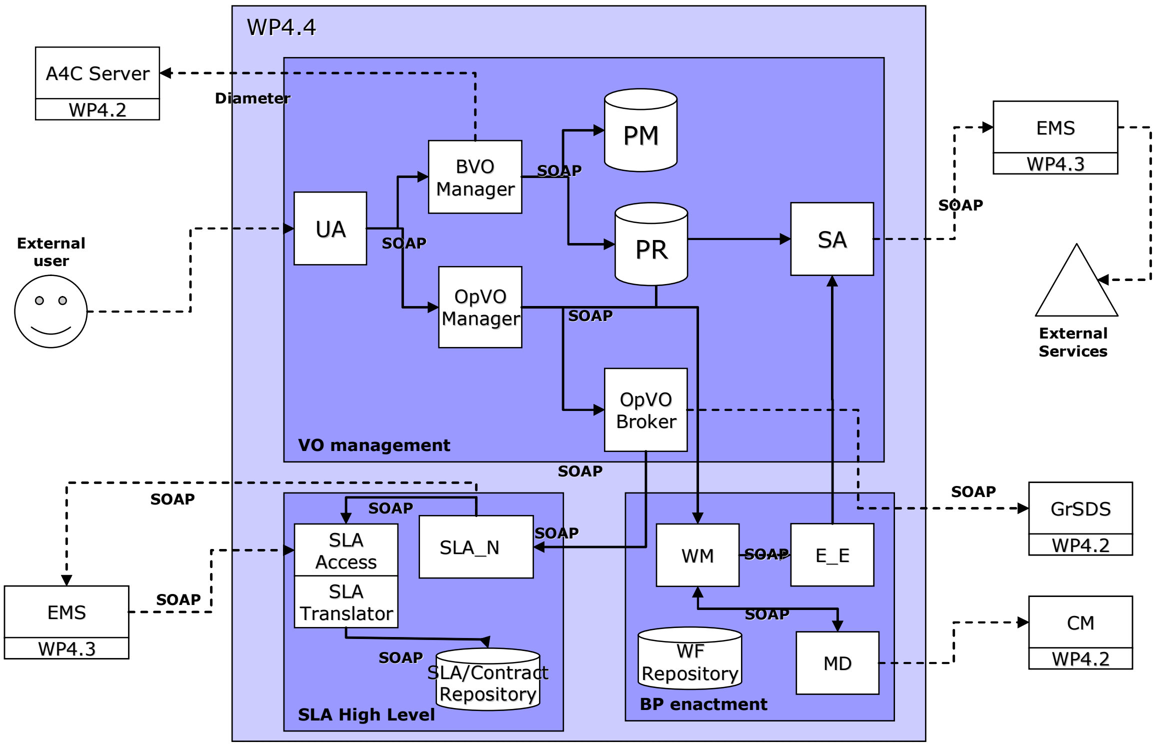


Figure 1 –Interactions between the software components

The above table and figure have the goal of providing the basic background in order to understand the description of the Prototype behaviour provided in the next subsections. The SIP Broker WS-Interface is not included in the above figure because it does not represents a component itself (see section 3.3.5 for details).

2.2. Overview of available features

Section 1.1 introduced the main features provided and they can be grouped in three main scenarios covered by the Prototype:

- BVO administration;
- OpVO creation;

[3] Static because they do not show any kind of sequence but just that a interaction can occur between two components. For details about those relations refer to section 2.2 and to the description of each component (section 3).

- OpVO use

This section describes how the Prototype works (as a whole) in order to provide the features necessary to address the above scenarios and which components are involved to carry out each specific feature.

2.2.1. BVO Administration

The precondition to address any scenario is to set up a Base VO and configure it. In order to set up a BVO it is necessary to install all the services listed in Table 1 according with the hosting environment requirements described in section 3. All the hosting machines are assumed to be part of the same administrative domain[4]

In order to populate the BVO (including new members and services) some configuration actions have to be taken up and they are necessary to operate the following scenarios.

The functionalities associated to the configuration are executed by the administrator and they are:

- Definition of roles and rules valid inside the BVO
- Subscription to BVO
- Publishing services in the BVO
- Storing WF templates inside the BVO

2.2.1.1. Definition of roles and rules

Table 2 summarizes the general authorization rules associated with the main role that a participant can play within a BVO

Table 2 - Roles and associated authorization rules

Role	Rule
Customer	They are allowed to: ▪ Search for application ▪ Create OpVO that supports the selected application ▪ Use the created OpVO
Service Provider	They are allowed to publish a service in the GrSDS providing the required information
Administrator	He/she is allowed to configure the BVO services

The component involved to provide this functionality is the Policy Manager. This is a service developed in WP4.3. The Administrator will use the PM administrative interface to add new policies that will describe the rules above. (see [5] for details about the Policy Manager).

In this particular case, the rules are related to the authorization process and the associated authorization policies.

[4] Apart from the negotiator hosting machine. Each SP will have a SLA Negotiator Factory installed in the private administrative domain.

2.2.1.2. Subscription to the BVO

The following steps are necessary to subscribe a new BVO participant:

- The subscriber is member of a trusted NP domain
- Associating a role to the new subscriber
- Creating a UA instance associated to the new member
- Defining the profile of the new member
- Updating the BVO Participant Registry with the new member and the associated profile
- Updated the member profile in the home domain A4C

All the above operations are performed by the Administrator using the Participant Registry administrative GUI. That means to create a new member in the PR and to store the associated profile (see Annex A.1 for details about the content of the member profile).

If the subscription is successful the administrator of the home domain will be asked for updating the member profile in his home domain (see deliverable D4.2.3 for details [3])

Figure 2 describes the subscription process and a possible deployment of the involved services:

- The Administrator uses the Administrative GUI to add BVO participants and to associate them User Agent and profile
- The Administrative GUI updates the PR on the basis of the actions required by the administrator

Both Administrative machine and PR Hosting machine are hosted in the same administrative domain (hereafter BVO domain).

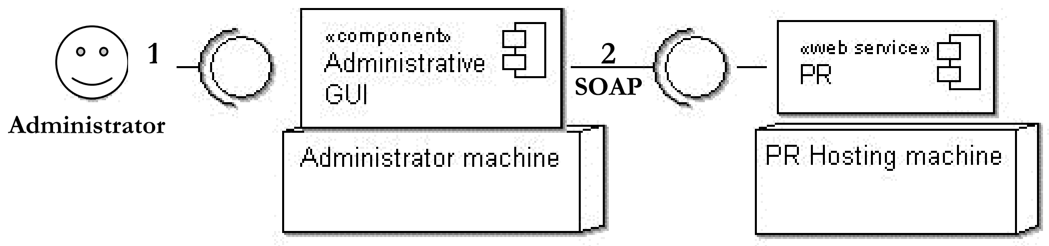


Figure 2 – Subscription to the BVO

2.2.1.3. Publishing services in the BVO

A member subscribed as Service Provider (SP) can publish the services he is able to provide. In order to do that the SP has to invoke the GrSDS providing all the required information in the publication process.

It is possible to do that in two different ways:

1. By communicating offline to the BVO domain Administrator the services to be published
2. By using a software client running in the SP domain to publish directly the information in the GrSDS

The first case is performed by the BVO domain administrator using an administrative GUI.

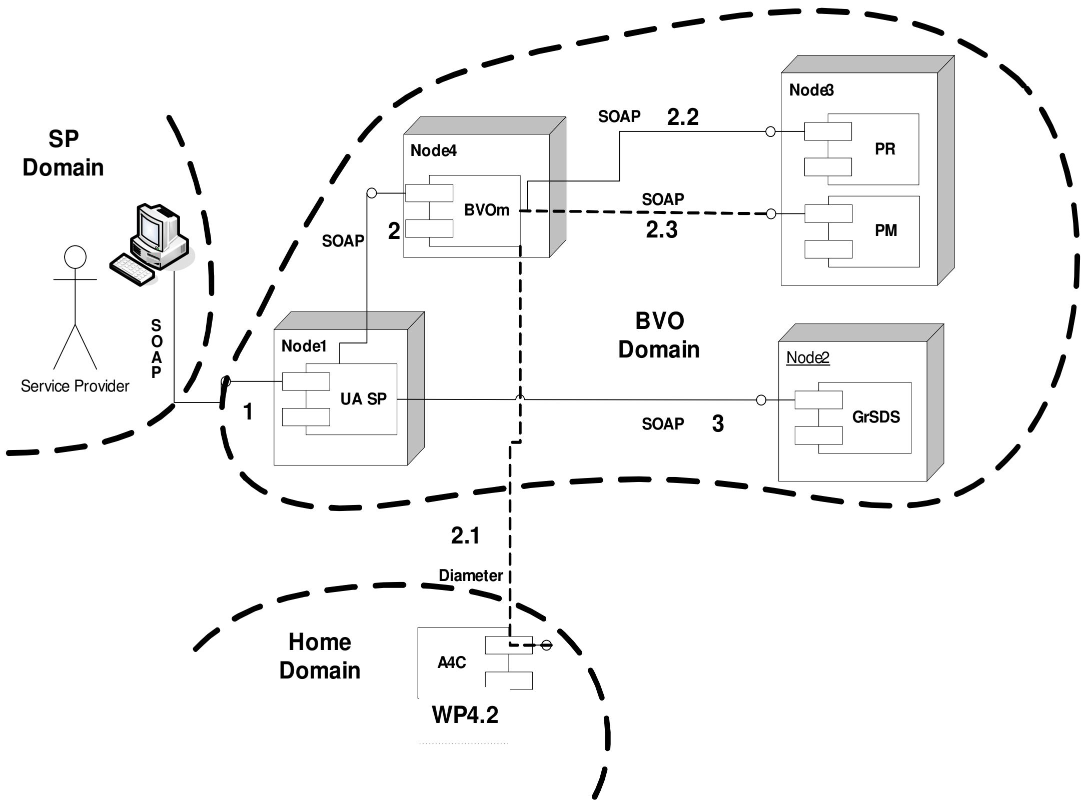


Figure 3 –SP publishes a service

The most interesting case is the second one that implies interactions between different domains and the above Figure 3 sketches the behaviour of the system:

1. A member subscribed as Service Provider invokes his UA to publish a service
2. Before performing the operation the UA asks the BVO Manager (BVOm) for authenticating and authorizing the request. The BVOm:
    2.1. Checks the identity[5] with the A4C of the SP home domain[6]
    2.2. Retrieves the role associated to the identity from the PR
    2.3. Retrieves the policy associated to the role and takes a decision
3. If the authentication and authorization are successful, the UA invokes the publication on the GrSDS on behalf of the Service Provider.

2.2.1.4. Storing WF templates

The WF Repository of the Base VO has to be populated with the WF templates associated to the available applications (e.g. eHealth, eLearning, DHCM…).

The BVO Administrator does that using a dedicated client that allows storing the template and the associated description files that allow searching for services to be orchestrated in the specific template and to deploy the WF described by the template.

[5] See deliverable [1] for details about Akogrimo identity model and management

[6] The SP home domain is the domain of the Network Provider where the SP is registered (see section 3.5.1 for description of Akogrimo multidomain model)

2.2.2. OpVO creation

If a BVO is established, the members can start using it. In particular, the authorized members (customers) can ask for creating an OpVO. The OpVO is the environment that will allow executing and invoking the application. The creation of an OpVO implies the creation of a dedicated domain (inside the BVO domain) that will be owned by the customer that has asked for its creation, hereafter the OpVO domain.

The OpVO creation can be logically split in four phases that in any case are executed in sequence and are part of a single process:

- Initial OpVO domain setup
- Search and negotiation for services to be orchestrated
- WF deployment
- Final setup

2.2.2.1. Initial OpVO domain setup

A BVO member registered as a customer can start the process that ends with the creation of an OpVO. Figure 4 describes the first phase of this process:

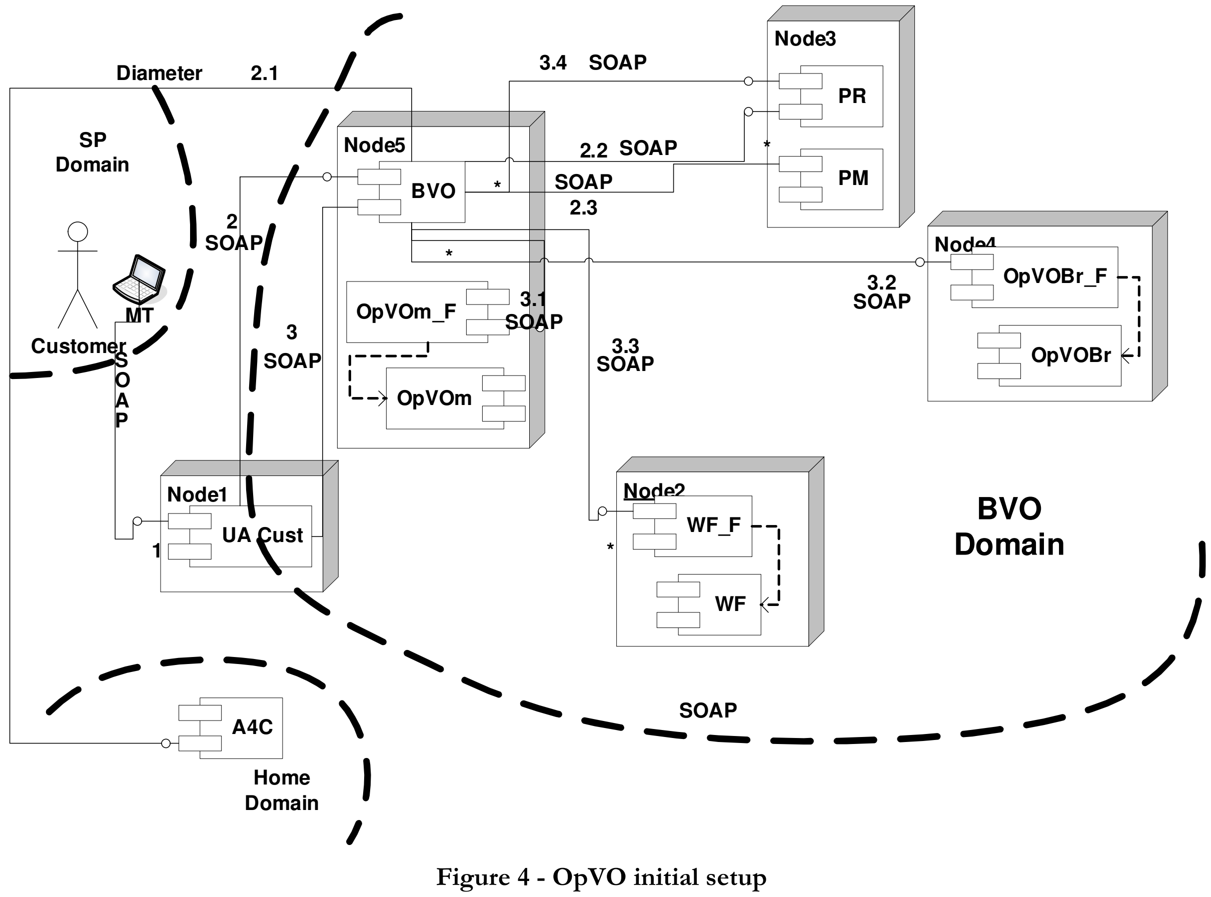


Figure 4 - OpVO initial setup

Figure 4 depicts the following behaviour:

1. A customer of the BVO invokes his UA to ask the creation of an OpVO
2. Before processing the request the UA asks the BVOm for authentication and authorization of the requestor. The following steps 3.x are similar to the ones described in section 2.2.1.3 Figure 3

3. The UA processes the request invoking the BVOm that actually starts the initial setup creating an instance of:
   - 3.1. OpVOm invoking the OpVOm_F
   - 3.2. OpVOBr invoking the OpVOBr_F
   - 3.3. WM invoking the WM_F
   - 3.4. Each new instance is a new member of the BVO then the BVOm invokes the PR to update the list of participants

2.2.2.2. *Search and negotiation*

All services necessary to manage the OpVO have been created in the initial setup. The following phase actually starts the interactions among the different services. In particular this phase focuses on retrieving the WF template and searching the services to be negotiated in order to execute the WF.

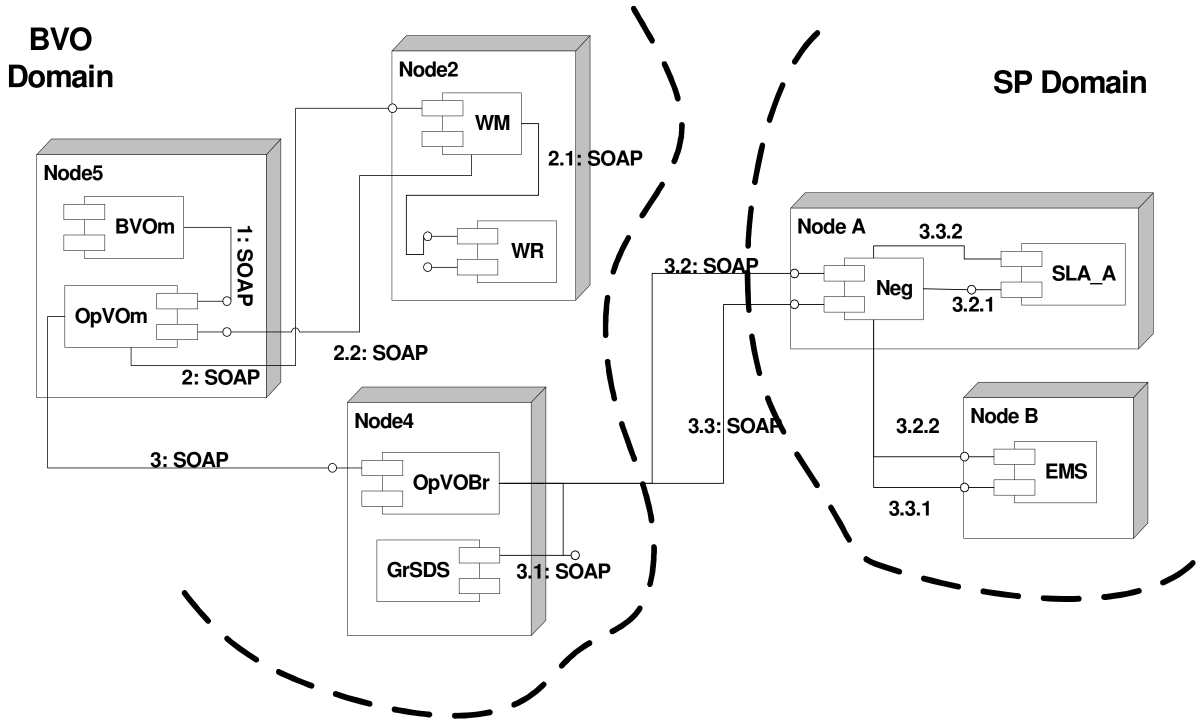


Figure 5 - Search and negotiation step

Figure 5 shows how the prototype components interact to find out the services to be orchestrated by the workflow:

1. This step follows step 3 of the initial OpVO setup phase: the BVOm invokes the new OpVOm instance to create the OpVO passing references to all created instances
2. The core of the OpVO is the execution of a workflow, then the OpVOm invokes the WM to instantiate the required application.
   - 2.1. WM retrieve the associated WF definition (it includes description of services to be invoked by the WF)
   - 2.2. WF asks the OpVOm for providing the single services to be orchestrated
3. OpVOm forwards the service description to the OpVOBr asking for providing references to available services of the specified type

3.1. OpVOBr searches the GrSDS[7] to find out service provider potentially able to provide the services

3.2. For each service the OpVOBr gets a list of potential SP and then invokes the related SLA Negotiator service to establish a contract. In step 3.2.2 the negotiator checks with the EMS the availability of resource to provide the services with the required quality of service and returns a counter offer

3.3. OpVOBr checks the offer and invokes negotiator to accept or refuse it. In step 3.3.1, if the negotiation is successful, the negotiator reserves resources with the EMS to make the service available at the required time and a final contract is established.

At the end of step 3 a set of parameters to invoke the negotiated services are forwarded back to the OpVOm

2.2.2.3. WF Deployment

After the second phase, all the information are available to pass from a template to a concrete WF, then the third phase can be started that brings to the deployment of the WF.

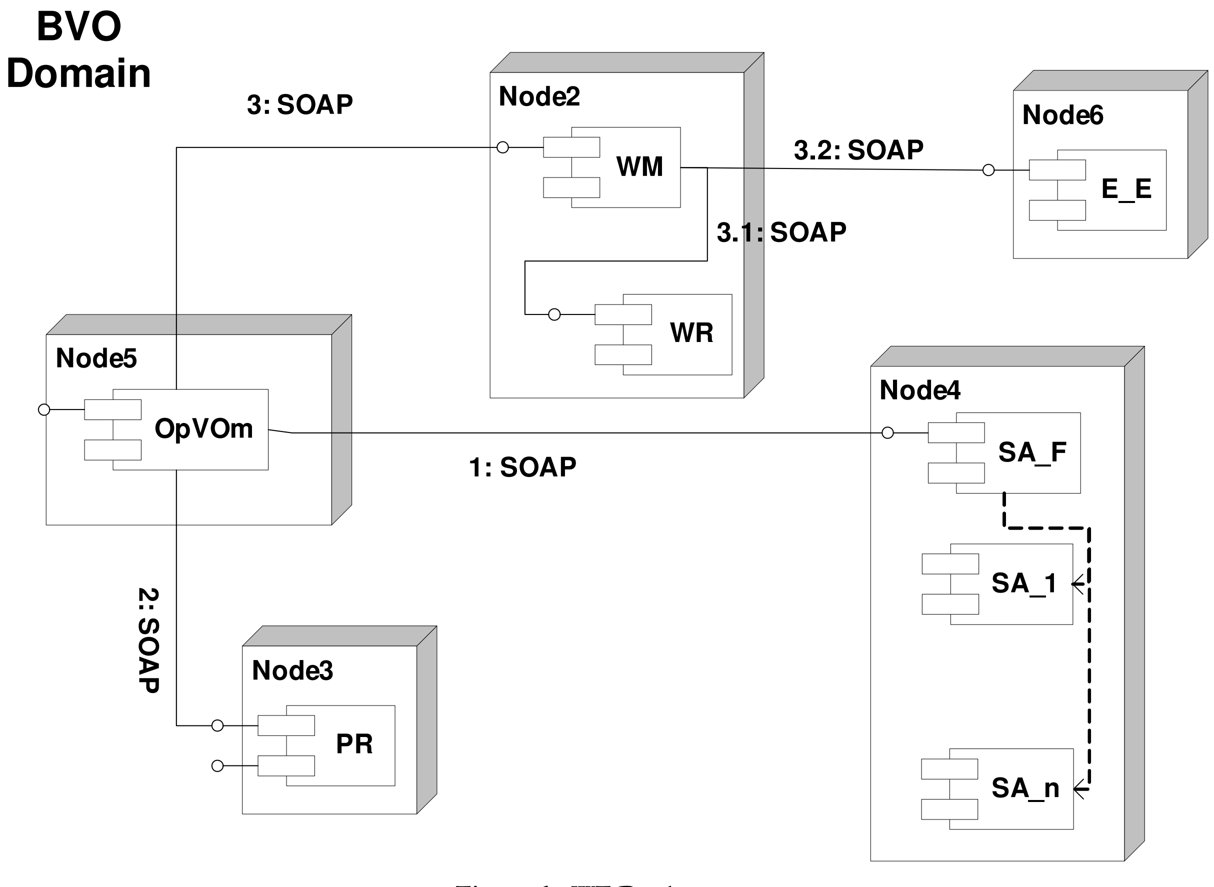


Figure 6 - WF Deployment

The OpVOm receives references to the negotiated services and additional information to invoke them, then it invokes:

1. The SA Factory for instantiating a SA for each service to be orchestrated. Each SA instance is configured on creation with the information to invoke the associated service

[7] For details about how to perform this query refer to section 3.4

2. The PR to include the new SA instances that are now members of the BVO
3. The WM providing the references to the SA instances (they acts on behalf of the external services inside the OpVO and they are invoked by the WF during the execution
   3.1. The WM gets the WF description
   3.2. The WM includes in this description the SA to be invoked during the execution and deploy this WF in the E_E.

At the end of this phase, a WF can be invoked on the E_E.

2.2.2.4. *Final Setup*

After the WF deployment phase the environment is ready to execute the WF, some final setup actions have to be performed in order to allow each service to interact properly.

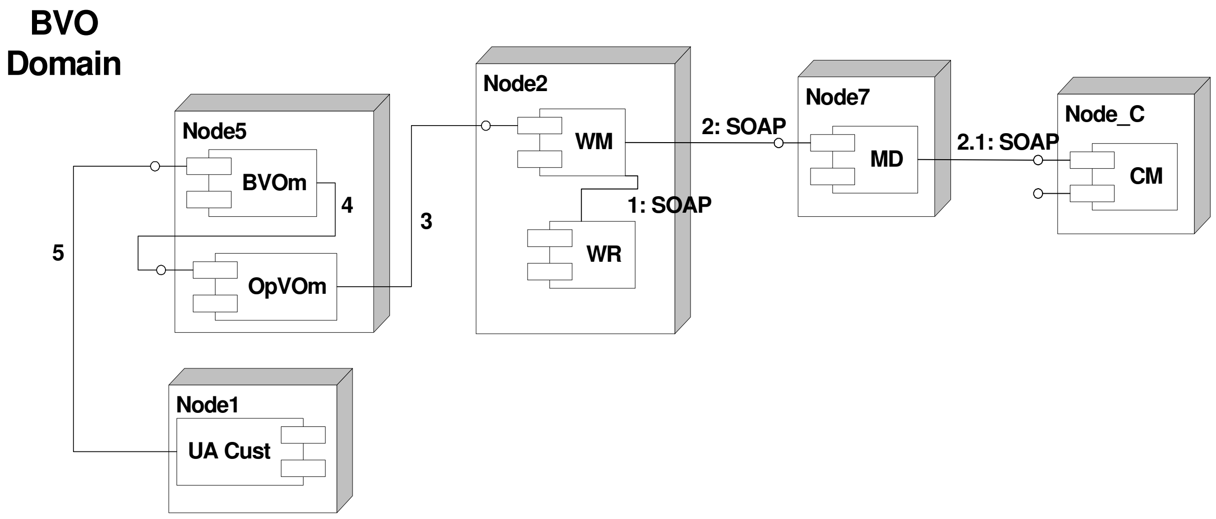


Figure 7 - Final Setup

The final setup consists in configuring the MD and the UA to allow, respectively, monitoring of WF execution and following invocations to the WF from the customer:

1. The WF Manager (WM) retrieves from the template the events (e.g. context changes, faults,…) on which the work flow engine has to be notified
2. The WM configures the MD with this information
   2.1. The MD subscribes to the CM to be notified about the specific context changes information
3. Step from 3 to 5 are the final ones: the process is successful and all the information are passed back until the component that has started the creation. The UA Cust configures itself to invoke the new workflow deployed in the E_E.

2.2.3. OpVO Use

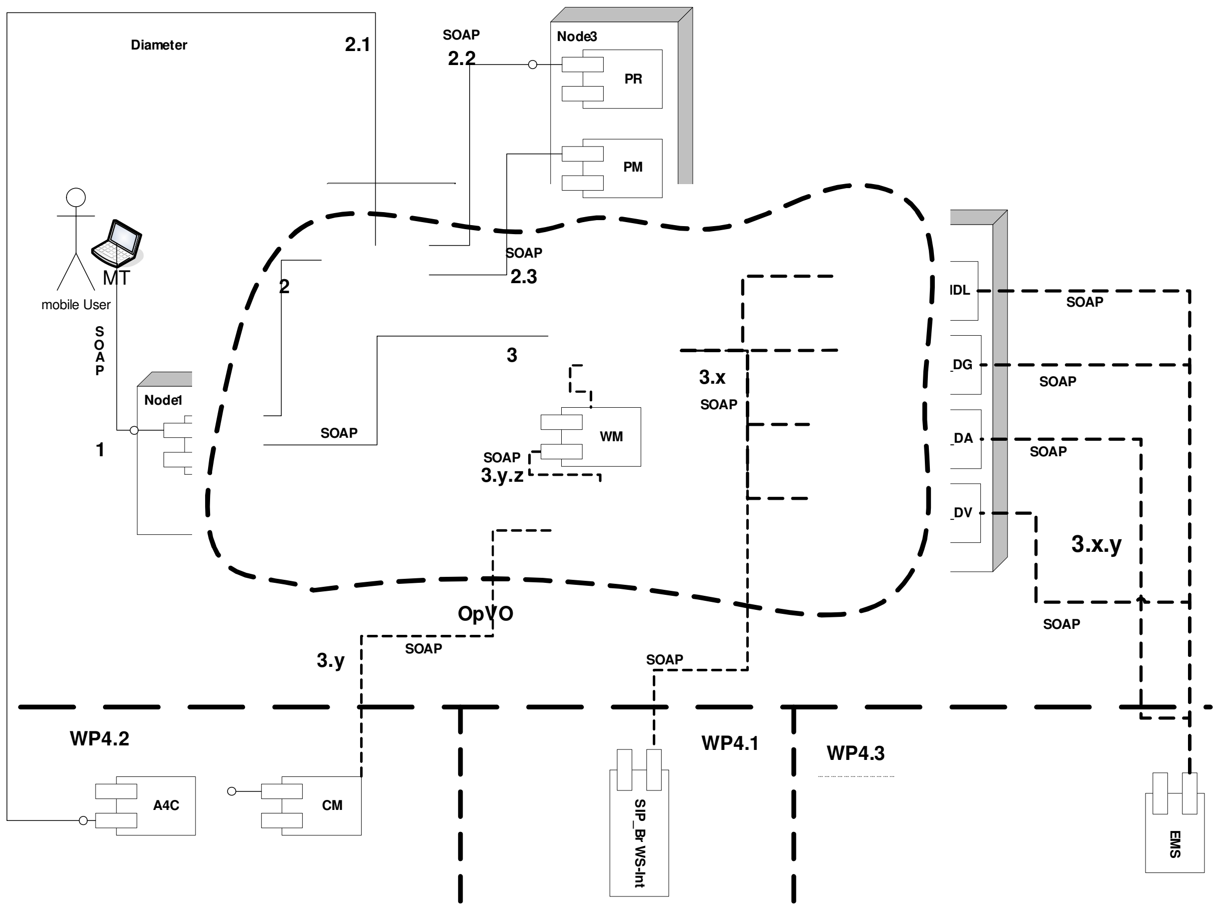


Figure 8 – A mobile user uses the OpVO

The meaning of each interaction[8] labelled with a sequence number is the following:

1. The mobile user (MU) has logged on the network and he/she has a reference to a list of UA. The MU chooses an application User Agent (UA) from the set of agents available for him/her. He/she will invoke the UA using the associate token and specifying the method and parameters to be invoked.
2. The UA invokes the OPVOm to validate the token received by the MU as part of the SOAP message headers
    a. OpVOm asks the A4C server for authenticating the incoming request
    b. If the authentication is successful, then the OpVOm calls the PR to retrieve the role associated to the identity of the requestor
    c. The OpVOm retrieves from the PM the authorization rights associated to the role and takes an authorization decision.
3. If the validation is successful the UA will invoke the method on a specific service inside the OpVO. The UA will have a list of possible services to be invoked and it will select the right one on the basis of the request coming from the MU. In the specific case, it will invoke the service exposed by the Enactment engine to start the workflow execution.
    3.x [9]During the workflow execution the engine will invoke the SAs that will act inside the OpVO on behalf of the external services involved in the workflow.

[8] On each connection between two components the protocol used for communication is specified. Most of interactions are established using SOAP over HTTP.

Furthermore, it will invoke the SIP Broker WS-Interface, as well if a SIP call is required.

i. The Service Agents (SAs) will forward the request to the EMS running in different administrative environment (the one of the Service Provider hosting the business service). In Figure 8, just one EMS is shown but several EMS can be involved depending on the result of negotiation phase (see section 2.2.2.2).

3.y. [10]During the workflow execution the Monitoring Daemon (MD) will receive notification from the Context Manager (CM) about a context change (or also other kind of events). The Enactment Engine will be informed about these changes through the WM and the workflow execution will be adapted depending on the event.

This is the operational behaviour of the GASS layer prototype and it is similar to the behaviour shown by the first prototype. As already explained for the first prototype, depending on the workflow script deployed in the Enactment Engine the steps 3.x and 3.y will be different.

With respect to the first prototype, the main new available feature is the dynamic OpVO creation phase. In fact, running an OpVO does not require for performing an offline setup as in the first prototype. In this release, the OpVO creation has been automated in order to support an execution close to a real test bed environment.

[9] This specific numbering (3.x and after 3.x.y) has been used to remark that the sequence of invocations is not known a priori but depend on the workflow design (they are linked to the Figure 8). Then several invocations can be associated to 3.x and for each of them a related invocation 3.x.y will be sent to the EMS

[10] Similar considerations can be done for 3.y numbering. Actually it is not known a priori how many and when context changes will happen.

3. Final prototype GASS services

This section outlines in detail the implemented components for the final prototype. The components have been implemented using two major software stacks as agreed within Activity 4 and WP5.1. Table 3 and Table 4 list the components involved in each stack.

Table 3 – Akogrimo Unix software stack

Unix/Java/WSRF Software Stack	
Operating System	Linux Ubuntu
Programming language	Java running in runtime environment for Linux
Web Server	Apache Tomcat 5
SOAP engine	Axis
WS-Resource framework	GT4 core

Table 4 - Akogrimo Windows Software Stack

	Windows/.NET/WSRF Software Stack	Windows/.NET/WSRF/ Enterprise stack
Operating System	MS Windows 2003 Server	MS Windows 2003 Enterprise Edition
Programming language	C# running on .NET Framework 1.1	C# running on .NET Framework 1.1
Web Server	IIS 6.0	IIS 6.0 Tomcat 5.0.28
SOAP engine	Microsoft WSE 2.0 SP3	Microsoft WSE 2.0 SP3
WS-Resource framework	WSRF.NET v2.1	WSRF.NET v2.1
XML Database		Xindice 1.1b4

Finally in the following subsections for each software module will be described the interface. Each method will be identified with the notation "E-X-Y-Z":

E = External interface

X = Service/group name

Y = Subservice of X

Z = Method name

For example the external method GetApplicationList belonging to BVO manager service that is part of the VO management service group will be identified with: E-BVO-VO-GetApplicationList.

3.1. VO management subsystem

3.1.1. BVO Manager

3.1.1.1. Brief Overview

The Base VO Manager is the key part of the policy and authorization enforcement at the top level of the Base VO. The Manager interacts as a "mediator" between the participant's registry, A4C server and external requestors. All service requests to the VO pass through the manager and are validated against the A4C server and participant's registry. Depending on the reply from the A4C server the Base VO manager either refuse or allow the request entering the Base VO. If access is granted the Base VO Manager creates an OpVO Manager and starts the process to create a new OpVO. Finally, the Participants registry is updated with the new participant's credentials. To aid this authorization in the BaseVO the service will incorporate the authorization mechanism PERMIS [6].

3.1.1.2. Functionality

Table 5 provides a list of the available methods on each service. The methods have been grouped in categories.

Table 5 - BVO manager service methods

BVO manager	
Incoming request management	
Description	This group includes methods to manage the incoming requests for joining to the BVO.
E-BVO-VO-GetApplicationList	
TokenAgentType[] getApplicationList(String token, String participantId)	
Description	The BVO Manager receives a request for the list of applications that this participant (identified by the participantID) is allowed to execute. The 'token' parameter is ignored for the time being. The return value is an array of pairs, consisting of the End Point Reference (EPR) of a UA and the associated OpVO token to invoke this UserAgent.

3.1.1.3. Interactions with other components

3.1.1.3.1. Main behaviour

The BVO manager interacts with (see also Figure 8):

- The User Agent

- The A4C (component outside the WP4.4)
- The PR

In the following Figure 9 the sequence diagram of these interactions is shown.

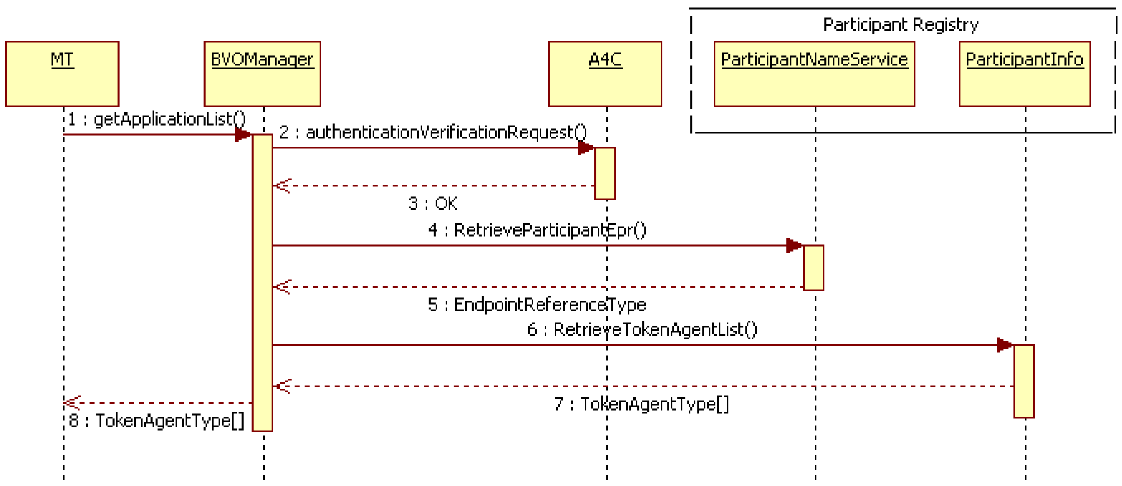


Figure 9 - BVO Manager interaction sequence

The above sequence provides details about the interaction related to step 1 in Figure 8 (in particular, steps 1, 1.1, 1.2)

In order to understand the meaning of each invocation, refer to the table of functionalities in the section related to the specific service. (e.g. for BVO Manager see section 3.1.1.2).

For information about A4C see [3].

3.1.1.3.2. Alternative behaviour

Unexpected and exception related behaviours of the BVO Manager can be caused by

- Calls to a method using wrong data types.
- Failure to provide correct authentication credentials when communicating with an interface.

These are the two behaviours we have encountered through both human and machine error, and purposeful use of incorrect data. All errors are logged.

3.1.1.4. Involved technologies

The implementation of this component comprises of the logic necessary to interact with the A4C server and participant registry (see Figure 9), using the return value in order to provide the requestor with specific token allowing him to invoke the application. This component has been implemented using the Akogrimo Unix/Java Software Stack and PERMIS (see the official web site for details [6]).

3.1.2. OpVO Manager

3.1.2.1. Brief Overview

The Operative VO Manager (OpVO Manager) takes the role of the VO manager in terms of authentication and authorisation of service calls to the VO during the execution of application specific services from the service provider domain within an Akogrimo application. Service providers once they have been authenticated by the Base VO, receive a token that can be validated by the OpVO manager during the lifetime of the Akogrimo application it is valid for. At this stage the token is essentially a set of textual assertions.

At the moment the tokens are simple strings and the tokens are passed every time as a part of the SOAP message headers. The User Agent is also authenticated using this process when calling the OpVO to invoke an application execution process, and to check the credentials of users who are part of the OpVO.

3.1.2.2. Functionality

Table 6 provides a list of the available methods on each service. The methods have been grouped in categories.

Table 6 – OpVO manager service methods

OpVO manager	
Validation of Invocation to the OpVO	
Description	This group includes methods to manage the incoming requests that ask for a service inside the OpVO.
E-OpVO-checkToken	
Boolean [] checkToken (String token, String participantId)	
Description	The OpVO manager will receive a request from a User Agent to check if an external user is authenticated and authorized to perform the required action in the OpVO, this request is accompanied by the token of the UA that will certifies if the UA is allowed to invoke the checkToken method on the OpVO Manager. The message will be encoded using pre-shared keys, to avoid man in the middle interception.

3.1.2.3. Interaction with other components

3.1.2.3.1. Main behaviour

The OpVO manager interacts with the UA. In Figure 10 the sequence diagram of this interaction is shown.

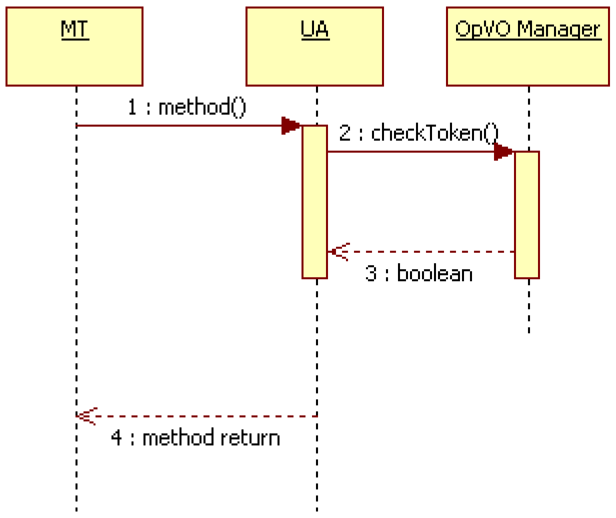


Figure 10 - OpVO Manager interaction sequence

The above sequence provides details about the interaction related to step 3 in Figure 8

In order to understand the meaning of each invocation, refer to the table of functionalities in the section related to the specific service. (e.g. for OpVO Manager see section 3.1.2.2). In step 1, method() is each method exposed by the UA.

After the step 3 in Figure 10 the UA will continue to execute the method (interacting with other components not shown here, because out of the scope of this sequence) and at the end (step 4) the result of method elaboration will be returned to the MT.

3.1.2.3.2. Alternative Behaviour

Unexpected and exception related behaviours of the OpVO Manager can be caused by

- Calls to a method using wrong data types.
- Failure to call the correct OpVO instance.
- Authentication errors.

These are the two behaviours we have encountered through both human and machine error, and purposeful use of incorrect data. The OpVO instance error sometimes is caused a failure to destroy existing instances of OpVO's that have been used. All errors are logged.

3.1.2.4. Involved technologies

The implementation of this component has dealt with logic to validate the token passed by the UA (see Figure 10). This component has been implemented using the Akogrimo Unix/Java stack.

3.1.3. User Agent

3.1.3.1. Brief Overview

This service belongs to VO Management topic and represents a user inside the OpVO. Moreover it is in charge to manage secure accesses to the application. The User Agent provides a standard front-end to be invoked from the outside. The OpVO user always will invoke the methods on the UA that will generate, in an automate way, the invocation of the right service inside the

OpVO. The UA exposes a fixed interface but depending on the input parameters passed to the invokeApplication method (see section 3.1.1.2), it is able to invoke a Web Service without knowing it in advance. This capability introduces a relevant automation in the frame of Akogrimo, because the platform provides a single generic service that allows invoking all the services available inside the BVO and OpVO. This is particularly important in the OpVO because it is not possible to know in advance the interfaces to invoke the workflow, and the UA provides the features to avoid to implement ad hoc client for each different workflow, because it exposes a fixed interface and create dynamically (at run time) the client to invoke the web service exposed by the workflow.

The service is implemented as a WSRF service, in this way, it will possible to have a dedicate UA instance for each different external user.

3.1.3.2. *Functionality*

3.1.3.2.1. *Interfaces*

Table 7 provides a list of the available methods on the UA. The methods have been grouped in categories.

Table 7 – User Agent methods

UserAgent	
Application methods	
Description	This group includes methods called from outside the BVO/OpVO to ask for a service inside the BVO/OpVO
E-UA-InvokeApplication	
public arrayOfXmlElement InvokeApplication(string methodName, arrayOfXmlElement inputMethodParameters)	
Description	This method allows starting the execution of a specific method on a service inside the OpVO. The OpVOToken is passed in the SOAP message. This token is extracted and elaborated by the SOAP message filter configured to guarantee the access to the User Agent instance only by authenticated and authorized requestors. It returns the object with invocation results or null.
E-UA-GetMethodList	
public string[ ] GetMethodList()	
Description	This method allows to an external invoker to retrieve the list of services available for the invocation inside the OpVO. For each service the returned list contains the name of the public methods available to OpVO members.
Configuration methods	

UserAgent	
Description	This group includes methods to be invoked in order to configure the User Agent. For example it is necessary to provide the services available in the OpVO, to set security policies…
E-UA-Configuration	
Public void Configuration(string serviceURL, string methodName, string[] xmlInputParametersFormat)	
Description	This method allows configuring the UA with the information about the services (endpoint and methods name) it can invoke in the OpVO.

3.1.3.3. *Interaction with other components*

3.1.3.3.1. *Main behaviour*

The UA interacts with (see also Figure 8):

- The MT
- The OpVO Manager
- The EEngine

Figure 11 shows the sequence diagram of these interactions:

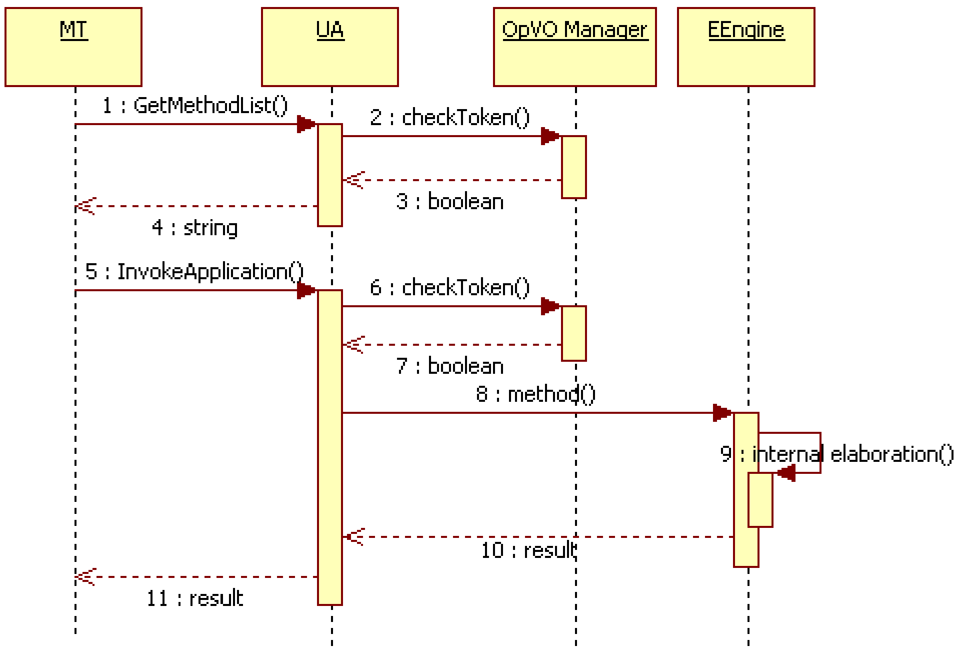


Figure 11 - UA interactions sequence

The above sequence provides details about the interaction related to step 4 in Figure 8

In order to understand the meaning of each invocation, refer to the table of functionalities in the section related to the specific service (In general, it is the section 3.x.y.2). In step 8, method() is a method exposed by the workflow instance in the specific demonstration scenario.

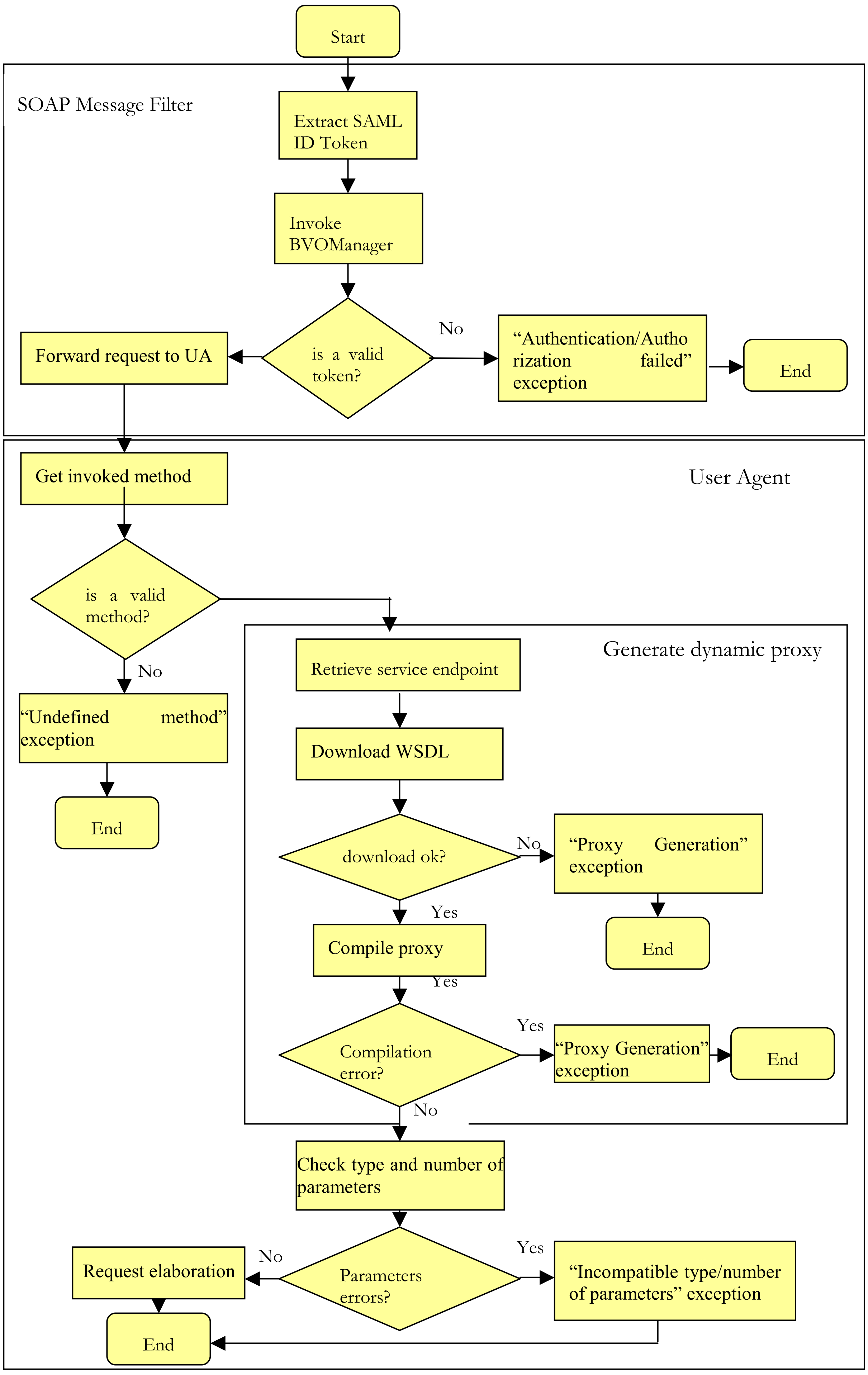
3.1.3.3.2. Alternative behaviour

The source of exceptional behaviour can be:

- Invocation timeout
- Authentication/Authorization failed
- Undefined method
- Dynamic proxy generation error
- Incompatible type/number of parameters

Invocation timeout exception is generated when the UA invokes the destination service but it doesn't receive a response in a defined time interval. The connection with the destination service is closed and an exception is forward to the requestor.

The other exceptions can be generated during the elaboration of the incoming request by the UA as shown in the following flow chart.


Start
SOAP Message Filter
Extract SAML ID Token
Invoke BVOManager
is a valid token?
No
"Authentication/Authorization failed" exception
End
Forward request to UA
User Agent
Get invoked method
is a valid method?
No
"Undefined method" exception
End
Generate dynamic proxy
Retrieve service endpoint
Download WSDL
download ok?
No
"Proxy Generation" exception
End
Yes
Compile proxy
Yes
Compilation error?
Yes
"Proxy Generation" exception
End
No
Check type and number of parameters
Parameters errors?
No
Request elaboration
End
Yes
"Incompatible type/number of parameters" exception

3.1.3.4. Involved technologies

The implementation of this component has dealt with the logic to generate, at run time, starting from a WSDL document, the proxy[11] code that will allow invoking the Web Service associated to the WSDL document (the WSDL document is retrieved at run time depending on the input parameters). This component has been implemented using the Windows/.NET/WSRF software stack.

The service is implemented using Web Service technology and in particular it will work with SOAP over HTTP. Furthermore it is able to elaborate the SOAP messages interacting with the SOAP pipeline in order to extract the VOToken from the related SOAP headers.

3.1.4. Service Agent Factory

3.1.4.1. Brief Overview

This service belongs to VO Management topic and represents a service inside the OpVO. Moreover, it is in charge of creating at runtime a Service Agent service (creation and deploy of a simple web service) used as gateway between OpVO domain and SP domain. Furthermore it creates service agent EPR and set information on created service agents.

At this stage the service is implemented as a simple WS.

3.1.4.2. Functionality

The Service Agent Factory is a component used to create in a dynamic way service agent service inside the OpVO domain. This component is invoked (CreateInstance method) by OpVOManager and performs the following actions:

1. create any service someone ask it
    a. create virtual directory on IIS to deploy created WS
    b. create physical directory on the file system (actually only the same where is deployed FactorySAService service), if not exists
    c. create all needed code files describing WS to deploy. To create the main WS file (*nameservice.asmx.cs*) is used information contained in a config file (ServiceConfig) passed by who ask the service creation.
    d. compile created code
    e. Runtime service aggregating (specific WSRF.NET task)
2. create resource to associate to the service
    a. uses information (EMS url and Reservation EPR, that is the unique identifier of the service instance produced by EMS at reservation time, which service agent will be associated) passed by who ask the service creation.

So it is needed to pass to CreateInstance method of FactorySA service two input parameters:

1. ServiceConfig that is a string containing an xml document where we have some information needed to create a specific service agent. This information is:
    - *Name service* contains the name of the service agent

[11] Proxy is synonymous of stub

- *Methods* contains one or more method that will be exposed by service agent. For each method is mandatory to specify the method name and input and output parameter type.

```
<?xml version="1.0"?>

<description xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

      <service name = "Generator"/>

      <methods>

            <method name="generateData">

                  <param_input type="string"/>

                  <param_output type="string"/>

            </method>

            <method name="HelloWorld">
```

2. ResourceInfo is a structure that contains the information that will be maintained in the SA resource. This information is composed by:
   - The EPR to the AdvanceReservation resource that was created at the end of the reservation phase. This EPR has been serialized into a string and is used as parameter when invoking the EMS service;
   - slaID string used as parameter when invoke EMS service;
   - EMS URI to use to set up EMS proxy

As return value of CreateInstance method has an EndpointReferenceType describing Service Agent service of which is asked creation. The following figure show actions made by FactorySA service when CreateInstance method is invoked.

3.1.4.2.1. Data structure

No data structure is available for this service.

3.1.4.2.2. Interfaces

Table 8 provides the list of the available methods on the Service Agent. The methods have been grouped in categories.

Table 8 - Service Agent methods

ServiceAgentFactory	
Instance Creation	
Description	This group includes one method to create the service inside the OpVO
E-FactorySA-CreateInstance	
EndpointReferenceType CreateInstance(string ServiceConfig, Information ResourceInfo)	

Description	Allows to create the SA service by using serviceConfig param and its EPR by using ResourceInfo param

In Table 8, the method belonging to "Instance Creation" allows creating a Service Agent, but the methods belonging to "Application" category will be specific to each Service Agent.

For example, the SA associated to the ECG_DG external service will expose the following application methods:

Table 9 - Application methods exposed by the SA of the ECG_DG

ECG_DG ServiceAgent	
Application methods	
Description	This group includes all the methods that are a replication of the methods exposed by the external ECG Data Generator service represented by this SA inside the OpVO.
E-SA-ECG_DG-GenerateData	
public data generateData (string patientName)	
Description	This method has the same external behaviour as the one described in the Application Specific Service section (see section 3.4.2)

3.1.4.3. Interaction with other components

3.1.4.3.1. Main behaviour

For this testbed, the FactorySA interacts with:

- OpVOManager

Figure 12 shows the sequence diagram of these interactions.

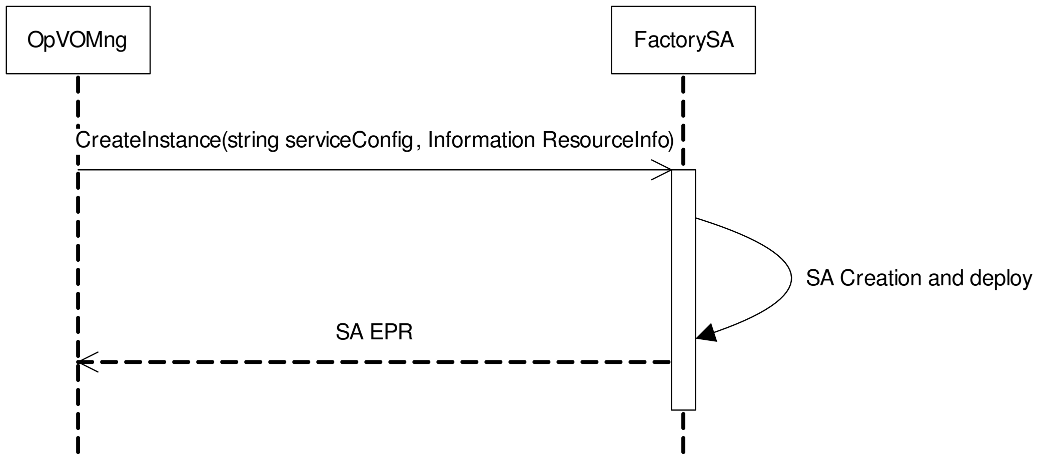

OpVOMng
FactorySA
CreateInstance(string serviceConfig, Information ResourceInfo)
SA Creation and deploy
SA EPR

Figure 12 - FactorySA interactions sequence

3.1.4.3.2. Alternative behaviour

No recovery actions are foreseen for this component

3.1.4.4. Involved technologies

The FactorySA has been implemented using Windows/.NET/WSRF software stack.

3.1.5. Participant Registry

3.1.5.1. Brief Overview

The Participant Registry service is actually constituted by three services that provide different kinds of information on the BVO/OpVO and their participants. These services are:

- **Participant Info**: this service will contain and manage all information related to participant features in the context of BVO/OpVO. This service will allow creating a WS-Resource associated to each participant and storing information related to the specific participant. In fact, it is able to manage information related to each different participant about:
  - Tokens
  - Roles
  - User Agent EPRs
  - Application and BVO/OpVO Manager identifier
- **VO info**: This service will manage information related to VO (BVO or OpVO) about its participants and its manager identifiers (name identity and endpoint reference). There will be a WS-Resource for each BVO/OpVO storing the mentioned information
- **ParticipantNameService**: This service will manage the mapping table in order to maintain an association between participant identifier and the endpoint reference of the resource created for him/her into the ParticipantInfo. By retrieving the endpoint reference it is possible to access to the participant related information. This service is a simple Web Service.

3.1.5.2. Functionality

Table 10 provides a list of the available methods on each service. The methods have been grouped in categories.

Table 10 - Participant Info service methods

Participant Info	
Tokens and agent management	
Description	The methods in this group manage the list of pairs, OpVOtoken and related User Agent EPR, associated to each participant and stored in the WS-resource that represents that represents them.
E-PR-ParticipantInfo-InsertTokenAgent	
public string InsertTokenAgent(string agentEpr, TokenType token)	
Description	Insert a token and an UA EPR to access a service in an existing participant WS-Resource. The result is the token identifier if operation is successful, otherwise "failure"
E-PR-ParticipantInfo-UpdateTokenAgent	
public bool UpdateTokenAgent(TokenAgentType oldEntry, TokenAgentType newEntry)	
Description	Updates a pair token and UA EPR of a participant with a new one.
E-PR-ParticipantInfo-UpdateToken	
public bool UpdateToken(TokenAgentType oldEntry, TokenType newToken)	
Description	Updates data in a token. Return true or false.
E-PR-ParticipantInfo-UpdateAgent	
public bool UpdateAgent(TokenAgentType oldEntry, string newAgent)	
Description	Updates UA data, in particular, the EPR of the UA. Return true if the update is successful, false otherwise
E-PR-ParticipantInfo-RemoveTokenAgent	
public bool RemoveTokenAgent(TokenAgentType entry)	
Description	Removes a token and its UA EPR associated to a participant Returns true if operation is successful, false otherwise

Participant Info	
E-PR-ParticipantInfo-RetrieveTokenList	
public TokenType[] RetrieveTokenList()	
Description	Retrieves the tokens' list of the participant Returns token type list if has success, null object otherwise
E-PR-ParticipantInfo-RetrieveAgentList	
public EndpointReferenceType[] RetrieveAgentList()	
Description	Retrieve the UA EPRs' list of the participant Returns endpoint reference type list if has success, null object otherwise
E-PR-ParticipantInfo-RetrieveTokenAgentList	
public TokenAgentType[] RetrieveTokenAgentList()	
Description	Retrieves the list of tokens and UA EPRs associated to the participant It returns the full list if successful, null object otherwise
Profile Management	
Description	The methods in this group manage the profiles[12] associated to each participant and stored in the WS-resource that represents it (so each method invocation doesn't need to specify the participant identifier: each WS-Resource on which the method is invoked is associated to a specific participant). Each participant can have different profiles each of one related to an OpVO where he/she has involved in.
E-PR-ParticipantInfo-AddProfile	
public string AddProfile(ParticipantProfileType participantProfile)	
Description	Adds a profile for the participant Returns the profile identifier if operation is successful, "failure" otherwise
E-PR-ParticipantInfo-UpdateProfile	
public bool UpdateProfile(ParticipantProfileType oldProfile, ParticipantProfileType newProfile)	

[12] The participant profile has not been used in the first prototype. The Participant Registry has been designed and implemented in order to manage it. In A.1, some notions are provided about what a profile is.

Participant Info	
Description	Updates an existing profile of the participant It returns true if the operation is successful, false otherwise
E-PR-ParticipantInfo-RemoveProfile	
public bool RemoveProfile(ParticipantProfileType profile)	
Description	Removes a participant profile It returns True if the operation is successful, false otherwise
E-PR-ParticipantInfo-RetrieveProfile	
public ParticipantProfileType RetrieveProfile(string profileId)	
Description	Retrieves a profile of the participant It returns the profile associated to the identifier if operation is successful, null object otherwise
E-PR-ParticipantInfo-RetrieveProfileListId	
public string[] RetrieveProfileListId()	
Description	Retrieves the list of profile identifiers associated to the participant Returns one or more profiles associated to the participant
Property management	
Description	This group of methods allows to manage the properties associated to each participant. Participant properties include: identifier of BVO Manager where the participants is registered, identifier of OpVOs where participant is involved, identifiers of application owned by the participant, …
E-PR-ParticipantInfo-ConfigureParticipantProperty	
public bool ConfigureParticipantProperty(ParticipantProperty prop)	
Description	Configures property of the participant. It returns true if the configuration is successful, false otherwise.
E-PR-ParticipantInfo-UpdateParticipantProperty	
public bool UpdateParticipantProperty(ParticipantProperty oldProp, ParticipantProperty newProp)	

Participant Info	
Description	Updates properties of a participant It returns true if the update is successful, false otherwise.
E-PR-ParticipantInfo-RetrieveParticipantProperty	
public ParticipantProperty[] RetrieveParticipantProperty()	
Description	Retrieves participant properties Returns participant properties array if operation is successful, null otherwise
E-PR-ParticipantInfo-RemoveParticipantProperty	
public bool RemoveParticipantProperty(ParticipantProperty prop)	
Description	Removes property of participant Returns True if deletion is successful, false otherwise
E-PR-ParticipantInfo-SetParticipantIdentifier	
public bool SetParticipantIdentifier(string id)	
Description	Set identifier of participant Returns True if setting is successful, false otherwise
E-PR-ParticipantInfo-GetParticipantIdentifier	
string GetParticipantIdentifier()	
Description	Returns the participant identifier

Table 11 – VOInfo service methods

VOInfo	
Participant reference management	
Description	This group of methods manage the reference of participant inside a VO. For each VO we will have a dedicated WS-Resource and it will store information about the participants. (so each method invocation doesn't need to specify the VO identifier: each WS-Resource on which the method is invoked is associated to a specific VO).
E-PR-VOInfo- AddParticipantReference	

VOInfo	
public bool AddParticipantReference(ParticipantMapping partRef)	
Description	Allows to add participant reference information in particular the participant id and the EPR of the related resource of the participant info service It returns true if the operation is successful, false otherwise
E-PR-VOInfo-UpdateParticipantReference	
public bool UpdateParticipantReference(ParticipantMapping oldPartRef, ParticipantMapping newPartRef)	
Description	It updates participant reference information It returns true if the operation is successful, false otherwise
E-PR-VOInfo- RetrieveParticipantEpr	
public EndpointReferenceType RetrieveParticipantEpr(string partId)	
Description	It allows to retrieve the participant endpoint reference Returns the participant endpoint reference if successful, null otherwise
E-PR-VOInfo- RemoveParticipantReference	
public bool RemoveParticipantReference(ParticipantMapping partRef)	
Description	Remove participant reference information Returns true if the operation successful, false otherwise
BVO/OpVO Manager setting	
Description	This groups includes methods that allow to set OpVO/BVO related information on the WS-Resource that represents the OpVO/BVO itself
E-PR-VOInfo-SetVOIdentifier	
public bool SetVOIdentifier(string id)	
Description	Set the VO (BVO or OpVO) identifier returns true if the operation is successful, false otherwise
E-PR-VOInfo-GetVOIdentifier	
public string GetVOIdentifier()	

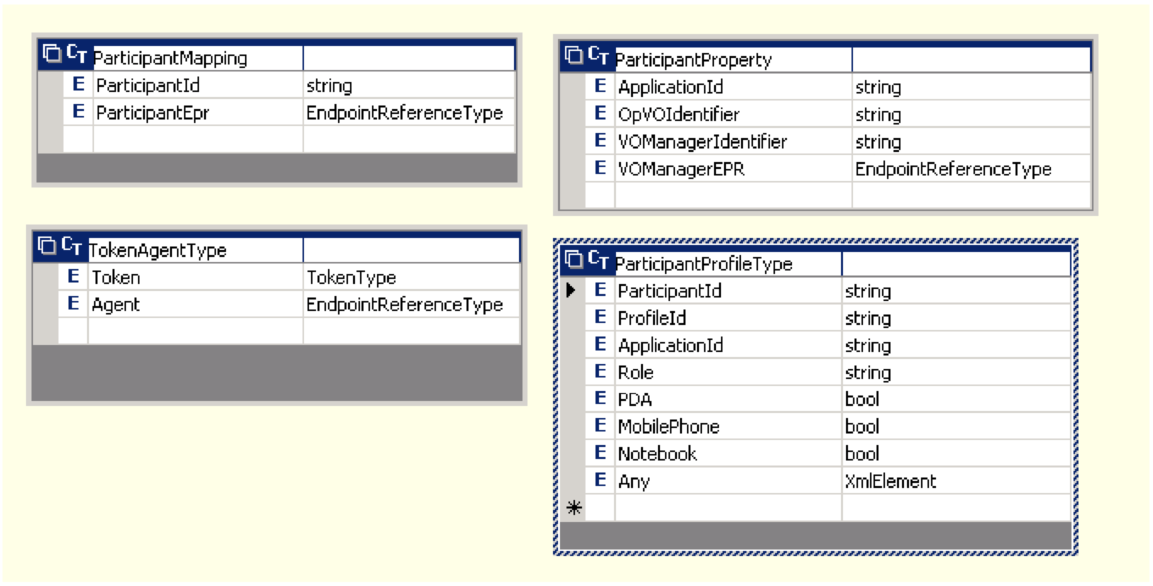
VOInfo	
Description	Get the OpVO/BVO identifier Returns the identifier if the operation is successful, "failure" otherwise
E-PR-VOInfo-SetManagerIdentifier	
public bool SetManagerIdentifier(string id)	
Description	Set the (BVO or OpVO) Manager identifier Returns True if the operation is successful, false otherwise
E-PR-VOInfo-GetManagerIdentifier	
public string GetManagerIdentifier()	
Description	Get the (BVO or OpVO) Manager identifier Returns the identifier if the operation is successful, "failure" otherwise
E-PR-VOInfo-SetManagerEpr	
public bool SetManagerEpr(string manEpr)	
Description	Set the (BVO or OpVO) Manager EndpointReferenceType Returns true if the operation is successful, false otherwise
E-PR-VOInfo-GetManagerEpr	
public string GetManagerEpr()	
Description	Get the (BVO or OpVO) Manager EndpointReferenceType Returns the EPR (serviceUrl#resourceID) if the operation is successful, "failure" otherwise

Table 12 – Participant Name Service methods

ParticipantNameService	
Mapping table management	
Description	This group of methods manage a table that stores the mapping between unique identifier of the participant and the EPR of the associated participant resource in the Participant info service
E-PR-ParticipantNameService-CreateParticipantMappingCollection	
public bool CreateParticipantMappingCollection()	

ParticipantNameService	
Description	Create the collection to map participant identifiers (or unique user name) and related endpoint reference It returns true if the collection is created right, false otherwise
E-PR-ParticipantNameService-CheckCollectionExistence	
public bool CheckCollectionExistence()	
Description	Check if the collection '/db/Akogrimo/ParticipantMapping' is already created in Xindice Returns true if the collection exists, false otherwise
E-PR-ParticipantNameService-RemoveParticipantMappingCollection	
public bool RemoveParticipantMappingCollection()	
Description	Removes the collection used to maintain mapping between participant identifier and their endpoint reference Returns true if the operation is successful, false otherwise
E-PR-ParticipantNameService-AddParticipantMappingEntry	
public string AddParticipantMappingEntry(string participantId, EndpointReferenceType epr)	
Description	Adds an entry inside the mapping table Returns Participant identifier if the operation is successful, false otherwise
E-PR-ParticipantNameService-RemoveParticipantMappingEntry	
public bool RemoveParticipantMappingEntry(string participantId)	
Description	Removes a participant from the collection It returns true if the action is successful, false otherwise
E-PR-ParticipantNameService-RetrieveParticipantEpr	
public EndpointReferenceType RetrieveParticipantEpr(string participantId)	
Description	Retrieves a document from a collection Returns the document if the operation is successful, null otherwise

For each component explained in this section we show the data structure they use to manage the set of data required to manage BaseVO participants.



ParticipantMapping	
ParticipantId	string
ParticipantEpr	EndpointReferenceType

ParticipantProperty	
ApplicationId	string
OpVOIdentifier	string
VOManagerIdentifier	string
VOManagerEPR	EndpointReferenceType

TokenAgentType	
Token	TokenType
Agent	EndpointReferenceType

ParticipantProfileType	
ParticipantId	string
ProfileId	string
ApplicationId	string
Role	string
PDA	bool
MobilePhone	bool
Notebook	bool
Any	XmlElement

Figure 13 - Data structures related to Participant Registry

The VOInfo component uses an array of ParticipantMapping; for each BaseVO participant it contains a participant identifier and the EPR of the WSRF resource that represents this user in the BaseVO.

The ParticipantInfo component uses an array of the following structures:

- PartipantProfileType contains the profile of the participant to BaseVO
- TokenAgentType contains the token to be used when invoking an User Agent instance
- ParticipantProperty contains information about the VO the participant is involved in

3.1.5.3. Interactions with the other components

The PR interacts with (see also Figure 8):

- The BVO Manager
- An administrative client

The interaction with the BVO Manager has been described in 3.1.1.3.

The invocations related to the management methods (e.g. update, add and remove) come from the administrative client and they are not reported here

3.1.5.4. Involved technologies

The implementation of this component has dealt with logic underpinning the management of BaseVO members. It provides a registry which exposes functionalities to collect personal information about each member, (the username valid in the context of the BaseVO, his profile listing his role in the BaseVO) to store information (username, profile) about all members of the BaseVO. This component has been implemented using the Windows/.NET/WSRF/Enterprise stack.

3.1.6. OpVO Broker

3.1.6.1. Brief Overview

The OpVOBroker service is a component of VO Management block and it is important for enacting of OpVO activities.

It interacts with GrSDS component to look for available services and then selects the services which are able to satisfy the user's needs according to his/her profile.

On the base of this service list the OpVOBrokerInstance starts the negotiation phase with a specific service provider. In each service provider domain is a Negotiator service instance that is in charge to negotiate contract with OpVOBroker and returns it a list of information needed to use a purchased service.

3.1.6.2. Functionality

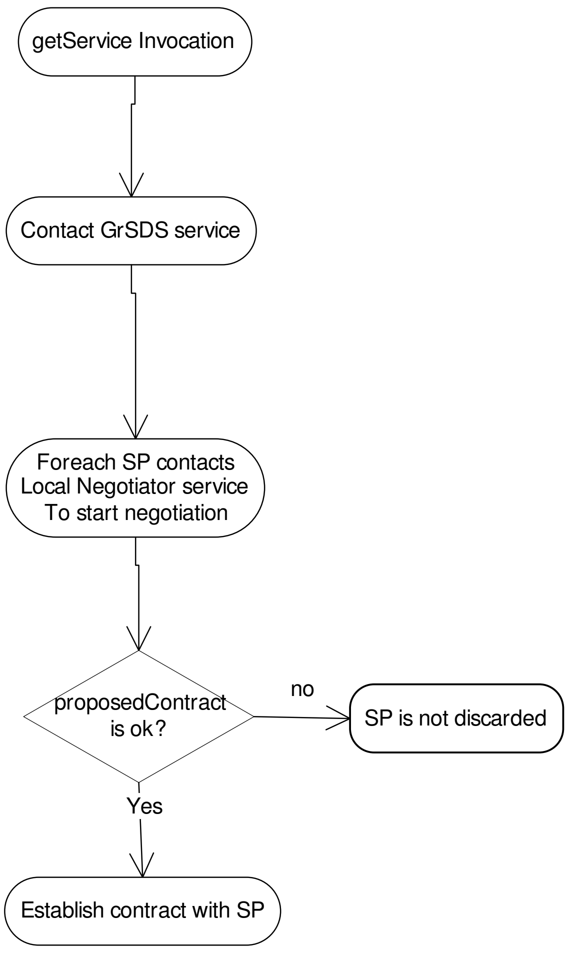

getService Invocation
Contact GrSDS service
Foreach SP contacts Local Negotiator service To start negotiation
proposedContract is ok?
no
SP is not discarded
Yes
Establish contract with SP

Figure 14 - OpVOBroker flowchart

The above flowchart describes what happens to OpVOBroker service when getService method is invoked. OpVOBroker is in charge of invoking GrSDS service to retrieve a list of Service Provider satisfying requirement used to make the search. In each SP domain there is a Negotiator service that is in charge of starting the negotiation process between the SP and the requestor. When Negotiator service is invoked, it returns to OpVOBroker a pre-Contract which has to be evaluated. If pre-Contract satisfies some constraints, OpVOBroker asks the Negotiator service to establish contract that will be returned to the requestor.

3.1.6.2.1. Data structure

No data structure is available for this service.

3.1.6.2.2. Interfaces

Table 11 provides a list of the available methods on the OpVOBroker. The methods have been grouped in categories.

Table 13 – OpVOBroker methods

OpVOBroker	
Discovery methods	
Description	This group includes methods called from outside to ask for a list of services satisfying requirements
E-OpVOBroker-getService	
public arrayOfXmlElement getService(XmlElement serviceRequirements)	
Description	This method allows starting a discovery of a services satisfying specific service requirements
Configuration methods	
Description	This group includes methods to be invoked in order to configure the OpVOBroker instance.
E-OpVOBroker-setEPR	
public void setEPR(EndpointReferenceType participantRegistryEPR, EndpointReferenceType policyManagerEPR)	
Description	This method allows configuring the OpVOBroker with the know EPR of the PolicyManager service and ParticipantRegistry service.

3.1.6.3. Interaction with other components

3.1.6.3.1. Main behaviour

The OpVOBroker interacts with (see also Figure 8):

- The GrSDS service
- The Negotiator service

In Figure 14 the sequence diagram of these interactions is shown.

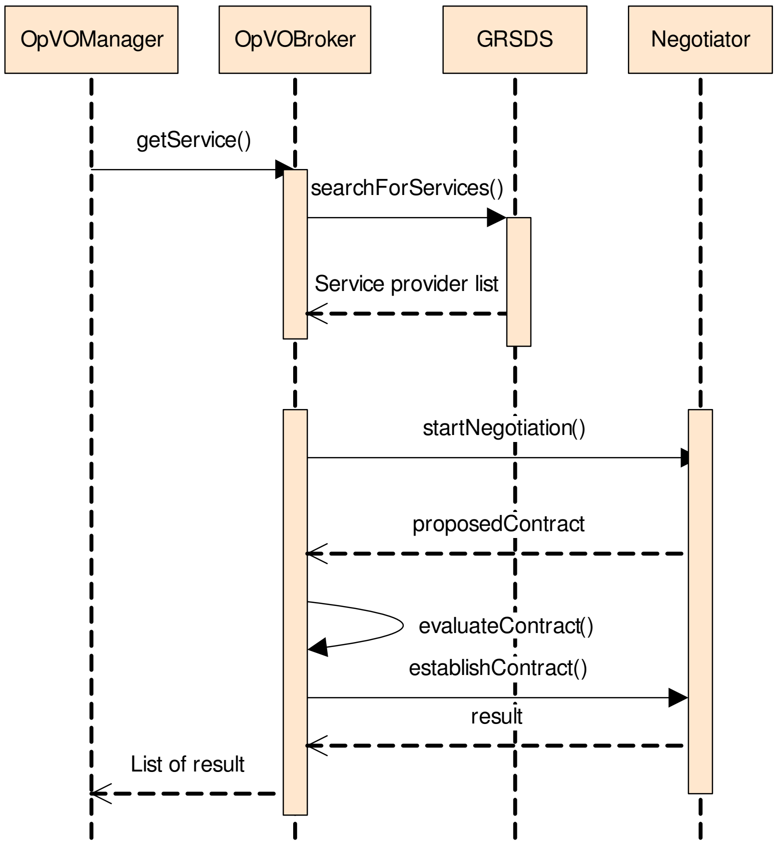


Figure 15 - OpVOBroker interactions sequence

3.1.6.3.2. Alternative behaviour

The OpVOBroker could find errors during the invocations of the external services, GrSDS service and Negotiator service. The source of exceptional behaviour can be the network connection between OpVOBroker and GrSDS or Negotiator. Network connection problems usually result in a time-out on the client side and could be handled by a second attempt or an increased time-out value. Increasing the time-out value for requests from the OpVOBroker to the GrSDS or Negotiator can be done by setting in the implementation code the timeout value of the proxy service (GrSDS or Negotiator) to infinite. No other recovery actions are foreseen for GrSDS service, instead for Negotiator service the error handling depends on the kind of error returned.

3.1.6.4. Involved technologies

This OpVOBroker component has been implemented using the Windows/.NET/WSRF software stack.

3.2. SLA High Level

3.2.1. SLA-Access

3.2.1.1. Brief Overview

This service provides all necessary functionalities that other components could require from the SLA document template and contract. Unlike SLA-Translator, SLA-Access does not access directly the documents stored in the repositories (SLA-Templates and SLA-Contracts), but it uses the SLA-Translator by passing the document Id and the specific tag it is looking for. So the SLA-Access contains the logic to retrieve and manipulate data from/to SLA template and contracts.

The SLA-Access has been implemented as a WS-Resource developed in C# (.NET) which exposes, at this moment, three methods (setParamToPreSLAContract, CreateLLContract and getQoSParameters). These methods offer to the consumer a way for accessing to concrete data of a specific document.

All these methods define what to do with concrete parts of the document, that is, to retrieve the service information, the description of the service, get some parameters, or for example to update the content of some tags, all of them carried out with a support service as the SLA-Translator.

3.2.1.2. Functionality

Table 14 provides a list of the available methods on each service. The methods have been grouped in categories.

Table 14 – SLA-Access service methods

SLA-Access	
SLA contracts and templates management	
Description	This group of methods allows to access the SLA templates/contracts and to modify them in a transparent with respect to the XML details.
E-SLA-Access-CreateLLContract	
`public string CreateLLContract(string userSPTemplateID, Low_level_parameter[] LLParameters,string expiration_time, string AREPR)`	
Description	It allows filling specific sections of SLA template, identified by userSPTemplateID, with the value of low level parameters that are specified in the LLParameters object. It returns a string if the operation has success, null object otherwise
E-SLA-Access-setParamToPreSLAContract	
`public string setParamToPreSLAContract(string UserSpTemplateID, int HLParams, string expirationTime)`	

SLA-Access	
Description	It allows filling specific sections of Pre SLA contract, identified by userSPTemplateID, with the value of high level parameters that are specified by HLParams object. It returns a string if the operation has success, null object otherwise
E-SLA-Access-getQoSParameters	
public objQoS getQoSParameters(string SLADocId)	
Description	This method is used to retrieve QoS parameters and thresholds to be monitored It returns an object with QoS parameters if the action has success, null object otherwise

3.2.1.3. *Interactions with the other components*

The SLA Access interacts with:

- The EMS that invokes it from the Grid Infrastructure Services Layer (actually any component interested in retrieving information on the SLA contract has to invoke the SLA Access). In the first prototype, the EMS is the only component that invokes the SLA Access.
- The SLA Translator

In the following Figure 16 the sequence diagram of these interactions is shown.

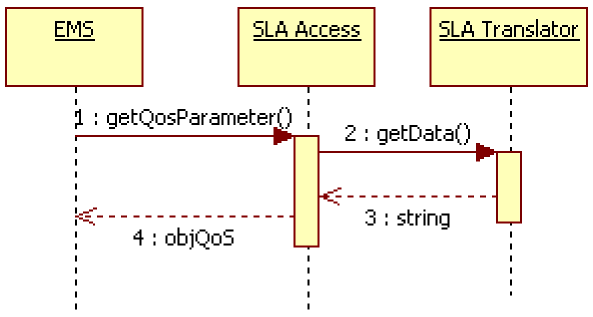


Figure 16 – SLA Access interactions sequence

In order to understand the meaning of each invocation, refer to the table of functionalities in the section related to the specific service. (In general, it is the section 3.x.y.2).

The interactions shown in Figure 16 take place during the Hosting Environment (HE) setup (see section 5.3.2 in [4]).

3.2.1.3.1. *Alternative behaviour*

The source of exceptional behaviour can be:

- Invocation timeout

- External Exception

Invocation timeout exception is generated when the SLA-Access invokes the SLA-Translator service but it doesn't receive a response in a defined time interval. The connection with the destination service is closed and an exception is forward to the requestor.

The external exceptions are generated by the SLA-Translator and then forward to the requestor without any manipulation.

3.2.2. SLA-Translator

3.2.2.1. Brief Overview

This service is the responsible for providing access to SLA documents (SLA-Template and SLA-Contract) and get (or set) information of them. For each document the associated ID is managed and SLA-Translator interacts with the repositories in order to retrieve the appropriate document.

The SLA-Translator has been implemented as a WSRF compliant service developed in C# (.NET) which exposes two methods (getData and setData) for getting/setting information from a document. There are only two methods offered to stress on the main use of this service, but each one has a set of input parameters which value have an influence on its results, so they can be seen as a generalization of many more methods.

This type of implementation allows choosing among different types of results just by changing the input parameters of both operations.

3.2.2.2. Functionality

Table 15 provides a list of the available methods on each service. The methods have been grouped in categories.

Table 15 – SLA Translator service methods

SLA- Translator	
SLA Contracts and Templates management	
Description	This group of methods allows processing the XML document that represents the SLA contract/template
E-SLATranslator- getData	
public string getData (string docType, string SLADocId, string rootTag, int occurrence)	

SLA- Translator	
Description	This method will be used for retrieving specific sections from a template or contract. In getData method, the consumer has to specify the type of document being requested ("Template" or "Contract") and the document ID, as well as the tag name involved in the operation by setting the simple tag name or a concrete root tag. Furthermore, it also has to specify how many occurrences it expects from the results that is choosing between the first occurrence found (1) / the first one and its siblings (2)./ all the occurrences (3). It will return a string that will represent the XML document containing all the information that matches the requirements in input.
E-SLATranslator- setData	
public string setData (string docType, string SLADocId, objUpdateTag objUpdate)	
Description	This method will be used for setting a specific tag in a template or contract. In setData method, the consumer has also to specify the type of document being requested ("Template" or "Contract") and the document ID. Furthermore, it also has to fill a specific object type with the data it wants to update. It returns a string containing the new ID of the updated document if the operation is successful, null otherwise

3.2.2.3. *Interactions with the other components*

The SLA interacts interacts with:

- The SLA Access
- The SLA Repository

Figure 17 shows the sequence diagram of these interactions.

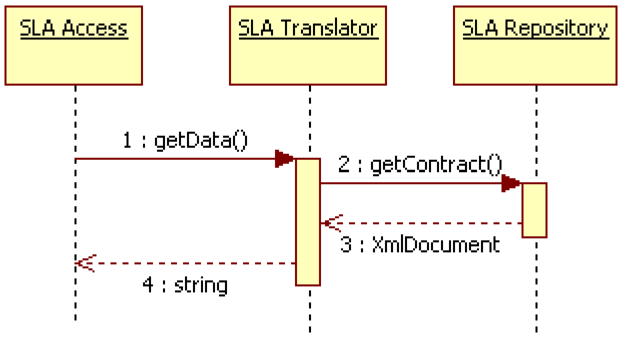


Figure 17 – SLA Translator interactions sequence

In order to understand the meaning of each invocation, refer to the table of functionalities in the section related to the specific service.

The interactions shown in Figure 17 take place during the HE setup (see section 5.3.2 in [4]).

3.2.2.4. Involved technologies

The SLA Translator implements the logic to retrieve a specific field required by the SLA-Access from the SLA document. Actually this logic is implemented through the couple: SLA-Access and SLA Translator.

It is implemented as a WS-Resource. There will be an SLA-Access instance associated to a specific SLA-Translator

Implementation has been done using the Windows/.NET/WSRF/Enterprise Software Stack.

3.2.3. Contract/Template-Repository

3.2.3.1. Brief Overview

The SLA Template Repository allows storing and retrieving of SLA documents. This service is used as a storage facility by the SLA-Translator.

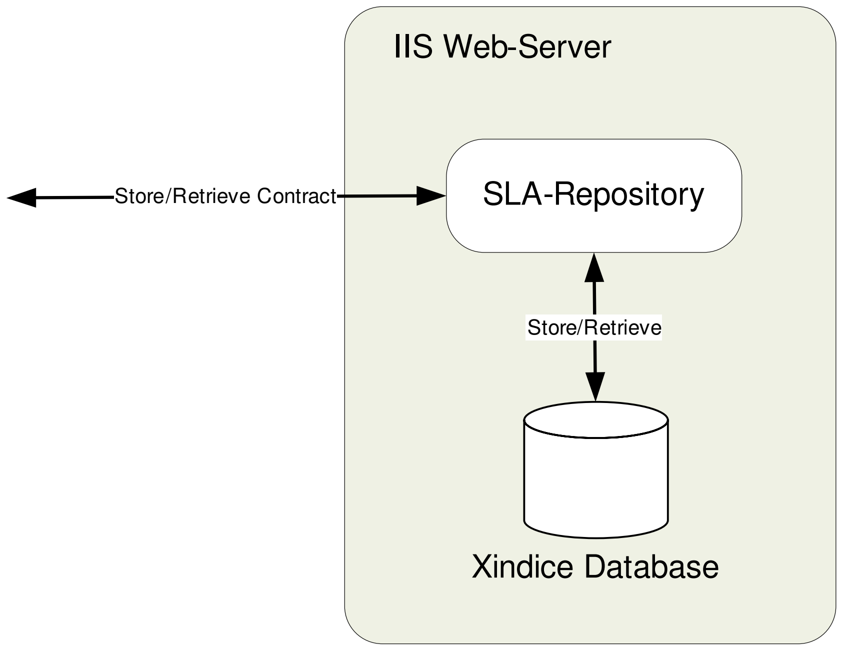


Figure 18 - SLA Contract and Template Repository

The SLA Template/Contract Repositories are currently implemented as two regular Web Services deployed in C# virtualizing a file system or storing documents in a XML database.

3.2.3.2. *Functionality*

Table 15 provides a list of the available methods on each service. The methods have been grouped in categories.

Table 16 - SLA repository service methods

SLA- Repository	
Storing SLA contracts and templates	
Description	This group of methods allows to store and retrieve templates and contracts from the repository
E-SLARepository- storeTemplate	
public string StoreTemplate(string templatePath, bool overwrite)	
Description	The StoreTemplate() method accepts a path to a local file (templatePath) and a parameter 'overwrite' that specifies if the method is allowed to overwrite existing SLA templates with the same name. Currently the name is used to identify the SLA template document.
E-SLARepository- GetTemplate	
public XmlDocument GetTemplate(string templateId)	

SLA- Repository	
Description	Given the name of an SLA Template as templateID, the method GetTemplate() retrieves the document and returns it as an XmlDocument
E-SLARepository- storeContract	
public string StoreContract (string contractPath, bool overwrite)	
Description	The StoreContract() method accepts a path to a local file (contractPath) and a parameter 'overwrite' that specifies if the method is allowed to overwrite existing SLA Contract with the same name. Currently the name is used to identify the SLA Contract document.
E-SLARepository- GetContract	
public XmlDocument GetContract (string contractId)	
Description	Given the name of an SLA Contract as ContractID, the method GetContract () retrieves the document and returns it as an XmlDocument

3.2.3.3. Interaction with the other components

The SLA Repository interacts with the SLA Translator. In general, any component interested in retrieving a contract and/or a SLA template has to invoke it.

The interaction has been already described in section 3.2.2.3

The methods related to storage functionalities have been used using dedicated client in order to store the XML document to be retrieved during the HE setup (see section 5.3.2 in [4]).

3.2.3.3.1. Main Behaviour

The regular behaviour consists of store, update and retrieve operations of SLA documents. No computational or otherwise complex internal operations are involved.

3.2.3.3.2. Alternative Behaviour

The source of exceptional behaviour can be one or more of the following:

- The network connection
- The IIS Web-server
- The .NET runtime environment
- The Xindice database (if SLA contracts are stored in the database)
- The file system (if SLA contracts are stored on disk)
- The implemented functionality

Network connection problems usually result in a time-out on the client side and could be handled by a second attempt or an increased time-out value. Problems of the Web-server or the .NET runtime environment are considered out of the scope of this document. So the main sources of exceptional behaviour are the database or hard disk and the implementation itself. All of these errors are reported to the client by mean of exceptions. Following diagram gives an overview of the exception handling control flow when SLA documents are retrieved from the repository.

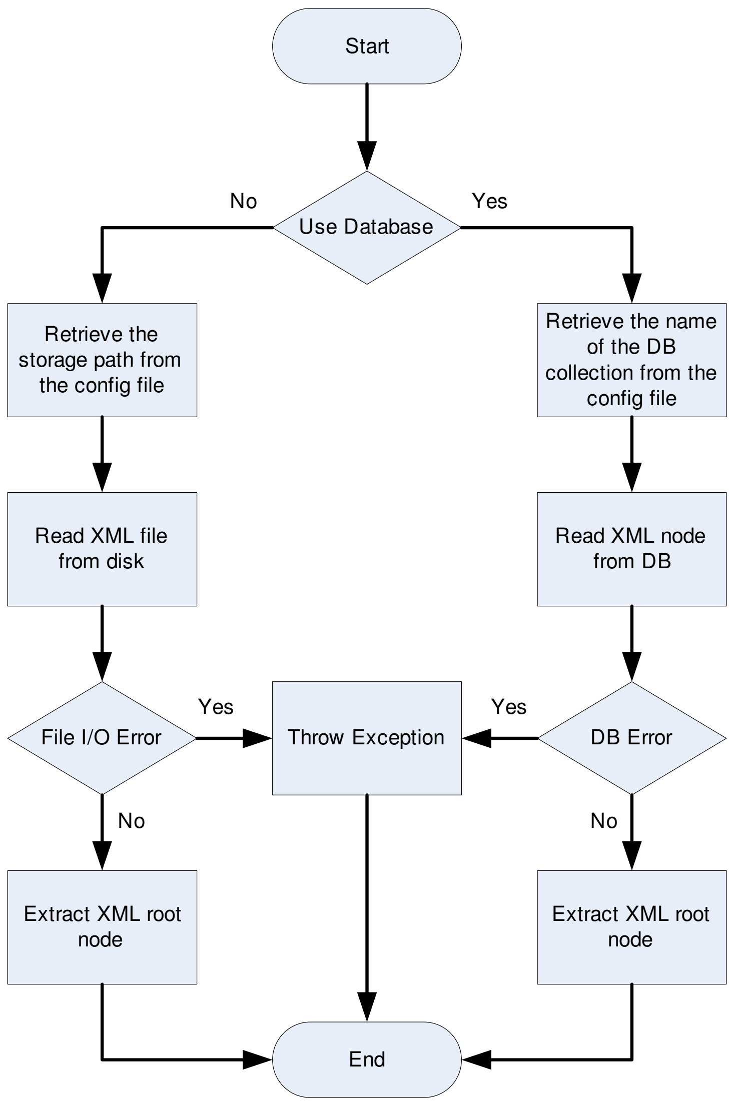


Figure 19 - SLA Document Read Control Flow

When documents are stored in the repository the control flow is slightly more complex as can be seen in the following diagram:

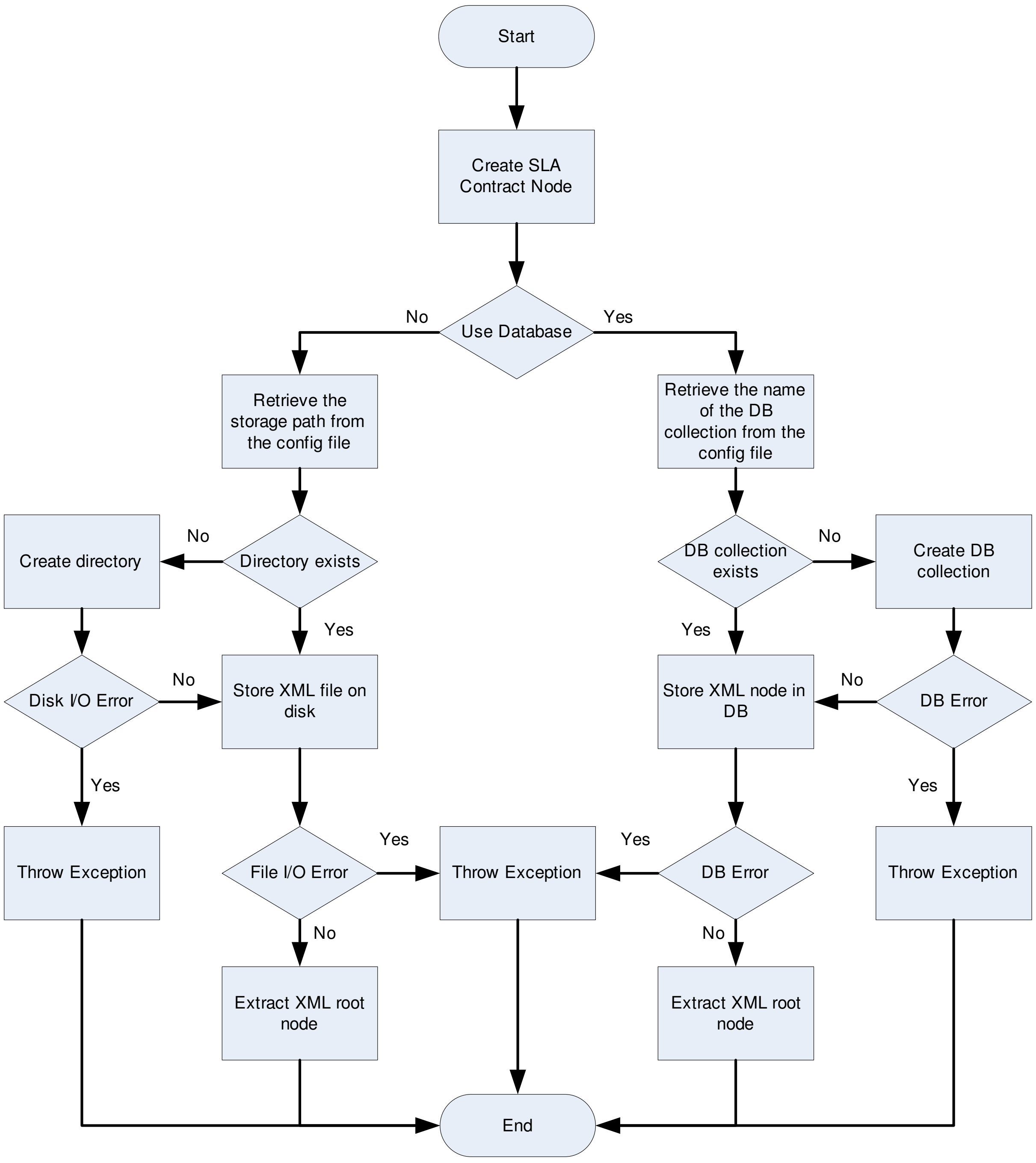


Figure 20 - SLA Document Write Control Flow

3.2.3.4. Involved technologies

The implementation of this component has dealt with the management of SLA documents. It provides a repository to archive these documents and exposes functionality to parse the content of the document and to retrieve specific information.

SLA Repository has been developed using regular Windows ASP.net Web service using C# as implementation language and it will represent a simple interface to a file system.

It is implemented using the Windows/.NET/WSRF software stack and the Xindice 1.1b4: native XML database.

3.2.4. SLA Negotiator

3.2.4.1. Brief Overview

SLA-Negotiator Service represents the service that is contacted by the Operative VO Broker component (OpVoBr) in order to lead the service negotiation process in the SP domain. The OpVoBr has previously requested to the Grid Service Discovery component the list of service providers that offer a particular service. Then the OpVoBr contacts the SLA-Negotiator service of the given service provider to initiate and lead the negotiation process. From an implementation point of view, SLA-Negotiator is a MultiResource Factory Service prepared for the co-existence of several Virtual Organisation working in parallel. That means that there is a service that exposes an operation to generate SLA-Negotiator service instances.

3.2.4.2. Functionality

Next, we include some tables in order to detail the functionality and implementation of the two operations exposed by SLA-Negotiator:

Table 17 – SLA-Negotiator methods

SLA-Negotiator	
Operations	
Description	This group includes methods called from OpVOBroker in order to perform service negotiation.
E-SLANG-StartNegotiation	
public String do_startNegotiation(String serviceID, String userID, int hlparam, String expirationTime, String UserSPTemplateId)	
Description	This method allows starting the negotiation of a given service for a given user with certain high level parameters. The aim of this operation is to establish a contract between the user and the Service Provider where the terms are expressed as high level functionalities. It is important to note that the information enclosed in this contract refers to high level requirements (i.e. refresh time less than X seconds) and not to low level parameters like cpuUtil and diskUtil. Negotiator will look up the resource availability and invoke the SLA-Access to store the Contract into the Repository with the high information parameter information.
E-SLANG-EstablishContract	
public String do_establishContract(String UserSPTemplateID, LLParams llParams, String expirationTime, EndpointTypeReference AREPR)	

SLA-Negotiator	
Description	Once agreed with the Contract including the high level requirements, SLA-Negotiator tells EMS to carry out the advanced reservation of resources to provide the agreed Contract. Finally, SLA-Negotiator stores the SLAContract, including the low level parameters, into the repository via SLA-Access to be available for the SLA-enforcement process once execution starts. It is important to remark that this contract includes the low level parameters that are going to be monitored by the SLA enforcement mechanism.

3.2.4.3. Interaction with other components

First, we include a table summarizing the components which SLA-Negotiator interact with as well as the protocol used:

Interaction with other components	Protocol
Policy Manager	SOAP/HTTP
EMS	SOAP/HTTP
SLA-Access	SOAP/HTTP

More detailed information is shown in the sequence diagrams including the interactions taking placed in both operations:

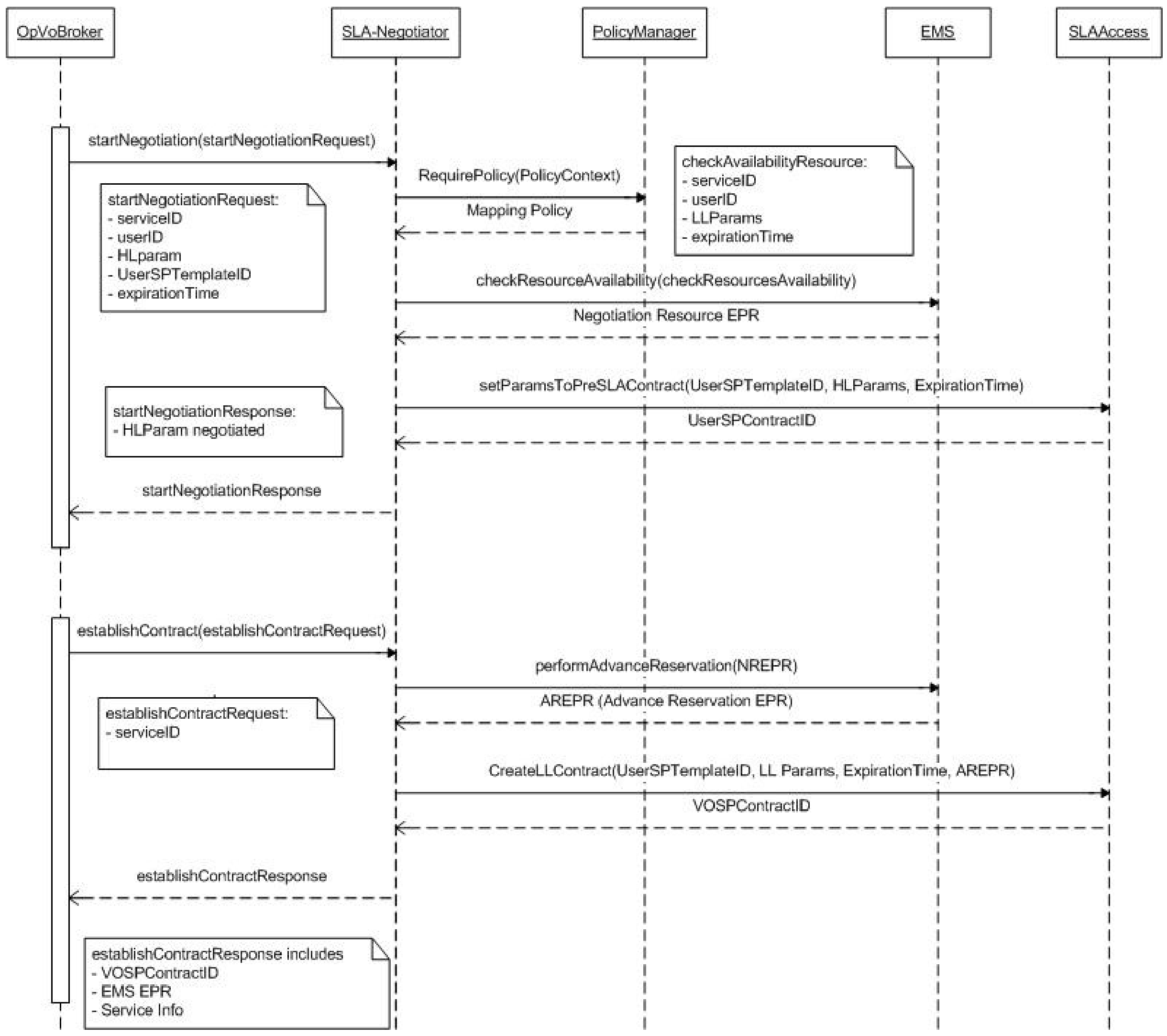

OpVoBroker
SLA-Negotiator
PolicyManager
EMS
SLAAccess
startNegotiation(startNegotiationRequest)
startNegotiationRequest:
- serviceID
- userID
- HLparam
- UserSPTemplateID
- expirationTime
RequirePolicy(PolicyContext)
Mapping Policy
checkAvailabilityResource:
- serviceID
- userID
- LLParams
- expirationTime
checkResourceAvailability(checkResourcesAvailability)
Negotiation Resource EPR
setParamsToPreSLAContract(UserSPTemplateID, HLParams, ExpirationTime)
UserSPContractID
startNegotiationResponse:
- HLParam negotiated
startNegotiationResponse
establishContract(establishContractRequest)
performAdvanceReservation(NREPR)
AREPR (Advance Reservation EPR)
establishContractRequest:
- serviceID
CreateLLContract(UserSPTemplateID, LL Params, ExpirationTime, AREPR)
VOSPContractID
establishContractResponse
establishContractResponse includes
- VOSPContractID
- EMS EPR
- Service Info

Figure 21 –SLA Negotiator interactions

As it can be seen in the sequence diagram, SLA-Negotiator interacts with Policy Manager in order to retrieve the low level parameters that corresponds to the high level parameters chosen by user. The mapping between the high and low level parameter is stored following a policy format. The object retrieved from Policy Manager is an XML document where this info is stored. Next, SLA-Negotiator checks with EMS the availability of resources with the obtained low level parameters from the Policy Manager. In the case that there are resources available, the EMS returns back the EPR of the Negotiation Resource. Finally in this first phase, SLA-Negotiator pushed SLA-Access to create the User-Service Provider contract where the high level parameters have to be included. The Low level information has been decided to be stored in a different contract since this information should be absolutely transparent to the user. For example, a general user only understands if its screen refresh time is less or greater than a given number of seconds, but he does not care about the resources values required to obtain such performance (cpuLoad, memoryUsage….).

The second operation exposed by SLA-Negotiator has the aim to generate and store a different contract that includes the low level parameter information. This contract is basic for the SLA enforcement process. Once the service starts running the EMS initiates the SLA enforcement mechanism to be sure that the user enjoys the service in the terms established in the contract (i.e. refresh time less than X seconds). The sequence diagram establishes that in the phase apart from creating/storing the new contract, SLA-Negotiator indicates EMS to perform advance reservation of resources.

3.2.4.3.1. Error handlings

So far, SLA-Negotiator interacts with different components (Policy Manager, EMS and SLA-Access) when OpVoBr invokes one of the two operations exposed by the Service (startNegotiation and establishContract). The sequence diagrams defined so far do not include the case where the EMS does not find any resource available with the current low level parameters. For this particular case, no recover mechanism has been defined so far.

Now, once EMS can not find resources available for the current HLParam, SLA-Negotiator will try with a high level parameter less restrictive. SLA-Negotiator will ask again EMS to search resources with less restrictive conditions. This process repeats until either EMS finds a resource available or there are no more high level parameters. In the latter case, SLA-Negotiator will inform OpVoBr to try to negotiate the service with another Service Provider or launch another mechanism.

Next we include the sequence diagram where the NULL management is depicted:

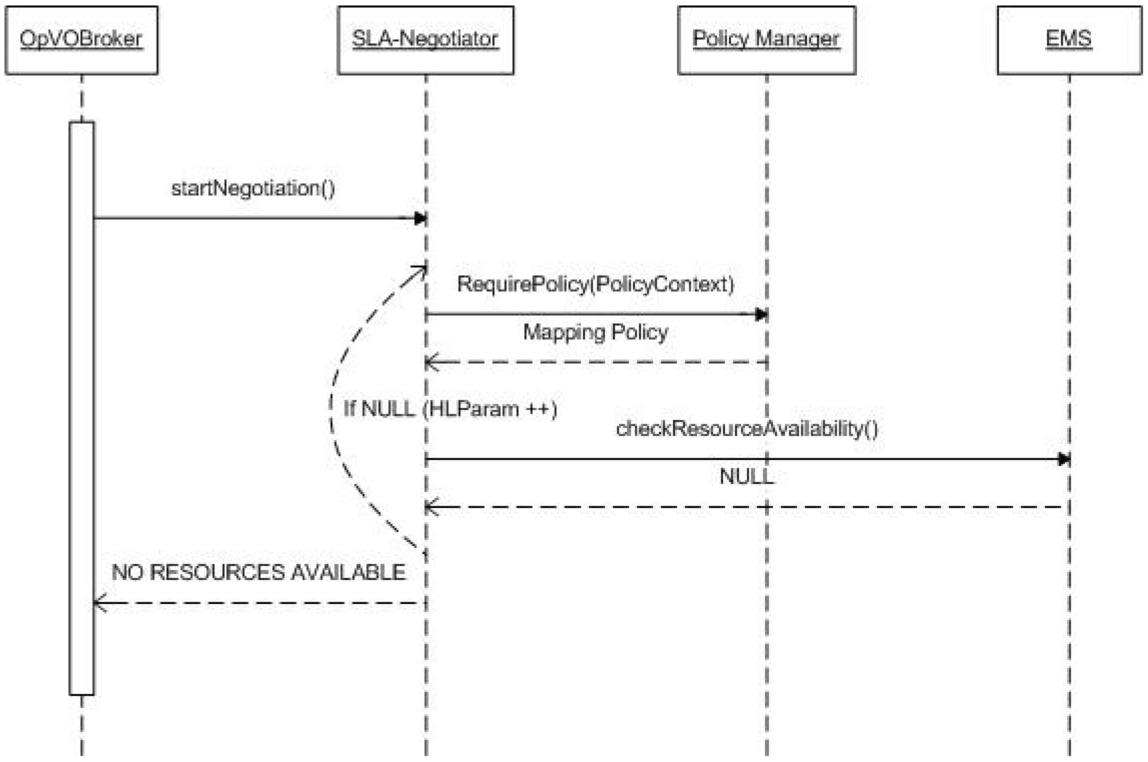


Figure 22 - SLA-Negotiator sequence diagram representing the handling errors case

3.2.4.4. Involved technologies

The development of this component involved the WSRF implementation provided by java WS-Core Container included in GT4.0.1.

3.3. BP enactment

3.3.1. WorkFlow Registry

3.3.1.1. Brief Overview

The Workflow Registry is the component in charge of storing implementation files of published workflows. Such files will be stored in relational databases, along with annotations necessary for semantically identifying workflows. A unique key identifies each workflow and its semantic annotations. Semantic annotations are stored as simple keywords.

The Workflow Registry contains Business Process Execution Language (BPEL) templates that correspond to the Business Processes. In addition the Workflow Registry contains what is called "Process Deployment Descriptor". This additional file is related to the workflow template(s) and it contains the name of the abstract services. The abstract services will be substituted with concrete services by the Workflow Manager. In this way the Workflow Manager component should not parse the entire BPEL script.

The Workflow Registry makes use of the MySQL relational database. It is exposed as a web service that offers methods to upload/retrieve templates. The web service is implemented in Java.

The template search could be done based on the unique workflow identifier or based on simple keywords.

3.3.1.2. Functionality

3.3.1.2.1. Data Structure

The Workflow Registry makes use of a MySQL database where information about workflow templates is stored.

The database has the structure reported in Table 18.

Table 18: Workflow Registry Data structure

Field	**Type**	**Brief Explanation**
Filename	String	File name of the workflow template.
TemplateID	String	Template identifier. This identifier coincides with the ones reported in the GrSDS.
SPDomain	String	Domain application of the Service Provider.
FileID	String	File identifier. This field differs from the template identifier. It is an internal ID used to physically retrieve the workflow template file.

3.3.1.2.2. Interfaces

Table 19 provides a list of the available methods on each service. The methods have been grouped in categories.

Table 19 - WF Registry service methods

WF- Registry	
Storing WF templates	
Description	This group of methods allows to store and retrieve WF templates to/from the repository.

WF- Registry	
E-BPE-WFRegistry- Store	
workflowID **store**(WFTemp[], filename, tempID, SPdomain)	
Description	It returns a workflow identifier assigned by database upon success. The parameters are: the workflow template files (BPEL and PDD), the filename to be associated to the wf template, the template identifier, the Service Provider domain.
E-BPE-WFRegistry- Get	
WFTemp[] **get**(workflowID)	
Description	It returns the workflow template (BPEL and PDD) files upon success. The input parameter is the workflow identifier.
E-BPE-WFRegistry-Update	
result **update**(workflowID, keywords[], values[])	
Description	It returns success or failure. The parameters are the workflow identifier, the keywords to be updated and their new values. The admitted keywords are: "temeplateID","Filename","SPDomain".
E-BPE-WFRegistry-Remove	
Result **Remove**(workflowID)	
Description	It returns success or failure. This function removes the specified workflow from the database. The parameter is the workflow identifier.

3.3.1.3. Interactions with other components

3.3.1.3.1. Main Behaviour

The Workflow Registry interacts with:

- The Service Provider
- The Workflow Manager

The Service Provider is in charge of storing the workflow template files to the Workflow Registry.

The Workflow Manager is in charge of retrieving the workflow template files from the Workflow Registry.

Workflow template files can updated/removed by the Service Provider.

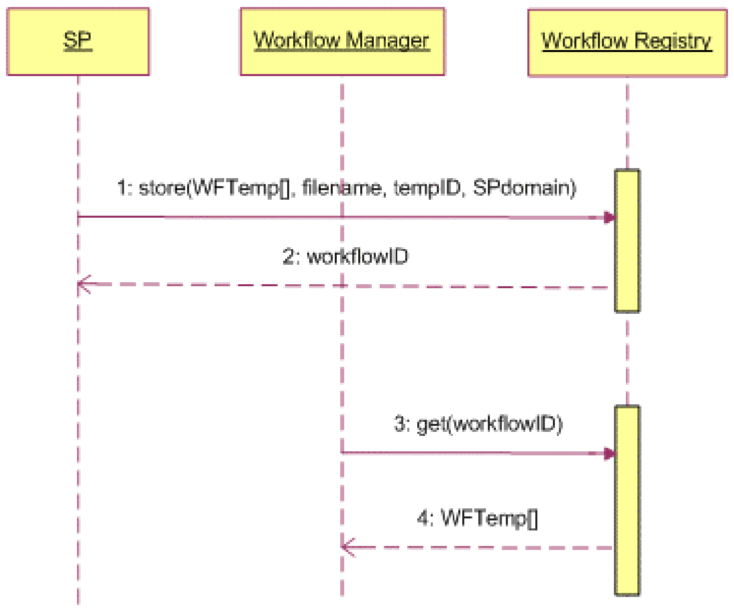


Figure 23: Workflow Registry Interfaces

In Figure 23 the sequence numbers outline that a workflow template could be retrieved after it has been uploaded. In any case the two actions do not need to be one just after the other.

3.3.1.3.2. Alternative behaviour

The possible causes of failures are:

- The filename is already present in the Workflow Registry
- The template identifier (templateID) does not exist
- The Workflow template file is corrupted

In the first two cases the request should be submitted again with the correct parameters. If the Workflow Template file is corrupted, the Workflow Registry returns an error that is reported to the upper layer, that is to the Workflow administration level. No automatic recovery is foreseen and the error should be communicated to the Service Provider.

3.3.1.4. Involved technologies

The implementation of this component has dealt with the management of WF related files stored in the associated relational database. This service implements the logic to associate the different files between them and it virtualizes the access to the database through a Web service interface.

Involved technologies:

- MySQL: Relational Database
- Basilar Web Service specifications (SOAP, WSDL)
- Java programming language
- Linux Ubuntu Operating system
- Tomcat: Web Service
- Axis: SOAP engine

3.3.2. Enactment Engine

3.3.2.1. Brief Overview

The Enactment Engine (also called Workflow Engine) is the component of the Business Process Enactor in charge of enacting specific BPEL processes submitted by the Workflow Manager component. The Enactment Engine will support execution of BPEL processes by calling services with a WSDL description.

The Enactment Engine will be exposed to external entities as a normal web-service to which invocation of methods will be possible. Methods will include those for starting and cancelling a business process, those for inspecting and retrieving inputs and outputs to/from the workflow and status information of a workflow.

3.3.2.2. Functionality

3.3.2.2.1. Data Structure

No data structure is available for this service.

3.3.2.2.2. Interfaces

Table 20 provides a list of the available methods on each service. The methods have been grouped in categories.

Table 20 – Enactment engine methods

Enactment- engine	
Workflow instances management	
Description	This group of methods allows to manage the workflow instances living inside the Enactment Engine
E-BPE-EEngine-deployBpr	
processID **deployBpr**(bpelFile)	
Description	This method is used for submitting a workflow to the engine for enactment. It is semantically equivalent to the method “workflowRef **Submit**(*wfTemplateFile*)“ described in section 3.2.2.4.3 of deliverable [2] It returns an identifier of the deployed workflow.
E-BPE- EEngine-resumeProcess	
bool **resumeProcess** (processID)	

Enactment- engine	
Description	This method is used for asking the engine to start the enactment of a submitted workflow. It is semantically equivalent to the method "**Start**(workflowRef)" described in section 3.2.2.4.3 of deliverable [2] It returns success or failure
E-BPE-EEngine-terminateProcess	
bool **terminateProcess (**processID**)**	
Description	This method is used for asking the engine to totally cancel the enactment of a submitted workflow. It is semantically equivalent to the method "**Cancel**(workflowRef)" described in section 3.2.2.4.3 of deliverable [2] It returns success or failure
E-BPE-EEngine-getprocessList	
process_list getProcessList**()**	
Description	This method is used for asking the engine the list of processes already submitted. Note that the resulting list includes all deployed processes, not only those actually executing.
E-BPE-EEngine-getProcessState	
process_state getProcessState**(processID)**	
Description	This method is used for asking state information about a workflow. It is semantically equivalent to the method "wf_state **GetWFState**(*workflowRef*)" described in section 3.2.2.4.3 of deliverable [2]
E-BPE-EEngine-setVariable	
process_variable setVariable**(processID, name)**	
Description	This method is used for setting value and state of a specific variable of a specific workflow. This method is very important because it used to set any context change happening during the workflow execution.
E-BPE-EEngine-getVariable	
process_variable getVariable**(processID, name)**	

Enactment- engine	
Description	This method is used for asking value and state of a specific variable of a specific workflow. It is semantically equivalent to the method "wf_variable **InspectVar**(*worfklowRef*)" described in section 3.2.2.4.3 of deliverable [2]
Business process logic related methods	
Description	This group will include all the methods that are strictly related to the business process logic. Of course, they will be different depending on the business process design itself, so they are not listed here, because they are application specific..

3.3.2.3. Interaction with other components

3.3.2.3.1. Main behaviour

The Enactment Engine interacts with:

- UA
- SA
- Monitoring Daemon
- WF Manager

In Figure 24 the sequence diagram of the above interactions is shown, in particular, it refers to the interactions with the business logic related methods.

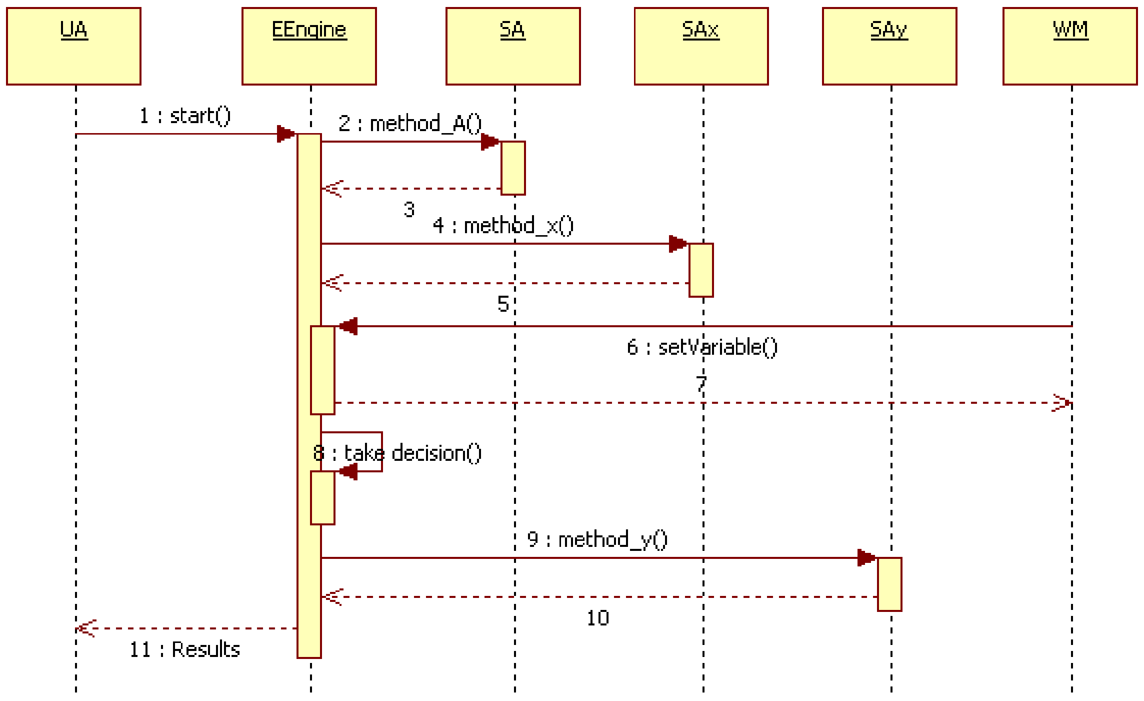


Figure 24 – Enactment Engine interactions sequence

The above sequence provides details about the interaction related to step 3 in Figure 8

In order to understand the meaning of each invocation, please refer to the table of functionalities in the section related to the specific service (sections 3.x.y.2). Actually, apart from the method invoked by the Workflow Manager, the other ones are generic names because they are related to the application logic, and during the workflow instance execution several methods on the application services are invoked. The sequence of interactions doesn't reflect a real temporal sequence because the invocation of SA depends on the business logic of the workflow.

The object that in Figure 25 is labelled with "EEngine" actually is the running instance of the application workflow (it is run by the EEngine).

The interaction with the Workflow Manager is performed during the OpVO creation to configure the Enactment Engine and it involves the management methods. The following sequence describes this interaction that involves an invocation from a generic client that actually during the OpVO creation is the OpVO Manager.

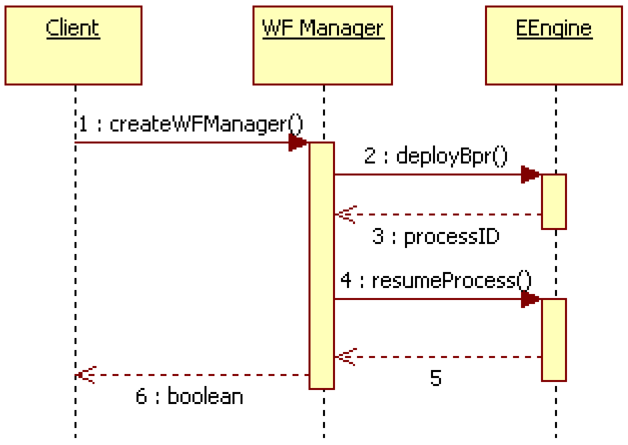


Figure 25 – Enactment Engine interactions sequence

3.3.2.3.2. *Alternative behaviour*

The Workflow Engine could encounter internal errors or can receive an error during the execution of the services it invokes.

In case of internal errors no recovery actions are foreseen unless the Business Process Designer has modelled the workflow to take care of them. No automatic recovery actions are foreseen.

In case of errors during the invocation of external services, the error handling depends on the modelling of the specific workflow. The Workflow Engine is able to catch errors thrown by the services and it is able to takes recovery actions like: new invocation, alternative branch execution.

3.3.2.4. *Involved technologies*

The implementation of this component has dealt with the logic to administer the ActiveBPEL Engine. Some API provided by the ActiveBPEL Engine has been used in the implementation and it has been implemented as a Web Service to simplify interactions with other components using Java as programming language.

Involved technologies:

- ActiveBPEL engine: BPEL script execution engine.

- JDK: 1.4.2_09-b05 (mandatory).
- Ant: 1.6.5 (mandatory).

3.3.3. WorkFlow Manager

3.3.3.1. Brief Overview

This is a component of the Business Process Enactment service (the other components being the Enactment Engine (EE), the Monitoring Daemon (MD) and the Workflow Registry (WR)). Its purpose is to transform *workflow templates* into workflows that can be carried out by the Enactment Engine; part of this transformation includes service instantiation and registration for context changes (with the Context Manager) and SLA violations (with SLA Enforcement). It also receives context/SLA notifications from the MD and (possibly) interrupts or redirects the current workflow. A Workflow Manager (WM) is created (by an OpVO Manager, or by another Workflow Manager); thus each instance of the WM manages a single workflow. (More strictly, it manages a single instance of a workflow template; this may generate multiple workflows.)

At this stage, the WF manager will implement just a limited set of functionalities, in particular related to the management of context change and the business process deployment.

3.3.3.2. Functionality

Table 21 provides a list of the available methods on each service. The methods have been grouped in categories.

Table 21 - WF manager service methods

WF- Manager	
Workflow instantiation	
Description	It includes a group of methods that allows to set up the environment to manage the workflow during its execution.
E-BPE-WM-CreateWF_Manager	
EndPointReferenceType CreateWF_Manager(WFTemplate)	
Description	Create a new Workflow Manager to handle a workflow template (supplied as parameter). It returns a reference to the new WF Manager instance.
E- BPE-WM-StartWorkFlow	
Boolean StartWorkFlow (WFTemplate, EPR)	
Description	Instruction to an Enactment engine to deploy and instantiate a workflow to start processing its template. In this phase all the setup actions will be performed (e.g. subscription of monitoring daemon to context changes). It returns a Boolean true or false statement.

3.3.3.3. Interactions with the other components

The WF Manager interacts with the Workflow Repository to extract the workflow template, and is calls the Enactment Engine to update context and deploy workflows.

3.3.3.3.1. Alternative Behaviours

Unexpected and exception related behaviours of the Workflow Manager can be caused by

- Calls to a method using wrong data types.
- Failure to call the correct Workflow Manager instance.

These are the two behaviours we have encountered through both human and machine error, and purposeful use of incorrect data. The Workflow Manager instance error sometimes causes a failure to destroy existing instances of Workflow Manager that have been used. All errors are logged.

3.3.3.4. Involved technologies

The component has been realised using the UNIX/Java/WSRF software stack and was particular using the WS-Notification implementation of GT4 Core.

3.3.4. Monitoring Daemon

3.3.4.1. Brief Overview

This is a component of the Business Process Enactment service (the other components being the Workflow Manager (WM), the Enactment Engine (EE) and the Workflow Registry (WR)). Its purpose is to provide a web service interface that the Context Manager and SLA Enforcement can use to notify BP Enactment about context changes or SLA violations. It may perform some processing on the received notifications (such as further filtering and (in the longer term) concept mapping), before passing them to the WM for decisions/ action.

3.3.4.2. Functionality

Table 21 provides a list of the available methods on each service. The methods have been grouped in categories.

Table 22 - Monitoring Daemon service methods

Monitoring Daemon	
MD Administration	
Description	It includes a group of methods that allows to administrate the MD
E-BPE-MD-Create	
EndpointReferenceType Create()	

Monitoring Daemon	
Description	Create a new Monitoring Daemon. It is expected to be invoked by a WM, so that each WM has (to all intents and purposes) a dedicated MD
E- BPE-MD-GetContext	
Public GetContext(userID)	
Description	To be used by a client to query the last known context for a user ID.

3.3.4.3. Interactions with the other components

The MD interacts with:

- The Context Manager
- The Workflow Manager

In Figure 26, the sequence diagram of these interactions is shown. The Workflow Maanger is not included in this sequence because it interacts with the MD in a different phase to instruct (see below). Furthermore the Workflow manager is invoked by the MD to be informed about context changes.

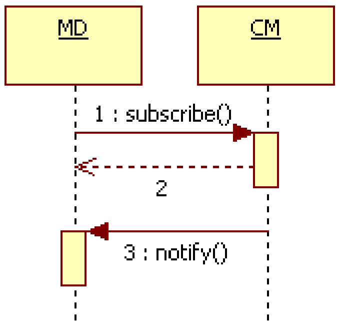


Figure 26 – MD interactions sequence

In order to understand the meaning of each invocation, refer to the table of functionalities in the section related to the specific service. (In general, it is the section 3.x.y.2).

It is worth mentioning that in order to subscribe to the CM the MD has to be instructed to do that. In the complete Akogrimo picture, the WF Manager will instruct the MD, during the OpVO creation, passing it the information about the context which it has to subscribe on. The MD calls the WF Manager to update context in workflows upon any significant context updates it receives from the workflow manager.

Details about the interactions between MD and CM can be found in [3] section 5.2.2.

3.3.4.4. Alternative behaviour

Unexpected and exception related behaviours of the Monitoring Daemon can be caused by

- Calls to a method using wrong data types.
- Registration of wrong context variable to user
- Error in update sensitivity

Passing wrong data types to the MD has been encountered through both human and machine error, and purposeful use of incorrect data. The registration of a user to the MD and subsequent registration with the context manager is a process that relies on good network connectivity. Failure at this stage causes the workflow manager to stall. Sensitivity of the RFID tag on the sensor has caused multiple context alerts being sent to the Monitoring Daemon in a short space of time. This has the effect of the MD overwriting previous updates and triggering multiple calls to the workflow engine in short spaces of time. This has the potential to negatively affect the behaviour of the workflow engine. All events and errors are logged in the service.

3.3.4.5. Involved technologies

Implemented using the UNIX/Java/WSRF software stack. Furthermore it uses the WS-Notification implementation available in GT4 core.

3.3.5. Sip Broker WS-Interface

3.3.5.1. Brief overview

This service does not represent a component itself, it is included in the Sip Broker component. A full description of the Sip Broker component more related to the SIP protocol can be found in official Akogrimo deliverable D4.1.3 (see [7]). Keeping this in mind, from now on we will refer to the WS-Interface as a component even when it is only a part of it.

This component aims to provide a WSRF service interface to the Sip Broker component available in the platform and explained in detailed in deliverables of WP4.1 and WP4.2. Among the functionalities offered by this service can be found

- Establish a sip call between two users
- Allow the transfer of an audio/video session from one device to other
- Obtain connection details of the mobile services running in different terminals
- Notify the change of the features of mobile services

This functionality is offered by means of two services: SipBroker WSRF service and the Sip Broker Notification Producer service.

3.3.5.2. Functionality

Table 23 details the signature of the operations exposed by Sip Broker WSRF service:

Table 23 - Sip Broker WSRF service

<u>**Sip Broker WSRF service**</u>	
Sip Broker WSRF Service	
Description	This group of methods allows communication with the Sip Broker component.

Sip Broker WSRF service	
E-BPE-SBI-startSession	
int **startSession**(user1,user2,sessionType,OpVOToken)	
Description	This method is used when a SIP Call between two akogrimo users is required. The variables user1 and user2 represent the identifiers of the Akogrimo users, sessionType represents the type of session (*application/sdp)* and the OpVOToken is a token to ensure that the request comes from the right Operative VO. It returns an int telling success (0) or failure and in case of failure, the type of failure.
E-BPE- SBI-transferSession	
int **transferSession** (user1,user2,device,sessionType,OpVPToken)	
Description	This method is used when a transfer of a session (between two users) has to be carried out from the current terminal to a new device. User1 and user2 are the Akogrimo users that are having an audio/video session, device represent the identifier of the new device where the session is going to be transferred, sessionType indicates the type of the session (in this case *application/sdp*) and OpVOToken is a token to ensure that the request comes from the right Operative VO It returns an int telling success (0) or failure and in case of failure, the type of failure.
E-BPE-SBI-getConnectionDetails	
Connection details **getConnectionDetails (**userID,serviceID**)**	
Description	This method is used for asking the connection details of a service (or all services) of a certain user. It returns a string which includes the URL at which the service can be found and the availability.

Table 24 provides the description of the operations provided by Sip Broker Producer Service

Table 24 – Sip Broker Producer Service

Sip Broker Notification Producer service	
Sip Broker Notification Producer Service	
Description	This service notifies the change on the connection details of mobile services to which EMS have been subscribed.

Sip Broker Notification Producer service	
E-BPE-SBI-subscribe	
void **subscribe**(topic)	
Description	This method is exposed in order to let other service to subscribe to some of its topics. This service only exposes a topic which informs the consumer service of the availability of mobile services. So far, EMS has to subscribe to this service to know online where the mobile services are located and their availability.
E-BPE- SBI-setState(state)	
void **setState** (state)	
Description	This method is used to update the topic and to provoke a notification to all consumers being subscribed to this topic. These topics include information of the availability and location of a mobile service associated to a certain Akogrimo user.

3.3.5.3. Interactions with other components

3.3.5.3.1. Main behaviour

The Sip Broker Interface service interacts with the following components:

- Enactment Engine
- EMS

The Enactment Engine interacts with the Sip Broker when it is required either to establish a SIP Call between two Akogrimo users or when a session transfer has to be carried out.

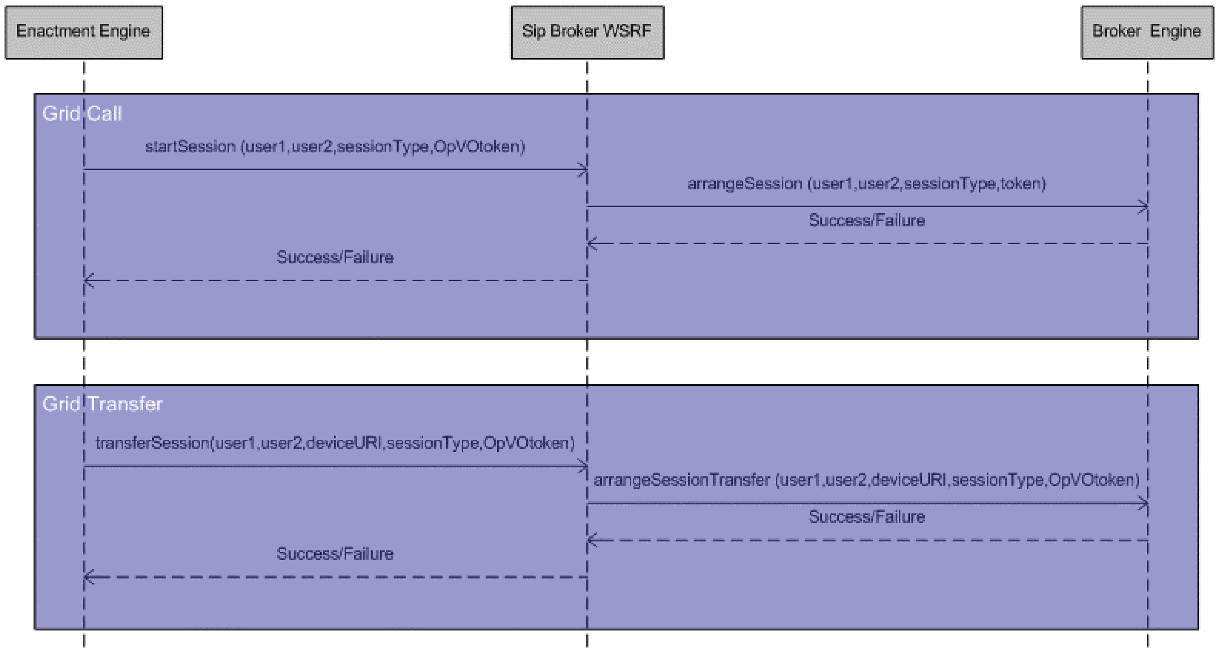

Enactment Engine
Sip Broker WSRF
Broker Engine
Grid Call
startSession (user1,user2,sessionType,OpVOtoken)
arrangeSession (user1,user2,sessionType,token)
Success/Failure
Success/Failure
Grid Transfer
transferSession(user1,user2,deviceURI,sessionType,OpVOtoken)
arrangeSessionTransfer (user1,user2,deviceURI,sessionType,OpVOtoken)
Success/Failure
Success/Failure

Figure 27 - Enactment Engine – Sip Broker WS-Interface sequence diagram

EMS needs information about mobile services. In this sense, Sip Broker is the service that is able to provide mobile services online. The information can be provided in two different ways:

- By invoking an operation directly (*getConnectionDetails* from Sip Broker WSRF)
- By subscription/notification mechanism (*Subscribe* from Sip Broker Notification Producer)

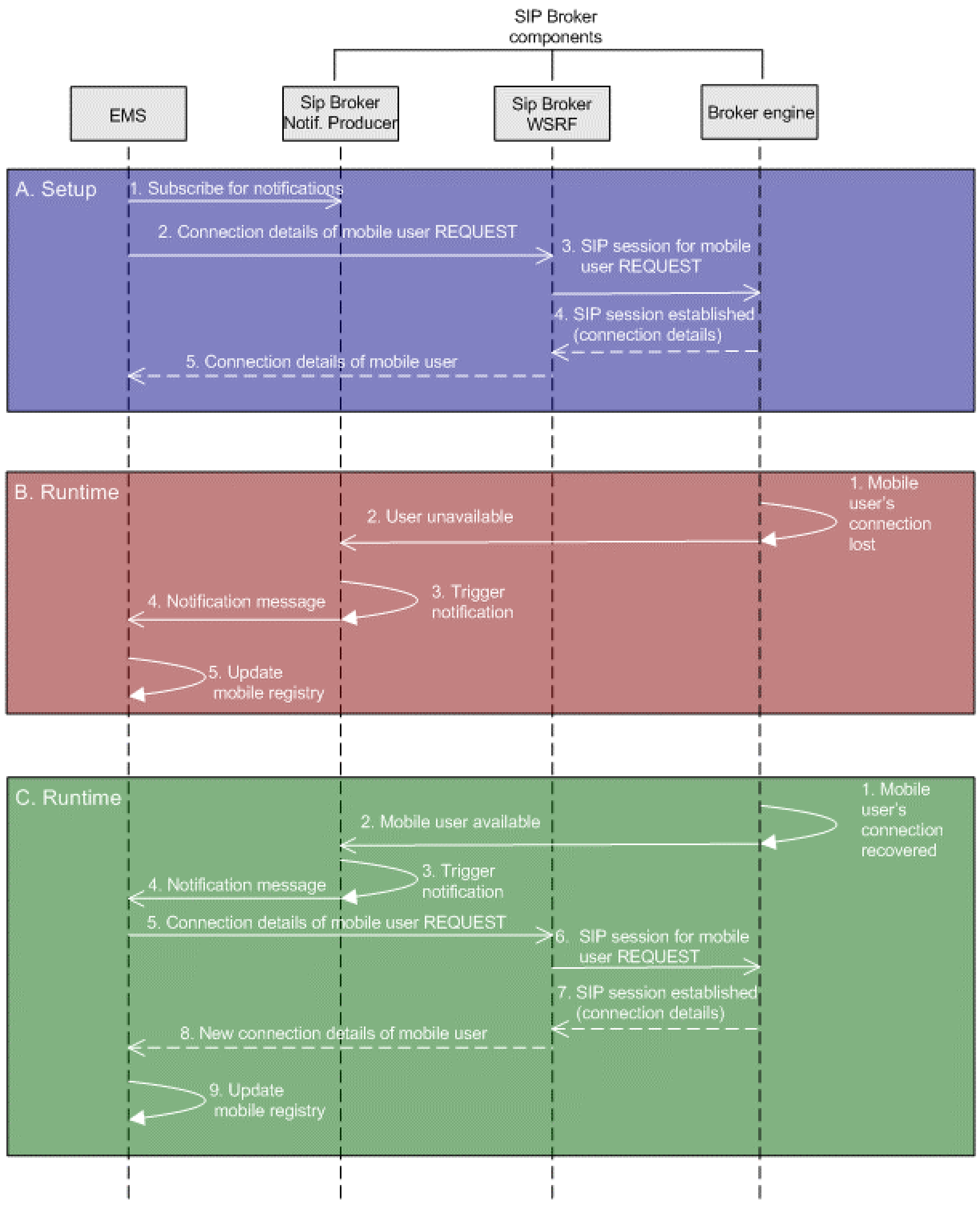

SIP Broker components
EMS
Sip Broker Notif. Producer
Sip Broker WSRF
Broker engine
A. Setup
1. Subscribe for notifications
2. Connection details of mobile user REQUEST
3. SIP session for mobile user REQUEST
4. SIP session established (connection details)
5. Connection details of mobile user
B. Runtime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

Figure 28 - EMS Sip Broker WS-Interface Sequence Diagram

3.3.5.3.2. *Alternative behaviour*

The Sip Broker WS-Interface is prepared to handle certain errors coming from the SIP infrastructure modules. The return value in the operations invoked by the Enactment Engine accounts for this type of errors. As it can be seen in the functionality section, the return value of these two operations (*startSession* and *transferSession*) is an *int* whose value indicates if the process has been carried out with success/failures. The meaning of the specific values are the same for both methods (0 SUCCESS, 1 DECLINED, 2 FAILED, 3 TIMEOUT, 4UAERROR and 5 NOTFOND). For instance, DECLINED means that then doctor does not want to make the call, FAILED one of the callers is not registered, TIMEOUT occurs when some of the callers is waiting too much to accept the call and NOTFOUND occurs when a grid transfer is being requested but no grid call is in progress).

The component will be updated during the maintenance phase in order to handle unexpected or erratic behaviours.

3.3.5.4. Involved technologies

Both services have been developed using the WSRF and WS-Notification implementation issued by Globus Toolkit 4.0. The services are deployed in the Kava WS-Core container also included in the Globus Toolkit 4.0.1 version.

3.4. Grid Service Discovery Service

3.4.1.1. Brief Overview

The service discovery server is divided into two parts – the service repository and the service discovery proxy. The service repository is principally replaceable, whereas the proxy stays the same. This way the proxy hides the actual service registry implementation from the search clients. In the testbed following configuration is used:

- Service Repository: ADONIS Business Process Management Toolkit with Akogrimo specific extensions running on a Windows machine
- GrSDS Proxy: C# Web-service running on a Windows machine

Figure 29 gives an overview.

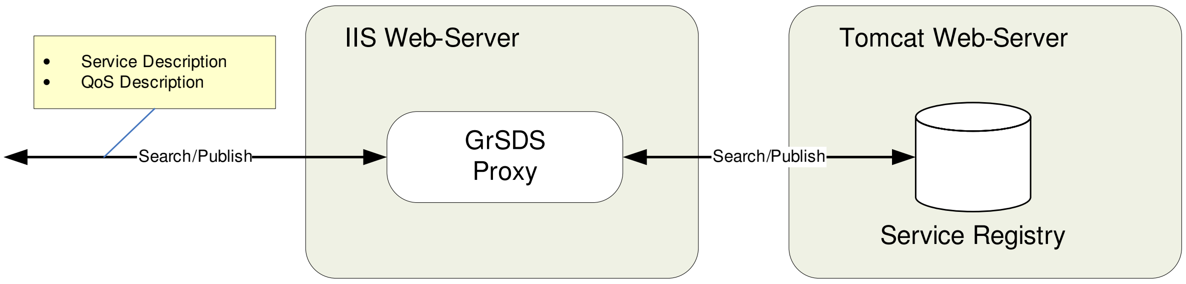


Figure 29 - GrSDS Design Overview

3.4.2. Functionality

The functionality of the GrSDS is logically divided into publishing of service descriptions and searching for services. Publishing involves publish, update and delete operations. The search functionality is accessed via the method SearchForServices().

Table 25 – GrSDS service methods

<u>GrSDS</u>	
Service Search	
Description	The OpVOBroker or a GUI client may use the GrSDS to find the desired services.

GrSDS	
E-GRSDS-SearchForServices	
public XmlElement SearchForServices(XmlElement serviceRequirements)	
Description	This method is used to provide a list of services that matches the service requirements passed as input
Service Publishing	
Description	Service providers can publish descriptions of the services they offer. The service publishing functionality of the GrSDS allows to store, update and delete service descriptions in the service registry.
E-GRSDS-PublishService	
public string PublishService(XmlElement serviceDescription)	
Description	PublishService() stores the service description in the service registry and creates a new unique id for that service description. The newly created service id is returned as a string value.
E-GRSDS-UpdateService	
public void UpdateService(string serviceId, XmlElement serviceDescription)	
Description	This method allows to update the service description of an existing service. The service id is the unique id that was returned from the PublishService() call.
E-GRSDS-DeleteService	
public void DeleteService(string serviceId)	
Description	Given the unique service id, this method deletes the service description from the service registry.

3.4.2.1. *Interactions with the other components*

3.4.2.1.1. *Main Behaviour*

The main functionality of the GrSDS is to allow publishing of and searching for services. The message elements of a search request and a response are show in the following two tables.

Table 26 - GrSDS Search Request

Descriptive Name	Search Request Message Element
Service Category	category
Service Type	serviceType
Geographical Location	location

Table 27 - GrSDS Search Response

Descriptive Name	Search Response Message Element
Service Name	serviceName
Service Description	serviceDescription
Service Id	serviceId
Service Provider Name	serviceProviderName
Service Provider Id	serviceProviderId
SP Internal Service Id	serviceProviderServiceId
Workflow Id	workflowId
SLA Template Id	slaTemplateId
Context Id	contextId
Negotiator URL	negotiatorURL
Negotiator Resource Id	negotiatorResourceId
BaseVO	baseVO

Searches are performed by the OpVO Broker in the OpVO creation phase or a customer.

The information that is provided by the Service Provider when a service is published consists of the content of both tables.

3.4.2.1.2. Alternative Behaviour

The source of exceptional behaviour can be one or more of the following:

- The network connection between the client and the GrSDS proxy, or the network connection between the GrSDS proxy and the Service Registry
- The IIS Web-server or the Tomcat Web-server
- The .NET or Java runtime environment
- The implemented functionality of either the GrSDS proxy or the Service Registry

Network connection problems usually result in a time-out on the client side and could be handled by a second attempt or an increased time-out value. Increasing the time-out value for requests from the GrSDS proxy to the Service Registry can be done by editing the Web.config file of the GrSDS proxy Web-service.

Table 28 - IIS Web.config File

```
<configuration>
  <system.web>
    ...
    <httpRuntime executionTimeout="123" ... />
    ...
  </system.web>
</configuration>
```

The `httpRuntime executionTimeout` is the maximum duration in seconds that a request to the Service Registry is allowed to take before it is considered timed-out.

Problems of the Web-server or the .NET or Java runtime environment are considered out of the scope of this document. So the main sources of exceptional behaviour are the implementation of the Service Registry and the GrSDS proxy.

The GrSDS design uses the adapter design pattern to abstract from the concrete implementation technology that is used for the Service Registry component. In the Akogrimo demonstration scenarios the ADONIS Business Management Toolkit is used as Service Registry. Because ADONIS is commercial software, the detailed internal error handling is out of the scope of this document. In principle any database with adequate search capabilities could be used as service registry. The purpose of the GrSDS proxy is to internally interface with the given implementation technology. So the following error handling flow chart deals only with the GrSDS proxy.

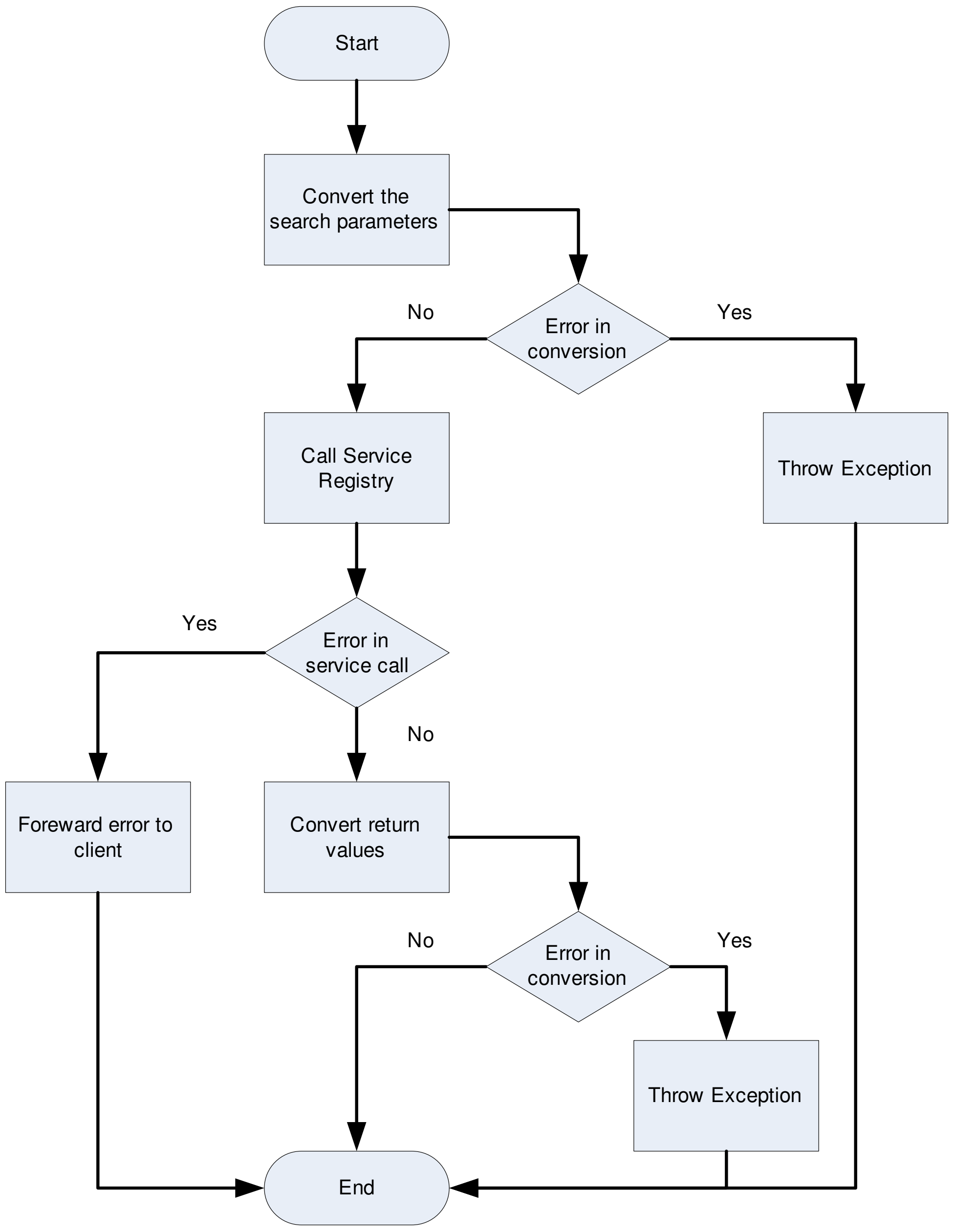


Figure 30 - GrSDS Proxy Error Handling

The error handling flow chart of the service publishing functionality is identical in that the parameters are converted into the required format and passed on to the Service Registry.

3.4.2.2. Involved technologies

The following two tables show the technologies used for the GrSDS Proxy service and the Service Repository.

Table 29 - Involved Technologies of the GrSDS Proxy

GrSDS Proxy	
Operating System	Windows 2003 Server
Web-Server	IIS 6.0

GrSDS Proxy	
Other Components	.NET Runtime v1.1

The GrSDS Proxy that interfaces with the actual Service Registry was designed to be a rather light weighted standard Web-service making it possible to use different underlying search engines.

Table 30 - Involved Technologies of the Service Repository

Service Repository (ADONIS Business Process Management Toolkit)	
Operating System	Windows 2003 Server
Web-Server	Apache Tomcat 5.5
Other Components	Java 1.5.0_08

The ADONIS Business Process Management Toolkit is a commercial software package developed by BOC Asset Management.

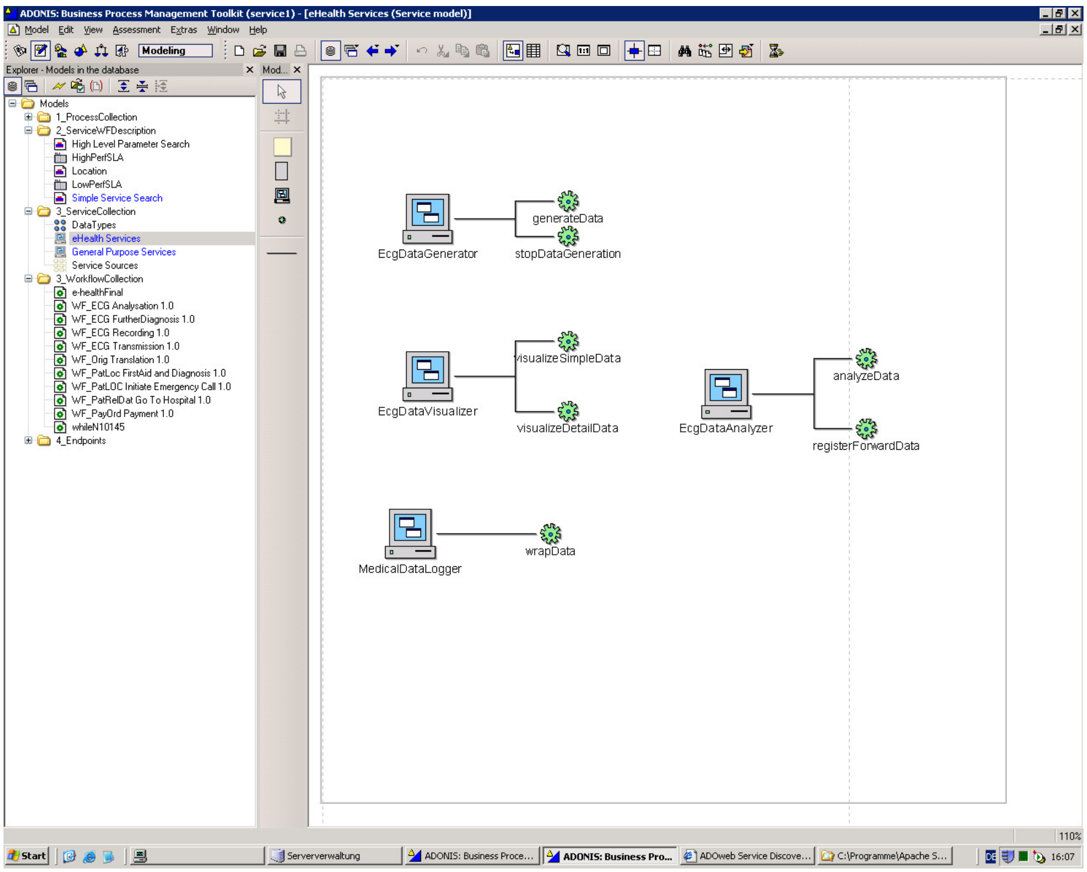


Figure 31 - ADONIS Business Management Toolkit

Service modelling is only a small part of the functionality of ADONIS. The richness of the modelling capabilities ranges from simple WSDL descriptions to semantic technologies like topic maps, which are used to model specific geographic locations like the hospital of the eHealth scenario with its different floors and rooms.

3.5. Security infrastructure

This section gives an overview of the overall security infrastructure developed in the frame of the Grid Application Support Service work package and the most of the available information have been extracted from the internal deliverable [8].

It is worth mentioning that, due to their relevance, in Akogrimo the multi domain aspects had a huge impact on the security infrastructure design choices, so before describing the developed security infrastructure the Akogrimo multi domain model is introduced again here.

Furthermore in order to put the reader into the picture, a complete overview of the GASS security infrastructure is provided but it will also explained any change introduced with respect the overall design in order to simplify the implementation.

3.5.1. Multi domain and security

A Virtual Organization is a dynamic collection of heterogeneous and distributed resources, which cooperate to satisfy common goals. Usually, the VO resources (users, services and physical resources) are owned and controlled by various organizations, these can often be viewed as separated by physical computing domains. In this model the Home Domain is responsible for the local administration of these resources and controls access to them according to their internal policies management mechanisms. In Akogrimo, we have identified five types of Home Domain:

- Customer domain; is an organization that buys services available in the Grid environment and makes these services available to its end users
- Network domain; is the domain of an organization providing network capabilities to access VO resources
- Service Provider domain; is the owner organization of resources involved in the VO, on the service level access to the VO's is managed by this domain.
- BVO domain; is the Akogrimo management services hosting domain and maintains the list of resources and members participating to VO activities; resources belong to Service Provider domain and members belong to the Customer domain
- OpVO domain; it is a logical domain created at run time in order to group all resources which are need to collaborate and to deliver a service to Customer domain. This domain lives in the BVO domain and includes resources from BVO and Service Provider domains.

The Figure 32 shows the relations between the BVO and the other domains, in particular it points out the presence of management services in the BVO domain which support this domain activities.

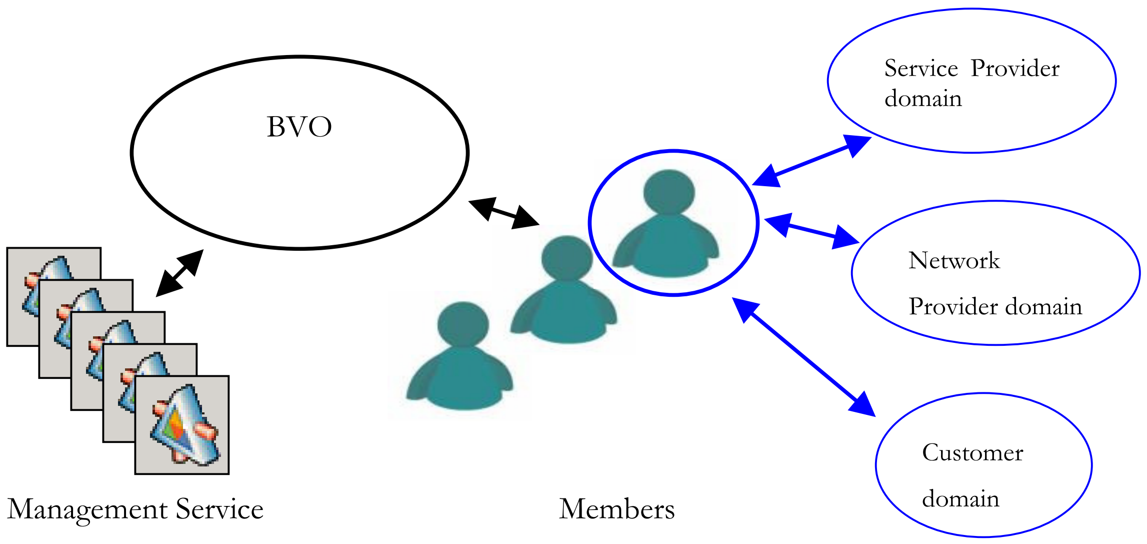


Figure 32 BVO definition

In [1] a *domain* is defined as "*administrative aspect of an organization (e.g. an Enterprise or a Company, a University Department or a Hospital) that comprises a set of individuals and resources. In a domain, the resources (hardware and software) are owned and managed by a common administrative authority*". We further detail this definition introducing the concept of *Administrative Domain*, which is a domain that includes at least an authentication and authorization authority and a list of members (see Figure 33).

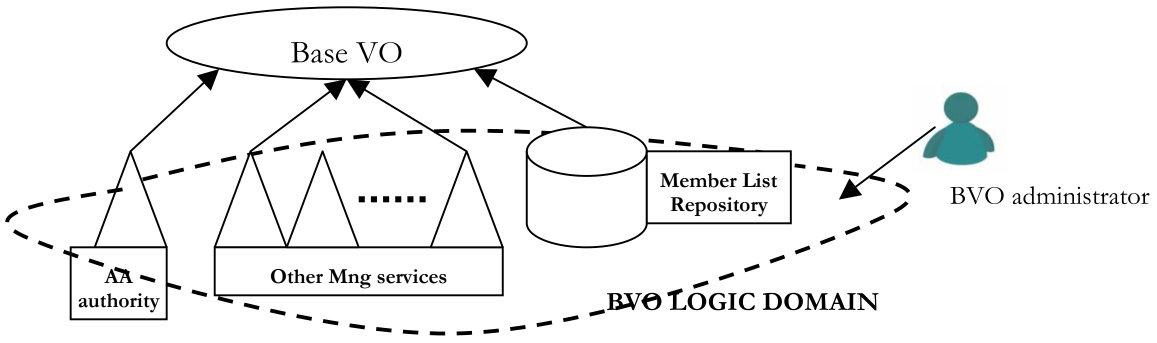


Figure 33 BVO described as an Administrative Domain.

The authentication and authorization service is a local authority that has to supervise the member's activity inside the domain; a repository exposed as service is used to manage domain list of members.

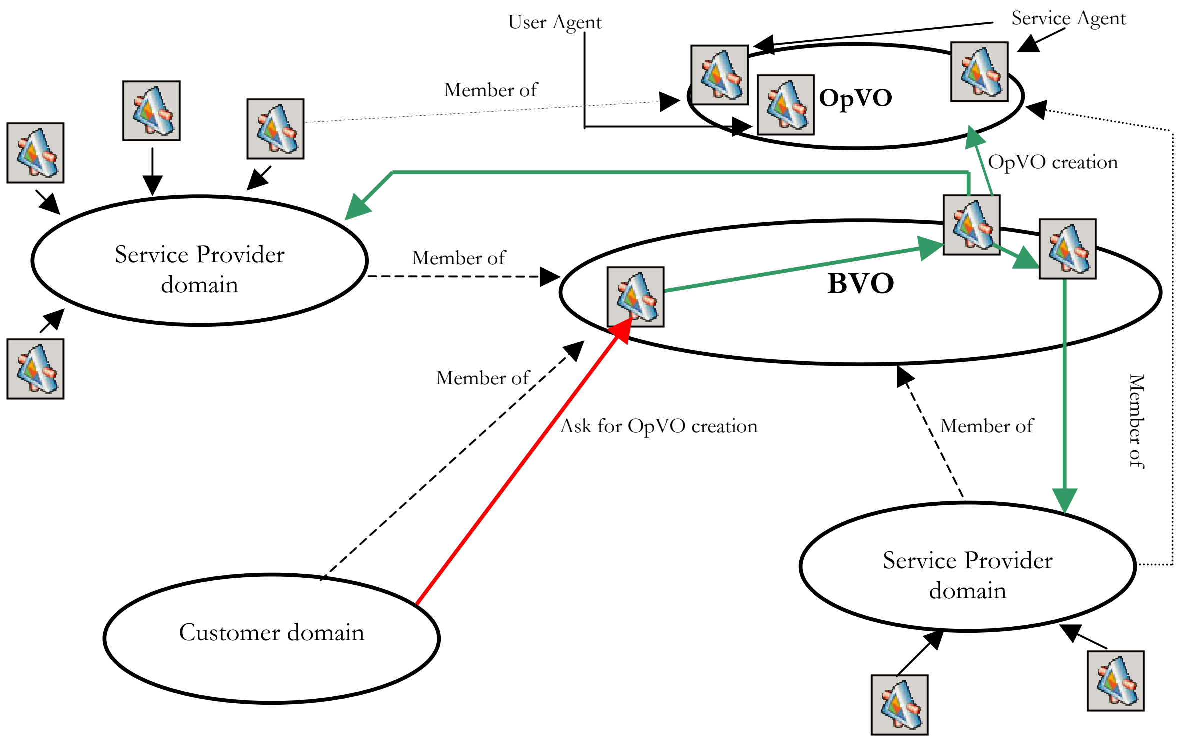
These components have an important role in a typical Akogrimo scenario, because it is based on the collaboration and cooperation between members of these domains. The main issue is allowing safe resources sharing and utilization, which requires:

- verify the identity of a requestor,
- check if the requestor is a member of the domain
- verify policies in order to authorize the request

To describe how interactions span the identified domains, we describe a generic service provisioning scenario. We suppose that a BVO is created and all Akogrimo management services have been configured and initialized. We assume an agreement has been established between a Service Provider and the BVO Manager for the sharing of some resources/services which are hosted in the domain of the Service Provider. Furthermore, we assume a Customer organization is a member of the BVO and it wishes to make available a service, provided by the BVO, to its end users.

As a member of the BVO, the Customer asks for the execution of an Akogrimo application which in turn leads to the creation of an OpVO which provides the requested services to fulfil the Akogrimo application requirements. The creation of the OpVO is authorized by BVO management services and requires collaboration between services in the BVO and in the Service Provider domains to negotiate the service instance that can fulfil Customer requirements. In particular, User and Service Agent instances are created; they assume an important role from the security view point because they are the component that should allow the cross and multi domain communication, in fact, they guarantee the secure communication from the external domain to the OpVO (external user of the OpVO) and from the OpVO towards the external domains (Service Provider domains. The logical OpVO domain is created and all selected service instances become members of this domain.

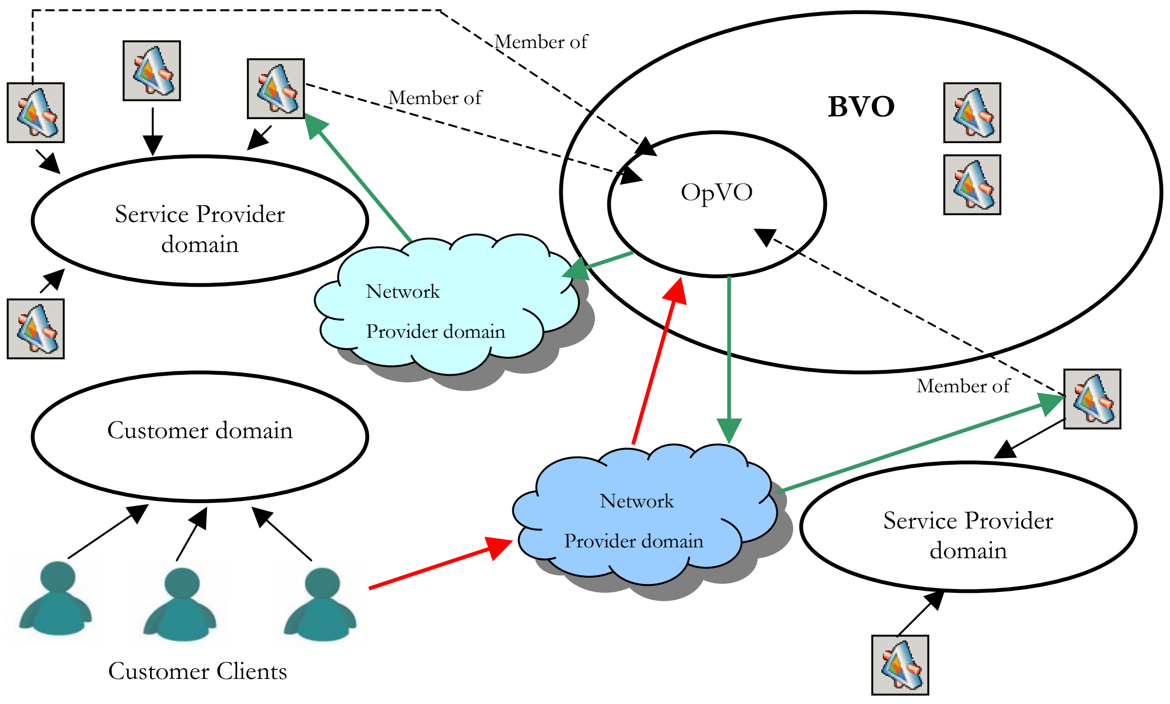
Figure 34 summarizes the process described above:



The customer asks for the OpVO creation as shown by red arrow. This request is elaborated by BVO services, which as to negotiate with the Service Providers the QoS for service instances (green arrows). Negotiated service instances become member of the OpVO and Service and User Agent instances are created

Figure 34 OpVO domain creation.

Then the customer can allow to its members to access OpVO functionalities according to rights defined in the Customer domain. An invocation starts from this user in the Customer domain and it passes through one (or more) Network Provider Domains to obtain access to the network. Then the request arrives into the OpVO domain, and from this domain several invocations have to start in order to use services in Service Provider domains involved in the OpVO execution. Of course also these last invocations must pass through several Network Provider domains that cannot be known in advance because the invoked services could be on mobile devices. These steps are described in Figure 35.



An end user, from the Customer domain, accesses to OpVO functionalities as shown by red arrows. The OpVO invokes services belong to Service Provider domains, as shown by green arrows, to orchestrate service capabilities and deliver the final service to the end user. The access to services in the OpVO and in Service Provider domains is realized via Network Provider domains.

Figure 35 The end user accesses the OpVO

The flow of inter domains interactions typical of the Akogrimo environment and described in this section have been taken into account in the design of the WP4.4 security infrastructure.

From the implementation viewpoint the following assumptions have been taken:

- BVO and OpVO will belong to the same secure perimeter. Even if several OpVO will be created inside a BVO, from security viewpoint all the services will belong to the same secure perimeter (the BVO perimeter). Although separate OpVO tokens will exist.
- Customer plays the user role as well
- Network provider is taken into account in the communication between costumer and BVO domain (red line in Figure 35).

3.5.2. Security infrastructure basics

The security infrastructure of Akogrimo is based on token distribution architecture around an Akogrimo Certificate Authority (CA). The Akogrimo CA presents methods to create Akogrimo tokens and check the validity of existing tokens. This service is called in order to authenticate communication within the Akogrimo VO and between the VO and external service provider domains.

Each communication is realized via a SOAP message; to allow the identification and authentication of the requestor, the SOAP message has to include a valid token and has to be signed by a trusted authority. When a SOAP message is received, a set of steps allow authenticating the sender as VO member.

Of course, as basilar assumption the approach does rely on a secure token distribution mechanism. The initial token distribution is based on a Trusted Third Party trusted by the VO members (including customers and service providers). It is also assumed that VO and SP core systems are not compromised before or during the initial token distribution.

Moreover, policy is enforced close to the point of decision. In respect to the Akogrimo architecture at the VO level policy management and enforcement stems from the Base VO. It is therefore the Base VO manager that has both the authority and means to make and enforce decisions relating to policy.

The Base VO is the key authority for VO tokens that it creates, with tokens also being created at service provider level. It is envisaged that the Policy component at Base VO level will have to consist of both an engine and tool to design policy; we use Permis as our Policy engine.

3.5.3. Functionalities

The main functionalities of the Akogrimo Security infrastructure involve authentication and authorisation of calls to the VO infrastructure. As discussed in the previous section with respect to the VO managers the UA provides a token that is authenticated. The requests made to the VO are authorised using the VO and PERMIS identity management authorization that is integrated into the BaseVO service.

According with the multi domain model and the final assumptions introduced in section 3.5.1, the security infrastructure has implemented four main use cases involving respectively:

- Authentication and authorization in communication between external requestors (customer domain or Service Provider domain) and VO domain
- Authentication and authorization in communication between services inside the VO domain

The following tables describe those use cases:

Table 31 - VO authentication use case

Use Case	**VO Authentication**
Description	This use case describes how a requestor (end user) is authenticated when asks for VO access. We use the generic term VO referring to BVO and OpVO because we have the same flow of events. Flow of events: • The user sends a request to its UA instance and specifies its SAMLID token and the username used in the VO. • The request is forward to VO Manager • The VO Manager sends the SAMIL token and username to A4C in the Home Network Provider domain • The A4C in the Home Network Provider domain returns to VO Manager the result of its authentication process

Use Case	VO Authentication
Actor	User, Home Network Provider, User Agent, VO Manager (BaseVO/OpVOManager)
Preconditions	The user has a SAMLID token issued by its Home Network Provider The user knows the endpoint of its UA instance The UA instance is a member of the VO and has a VO token
Post conditions	The user has been authenticated and then the authorization phase can be started

Table 32 - Intra VO Authentication use case

Use Case	Intra VO Authentication
Description	This use case describes how authenticate the service to service communication in the VO. Flow of events: • The service sends a request to another service • The service includes in the request its VO token • The VO token is forwarded by the receiving service to the VO Manager • The VO Manager authenticates the service requestor checking the validity of its VO token
Actor	VO services, VO Manager (BaseVO/OpVOManager)
Preconditions	The VO service has a key pair (private, public) The VO service has to have a VO token issued by the VO Manager
Post conditions	The VO service is authenticated as VO member

Table 33 - VO authorization use case

Use Case	VO Authorization
Description	This use case describes the requestor (end user) authorization process when he asks for VO functionalities. We use the generic term VO referring to BVO and OpVO because we have a same flow of events. Flow of events: • The VO Manager checks if the username is valid for the VO (username is stored in the VO Participant Registry) • The VO Manager retrieves the role from user profile using the username value • The VO Manager invokes Policy Manager to have the set of authorization policies to be applied for this role • The VO Manger matches the requested actions with user privileged in the VO
Actor	User, User Agent, VO Manager (BaseVO/OpVOManager), Policy Manager
Preconditions	The user has been authenticated successfully according to the above VO authentication use case The user has a username valid for the VO The user has a profile in the VO The user profile contains the role of the user in the VO The role specifies the rights/privileges of the user in the VO
Post conditions	The end user has been authorized to invoke the UA instance, which is able to act on behalf of the user inside the VO.

Table 34 - Intra VO Authorization use case

Use Case	**Intra VO Authorization**
Description	This use case describes the authorization process with regard to the communication between services inside the same VO. These services are likely to be BaseVO services or instances of BaseVO services, such as the OpVO Manager and Workflow Manager instances. It is not envisaged that service to service provider domain communication will take place via a Akogrimo VO. We use the generic term VO referring to BVO and OpVO because we have a same flow of events. Flow of events: • The service (requestor) sends a SOAP request to another service (destination) inside the same VO including the VO token • The request is intercepted by the Policy Enforcement Point (PEP)[13] • The PEP invokes the Policy Decision Point[14] (PDP) • The PDP invokes Policy Manager to retrieve policies about the use of service destination • The PDP accesses to requestor role in the VO member repository • The PDP matches the requested actions with service rights in the VO and requestor rights
Actor	VO member Services, PEP, PDP, Policy Manager, VO member repository
Preconditions	Each VO member service has a private and public key pair Each VO member service has a VO token issued by VO Manager The VO token contains service endpoint as subject and its public key The SOAP invocation has to contain the VO token and has to be signed by the requestor The SOAP invocation has been authenticated successfully Each VO member service has a role well defined in the VO
Post conditions	The request by VO member service is authorized or blocked

[13] From now on the term PEP refers to a SOAP filter in the SOAP engine of the invoked service able to extract from the SOAP message the header related to the security information sent by the requestor.

[14] From now on the term PDP refers to the VO manager (BVO or OpVO) that is able to take an authentication/authorization decision related to the actions required by a requestor with some credentials.

Apart from the above use cases, another scenario has to be considered related to the authentication and authorization for invocations from VO to Service Provider domain.

It is a typical multi-domain scenario because it addresses the issues related to the authentication of service requestor in the Service Provider domain. The services provided by a Service Provider are accessible via a Service Agent instance, which acts like a proxy for accessing to the target services. The SA is a member of a VO. So, this type of authentication involves the VO and Service Provider domains. The Service Agent, as VO member, has VO token; this token is sent to Service Provider which parses it to authenticate the requestor as OpVO member and to grant or block the request.

The process is similar to the one described in the intra domain authentication and same considerations can be done also for the authorization process. of course in this case the PEP and PDP components will be in the SP domain then the Service Provider can choose which implementation to use.

3.5.4. Interactions between the components

This section provides the list of components involved in the authentication and authorization (AA) process. Furthermore describes also the interactions between them providing the sequence diagrams related to the AA for:

- requests from an external user to the VO (see also use cases in Table 31 and Table 33)
- communications between services inside the VO (see also use cases in Table 32 and Table 34)

3.5.4.1. Involved components

3.5.4.1.1. Authentication

The components involved in the authentication process are: VOManager, A4C at Home Network Provider and the Authentication Decision Point.

The Authentication Decision Point is a special component of the SOAP pipeline and its task is to verify the validity of security tokens included in the incoming SOAP messages for the service.

The VOManager (see also section 3.1.1 and 3.1.2) and the Authentication Decision Point are components that are involved in each authentication scenario described above, while the A4C at Home Network Provider is the core component for the authentication of an end user when he or she accesses the VO functionalities via his or her User Agent instance (end user authentication scenario).

3.5.4.1.2. Authorization

The authorization process involves the VOManager, the ParticipantRegistry, the Policy Manager, and the Policy Enforcement Point (PEP).

The PEP is a filter in the SOAP Engine (actually it is the same of the Authentication Decision Point) and it starts the process in the pipeline which supervises incoming requests for a service in order to check if the requestor is authorized to ask for the specified action.

The role of the Policy Decision Point is taken over by the VO manager.

3.5.4.2. Sequence diagrams

Two sequence diagrams are provided here.

Figure 36 shows the steps for the authentication and authorization of an end user when they use their User Agent instance to invoke VO functionalities.

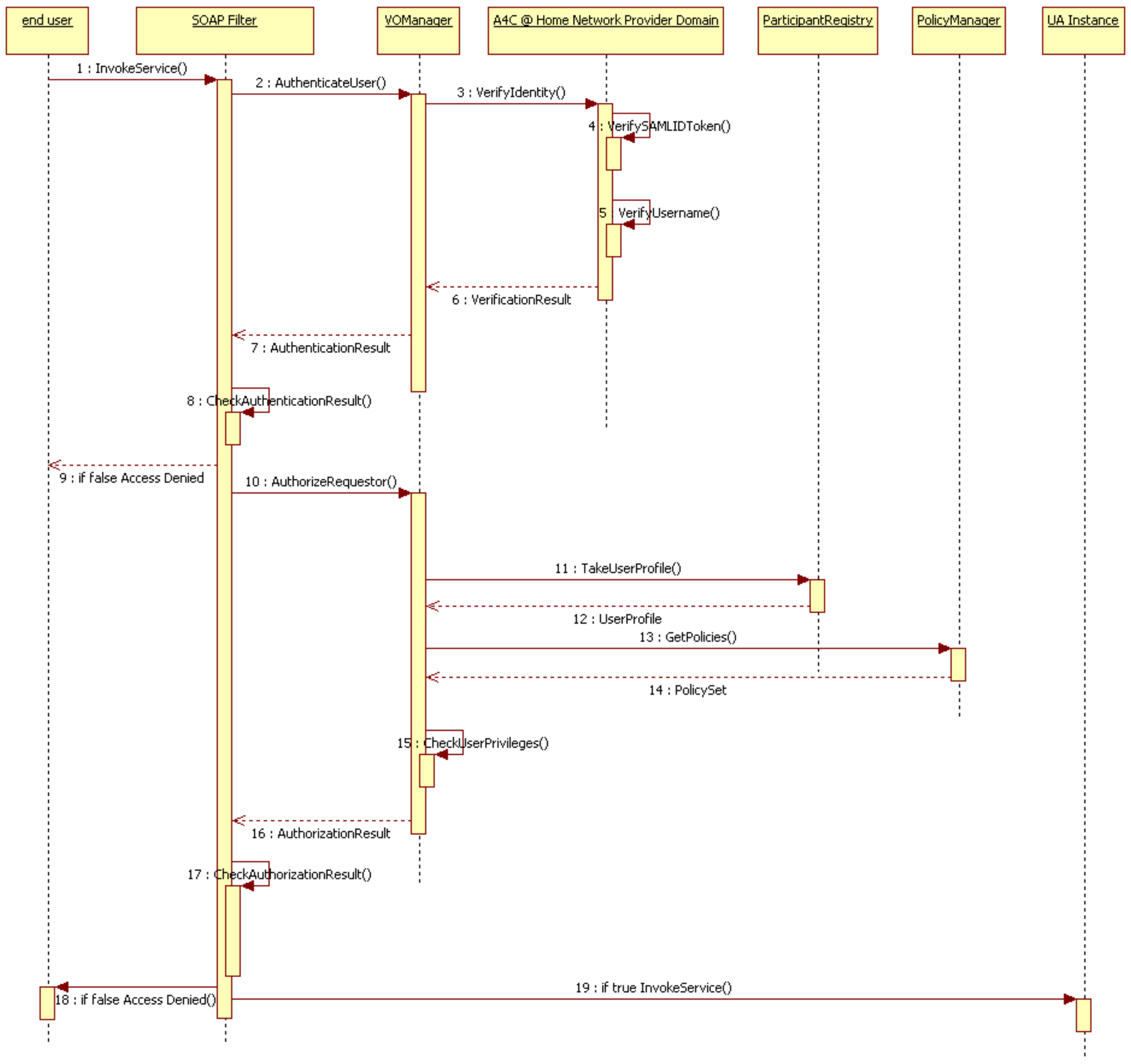


Figure 36 - End User AA

1. The SOAP filter catches the SOAP message asking for the invokeService method on the UA instance
2. The SOAP message is parsed to extract the security information (SAMLID token and user identity) and the VO manager is invoked to authenticate the requestor
3. The VO Manager checks the signature in the SAMLID token and if it is from a trusted A4C, invokes the A4C (in the network provider domain) to ask for authentication
4. 5. 6. The A4C performs checks on the SAMLID token and associated identities and returns the validation results
7. VO manager elaborates and passes back the validation results to the SOAP filter
8. 9. 10. SOAP Filter checks the results and decides to block the request or to continue in the filters chain invoking the VO Manager for request authorization

11. The VO manager in order to take an authorization decision retrieves the profile associated to the requestor identity.
12. The VO manager extracts the role from the received profile
13. 14. It gets the authorization policies from the Policy Manager
15. 16The VO Manager takes a decision on the basis of the information it has retrieved and passes back the authorization results

17. 18. 19. The SOAP Filter on the basis of the received results allows the request to be processed by the UA instance (19) or blocks it (18)

Figure 37 shows how the communication between two services inside the VO (VOservice1 and VOService2) has to occur from the authentication and authorization viewpoint. As members of the VO, these services have the VO token which contains service public key and is signed by VO Manager.

VOService1 tries to invoke the Method() on VOService2.

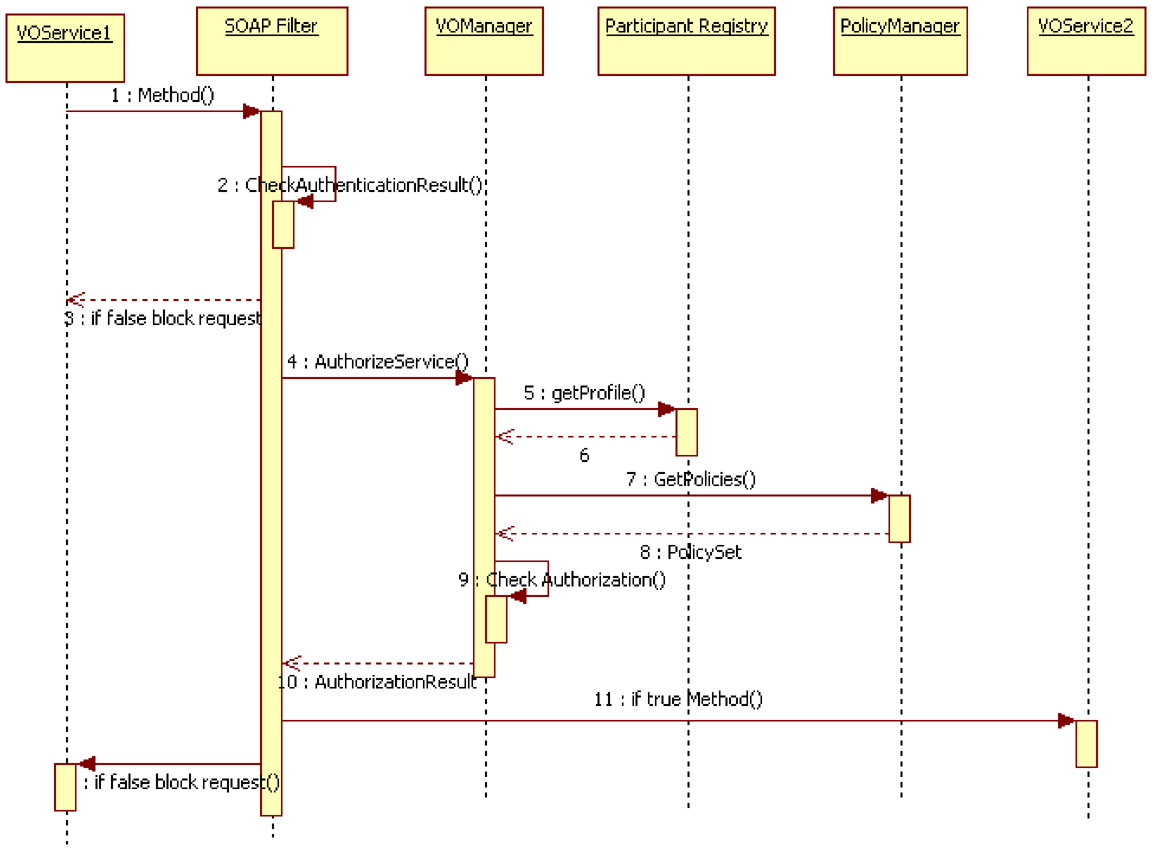


Figure 37 – AA between services inside the VO

1. The SOAP message for requesting Method() on VOService2 is blocked in the SOAP Filter of the VOService2 of the SOAP engine
2. The SOAP Filter extracts the security information included in the SOAP message and asks for authenticating the requestor.
3. 4. If the requestor is authenticated the SOAP Filter asks the VOManager for authorizing this request on the basis of requestor identity.

5. 6. The VO Manager retrieves the profile of the invoker and the associate role

7. 8. 9. The VO Manager gets the authorization policies associated to the role and takes a decision.

10. 11. 12. The SOAP Filter on the basis of the received results allows the request to be processed by the VOService2 (11) or blocks it (12)

3.5.5. Involved technologies

The implementation of security infrastructure has dealt with logic underlying:

- The implementation of SOAP Filter. Furthermore, filter have been implemented using tools available on hosting platforms (i.e. Axis and WSE) that provides features to manage the incoming SOAP message according with the WS-Security specification.
- The authentication/authorization process in the VO manager. It is based on the use of X.509 certificate and PERMIS for authorization

4. Conclusions

The goal of the WP4.4 prototype release described in this report was to implement the features that were not addressed during the first implementation cycle.

In the first prototype these missing features were grouped in three main areas:

- Dynamic OpVO creation
- Improvement of WF instances management
- Security infrastructure

These brief conclusions have the aim of summarizing at which extent the current prototype has met the objective of providing the above features.

In particular, referring to the above bulleted list:

- The implementation of the dynamic OpVO creation mechanism has covered all the pending functionalities that were related to this feature. The complete process has been summarized in section 2.2.2 and it involved the implementation of new services (e.g. OpVOBroker, SLA Negotiator,…) as well as the update of existing ones (e.g. OpVO Manager, BVO Manager,…). It is worth mentioning that a preliminary integration of a first release of all involved services has been successfully tested running a demo of preliminary implementation of the eHealth scenario.
- The mechanism managing the WF instance deployment and WF dynamic adaptation was improved, as well. The current implementation does not require offline setup anymore, but all the required configurations are performed dynamically at run time during the OpVO creation phase. Also this mechanism has been successfully tested with an intermediate implementation release, which has provided important feedbacks to finalize the implementation in the right way.
- The implementation of the security infrastructure is a pending task. Though a final design is available, a complete implementation is not available yet. This the main implementation task to be covered during the next months apart from the maintenance activities that can be required as result of the integration and evaluation phase.

In accordance with the report we can conclude that the most of the goals were met but some important implementation tasks are still pending and they will be covered during the next five months when the final implementation is going to be released.

During these months the available services of GASS layer will be integrated with the other available modules from other WPs and the integrated prototype will be validated through a quantitative testing that will provide relevant feedbacks that could result in further updates if the required changes will be evaluated necessary to execute successfully the validation phase.

References

[1] Akogrimo official deliverable: D.3.1.3 "The Mobile Grid Reference Architecture"

[2] Akogrimo official deliverable: D4.4.1 "Architecture of the Application Support Services Layer"

[3] Akogrimo official deliverable: D4.2.3 "Final Integrated Services Design and Implementation Report"

[4] Akogrimo official deliverable: D5.1.2 "Integrated Prototype"

[5] Akogrimo official deliverable: D4.3.2 Prototype Implementation of the Grid Infrastructure layer

[6] http://www.permis.org/

[7] Akogrimo official deliverable D4.1.3 "Final Service Network Provisioning concepts"

[8] Akogrimo internal deliverable ID4.4.3 "Updated architecture design"

A.1. Participant Profile

When registering to a VO, a participant profile is taken over and stored in the ParticipantRegistry. If the VO needs more or less attributes, the participant can change the attributes.

An example of such profile, used in the current implementation is:

Profile @ VO

Username
User Data (option)
Role
Attributes
Participant Public Key

Figure 38 - BVO participant profile example

This profile is described using a XML document and the related schema is defined below:

```
<xs:complexType name="ParticipantProfileType">
<xs:sequence>
<xs:element name="ParticipantId" type="xs:string" />
<xs:element name="ProfileId" type="xs:string" />
<xs:element name="Role" type="xs:string" />
<xs:element name="ParticipantPublicKey" type="xs:string" />
<!–user data -->
<xs:element name="ApplicationId" type="xs:string" />
<xs:element name="PDA" type="xs:boolean" />
<xs:element name="MobilePhone" type="xs:boolean" />
<xs:element name="Notebook" type="xs:boolean" />
<xs:any minOccurs="0" maxOccurs="unbounded" />
<!--end user data -->
   </xs:sequence>
   </xs:complexType>
<xs:element name="ParticipantProfile" type="tns:ParticipantProfileType" />
```

At the moment, the Participant Registry has been tested using XML document based on the above schema.

A.2. VO token

The architecture is secured by a token and policy enforcement infrastructure. The root of authority for VO tokens is the BaseVO Manager which provides BaseVO tokens for all members of the BaseVO. All communication to or from a BaseVO member is accompanied by a BaseVO issued token. The Operative VO mirrors this. However the OpVO Manager tokens are issued from the Base VO. Thus allowing the BVO control over the tokens in the static and dynamic VO infrastructure.

In addition to the BVO Token, when services and user agents join the BaseVO they are also given the Base VO's public Key. When they join the OpVO they also receive an OpVO

Membership token and the OpVO public key. The public keys are used to decrypt messages and tokens received from the Base VO or OpVO. This is a simple method of validation, rather than referring back to the BaseVO/OpVO

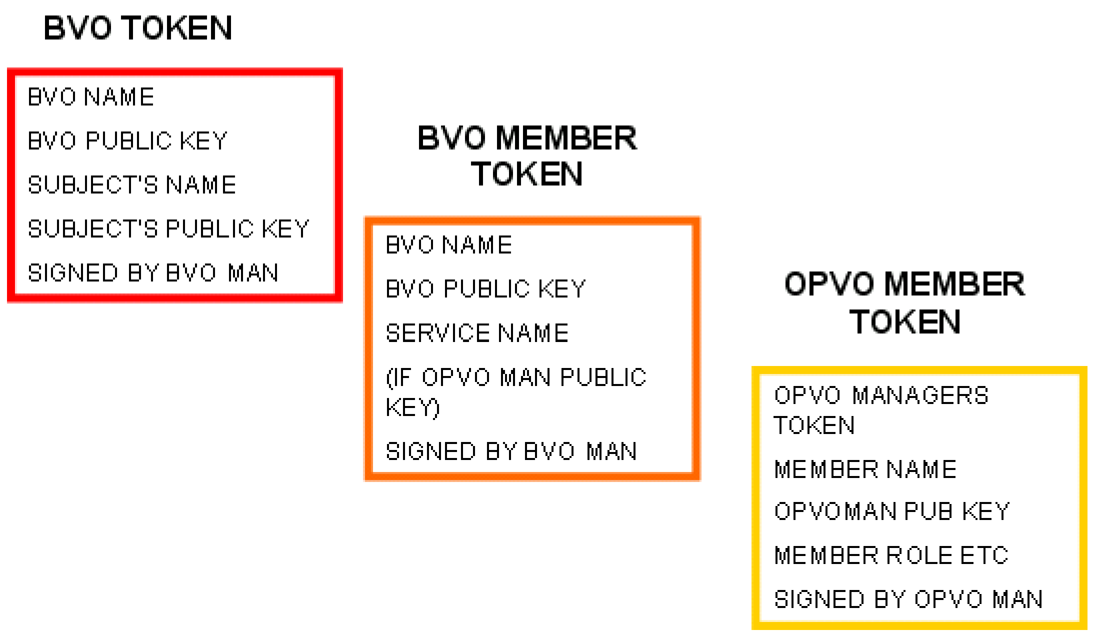


Figure 39 - VO Tokens

As the Figure 39 illustrates, the respective VO managers are the trusted validation (CA equivalent) parties for the individual tokens when sent cross domain, and they share their respective public keys between their members.

All VO tokens issued from the managers are encrypted using the receiver's public key and decrypted by the receiving services public key and the tokens are signed using the sender's private keys. Within a domain, all token flow is encrypted. In order to achieve this, public keys of the individual components sending the messages out are included in the tokens accompanying the message.