D4.4.2

Prototype Implementation of the Grid Application Support Service layer

WP4.4 Grid Infrastructure Services Layer

Dissemination Level: Public

Lead Editor: Giuseppe Laria, CRMPA

30/01/2006

Status: Final

SIXTH FRAMEWORK PROGRAMME

PRIORITY IST-2002-2.3.1.18

Grid for complex problem solving

Proposal/Contract no.: 004293

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

a. to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works;
b. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works.
c. For the avoidance of doubt, where the work is a musical composition:
    i. **Performance Royalties Under Blanket Licenses**. Licensor waives the exclusive right to collect, whether individually or via a performance rights society (e.g. ASCAP, BMI, SESAC), royalties for the public performance or public digital performance (e.g. webcast) of the Work.
    ii. **Mechanical Rights and Statutory Royalties**. Licensor waives the exclusive right to collect, whether individually or via a music rights society or designated agent (e.g. Harry Fox Agency), royalties for any phonorecord You create from the Work ("cover version") and distribute, subject to the compulsory license created by 17 USC Section 115 of the US Copyright Act (or the equivalent in other jurisdictions).
d. **Webcasting Rights and Statutory Royalties**. For the avoidance of doubt, where the Work is a sound recording, Licensor waives the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions).

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

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

a. You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any credit as required by clause 4(b), as requested.
b. If you distribute, publicly display, publicly perform, or publicly digitally perform the Work or Collective Works, You must keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or (ii) if the Original Author and/or Licensor designate another party or parties (e.g. a sponsor institute, publishing entity, journal) for attribution in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; the title of the Work if supplied; and to the extent reasonably practicable, the Uniform Resource Identifier, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work. Such credit may be implemented in any reasonable manner; provided, however, that in the case of a Collective Work, at a minimum such credit will appear where any other comparable authorship credit appears and in a manner at least as prominent as such other comparable authorship credit.

5. Representations, Warranties and Disclaimer

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

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

7. Termination

a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Collective Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.
b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

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

Activity **4**	Detailed Architecture, Design & Implementation
WP4.4	Grid Application Support Services Layer Architecture, Design & Implementation
Dependencies	This is the accompanying report prepared as the D4.4.2 Prototype Implementation of the Grid Application Support Services Layer deliverable.

Contributors:

Contributors (in alphabetical Order):

Annalisa Terracina (DATAMAT): section 3.3
Brian Ritchie (CCLRC): section 3.3
Ferry Syafei (UHOH): section 3.4
Francesco D'Andria (Atos Origin): section 3.2
Giuseppe Laria (CRMPA): section 1, section 2, contributions integration
Ian Johnson (CCLRC): section 3.1
Josep Martrat (Atos Origin): section 3.2
Nadia Romano (CRMPA): section 3.2, section 3.1
Raul Bori (Atos Origin): section 3.2
Robert Piotter (USTUTT): section 3.2
Tom Kirkham (CCLRC): section 3.1

Reviewers:[1]

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

Version	Date	Authors	Sections Affected
0.1	19/10/2005	Giuseppe Laria	Template
0.2	26/10/2005	Giuseppe Laria	Template revision focusing on modules description
0.3	9/11/2005	All	All. First version of developed service description
0.4	29/11/2005	Ferry Syafei	Application Specific Services description
0.5	2/12/2005	Raul Bori	SLA High level services description: update
0.6	23/12/2005	Ian Johnson, Tom Kirkham	VO management services description: update
0.7	12/1/2006	Giuseppe Laria	Update of the overall document structure and addition of sections 1 and 2
0.8	19/1/2006	Brian Ritchie	Section 3.3: update
1.0	31/1/2006	Giuseppe Laria	Final integration

Table of Contents

Executive Summary 11
1. Introduction – Overview 12
1.1. Addressed concepts 12
1.2. The Overall Picture 12
1.2.1. Offline Setup 13
1.2.2. Operational behaviour 13
2. The developed services 16
2.1. VO management subsystem 16
2.1.1. BVO Manager 16
2.1.2. OpVO Manager 17
2.1.3. User Agent (UA) 18
2.1.4. Service Agents 19
2.1.5. Participant Registry 21
2.2. SLA High Level 29
2.2.1. SLA-Access 29
2.2.2. SLA-Translator 30
2.2.3. Contract/Template-Repository 32
2.3. BP enactment 34
2.3.1. WorkFlow Registry 34
2.3.2. WorkFlow engine 35
2.3.3. WorkFlow Manager 38
2.3.4. Monitoring Daemon 39
2.4. Application specific services 40
2.4.1. Objectives 40
3. Conclusions 44

List of Figures

Figure 1 – The patient uses the OpVO 14

List of Tables

Table 1 - BVO manager service methods....16
Table 2 – OpVO manager service methods....17
Table 3 – User Agent methods....18
Table 4 - Service Agent methods....20
Table 5 - Participant Info service methods....22
Table 6 – SLA Access service methods....29
Table 7 – SLA Translator service methods....31
Table 8 - SLA repository service methods....33
Table 9 - WF Registry service methods....34
Table 10 - WF engine methods....36
Table 11 - WF manager service methods....38
Table 12 - Monitoring Daemon service methods....40
Table 13 – Application Specific services methods....41

Abbreviations

Akogrimo	Access To Knowledge through the Grid in a Mobile World
BVO	Base Virtual Organization
BVOm	BVO Manager
ECG	Electrocardiogram
GASS	Grid Application Support Services
MD	Monitoring Daemon
OpVO	Operative Virtual Organization
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
SLA_A	SLA Access
SLA_R	SLA Repository
SLA_T	SLA Translator
UA	User Agent
WF	Workflow
WF_E	Workflow Engine
WF_M	Workflow Manager
WF_R	Workflow Registry

Executive Summary

In this document, the functionality and implementation status of the first prototype of Grid Application Support Service layer is presented. The different components will be described and their interfaces and implementation technologies is documented.

In order to reach the current prototype, the WP4.4 participants have implemented and refined multiple modules that have been designed in the Grid Application Support Service layer architecture deliverable (D4.4.1). Of course at the current stage just a subset of all functionalities have been implemented.

This document assumes a general understanding of Akogrimo concepts and a high level knowledge of GASS layer architecture (see Deliverable 3.1.2 "Initial Overall Architecture" and for more details Deliverable 4.4.1 "Architecture of GASS layer")

It is organized as follows:

- Section 1 provides an overview of the scope and goal of this first prototype. Furthermore a summary of the available modules is provided and a sketch of their behaviour
- Section 2: provides a set of tables that list for each module the provided methods and their description. Details about the involved technologies are provided as well.
- In the conclusion it is underlined the status of the implementation with respect to the long term vision that will be achieved in the final prototype.

1. Introduction – Overview

The GASS layer represents the "front-end" of Akogrimo platform, in fact, it provides high level services that allow the application to have an access point to leverage on Akogrimo platform capabilities. This layer provides the following major subsystems (see D.3.1.2 "Detailed Overall Architecture Design" and D4.4.1 "Architecture for the Application Support Services Layer" for detailed design):

- VO management: includes services to manage BVO and OpVO.
- SLA High Level: it provides services to manage negotiation and contract definition.
- BP enactment: it provides services to manage the instantiation and execution of a workflow template.
- Application Support Services layer: provides services and tools to support specific testbed execution in the frame of Akogrimo environment.

Of course, in the first prototype we haven't implemented all components and also the implemented components don't provide all functionalities designed in the architecture documents. The details about the available components and related functionalities are available in the following section 2.

1.1. Addressed concepts

The first prototype of GASS layer has the aim of providing proof of concept with respect to some requirements of the demo scenario that in its turn is a preliminary prototype of the eHealth test bed. In particular, in this scenario, we assume an OpVO already exists and we are going to proof the feasibility of:

- BVO authentication (integration between network and grid layer authentication)
- Using an OpVO through the User Agent and Service Agent components.
- Adaptation of Business Process execution according with context changes.
- Interactions with other layers of the architecture:
  - WP4.3 that provides the infrastructure to execute the business services involved in the business process.
  - WP4.2 that provides components supporting authentication (A4C server) and business process adaptation (context changes notification through the context manager).

1.2. The Overall Picture

For each subsystem listed above a set of services has been implemented, that is:

- **VO management**:
  - BVO Manager (BVOm)
  - OpVO Manager (OpVOm)
  - User Agent (UA)
  - Service Agent (SA)
  - Participant Registry (PR)
- **SLA high level**:

- SLA Access (SLA_A)
- SLA Translator (SLA_T)
- SLA Repository (SLA_R)

- **BP enactment**:
  - Workflow Manager (WM)
  - Workflow Registry (WR)
  - Workflow Engine (WF_E)
  - Monitoring Daemon (MD)

In this first prototype just a set of the functionalities foreseen in the final version have been implemented then we have to assume that many setup actions have to be performed offline. Actually all the process of the OpVO creation is simulated in order to bring all the components in an operational status.

Then in order to use this prototype we have to identify two different phase: offline setup and operational status.

1.2.1. Offline Setup

In this step, we configure all the services involved in the operational phase. Of course this configuration steps will be performed by the Akogrimo platform itself in the final version of the prototype but, at the moment, due to missing functionalities we perform the configuration through some GUI clients.

The offline setup involves:

- Creation and configuration of the User Agent
- Configuration of the Service Agents
- Store information in Participant Registry and WF Registry
- Deployment of business process script
- Creation and Configuration of monitoring daemon.

1.2.2. Operational behaviour

The following Figure 1 shows how the prototype works when everything has been set. We are assuming the patient is already logged in the network and every actions to allow him/her to contact the WP4.4 has been performed (it is part of the prototypes developed in other WP4.x).

We are going to explain the meaning of each interaction labelled with a sequence number.

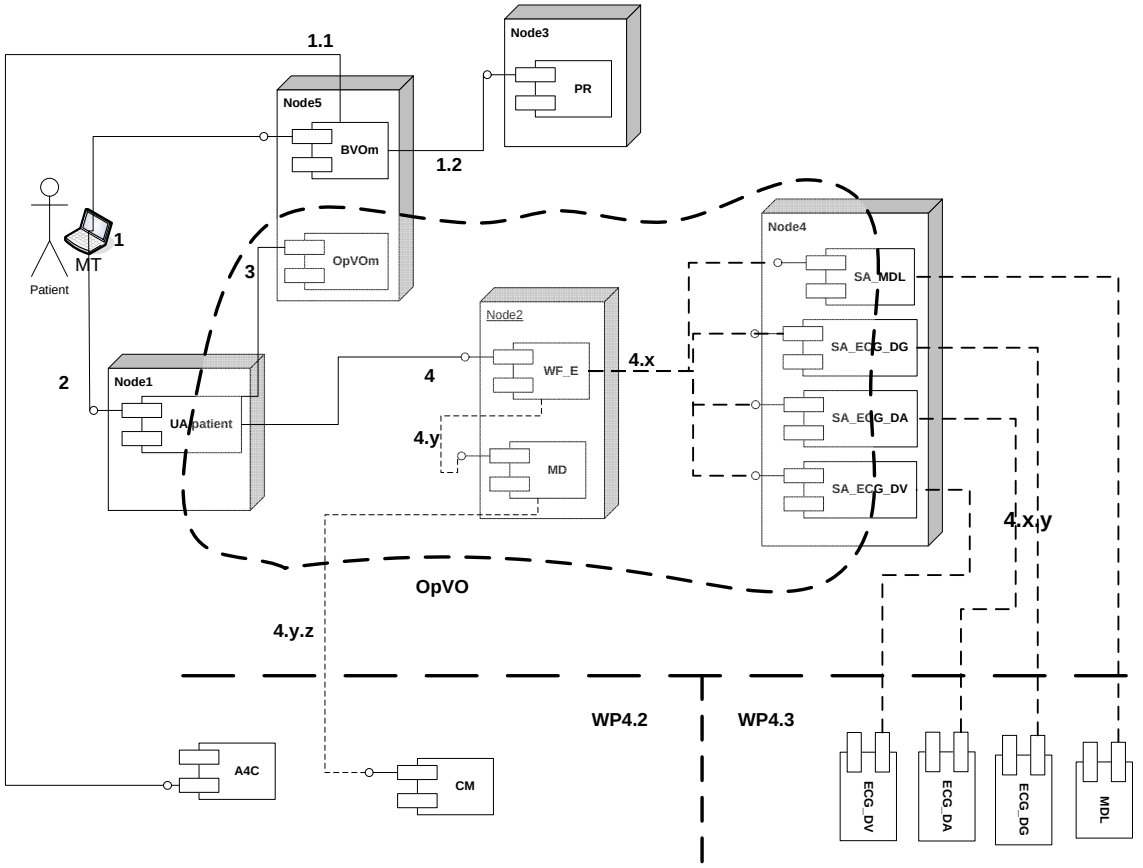

Figure 1 – The patient uses the OpVO

1. The patient has logged on the network and he/she has a reference to the BVO manager. The BVO manager is invoked in order to retrieve the list of services (User Agent) running in the BVO on behalf of the patient.

   1.1. BVOm asks the A4C server for authenticating the incoming request

   1.2. The authentication is successful, then the BVOm calls the PR to retrieve the list of pair UAs and related OpVO token to be returned to the patient.

2. The patient chooses an application (UA) between the available for him/her (received from the BVOm). He/she will invoke the UA using the associate OpVOtoken and specifying the method and parameters to be passed.

3. The UA invokes the OPVOm to validate the OpVO token received by the patient in the method invocation.

4. 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 patient. In the specific case, it will invoke the service exposed by the BP engine to start the workflow execution.

   4.x. 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.

      4.x.y. The SAs will forward the request to the business service instances running in different administrative environment and managed by services developed in the frame of WP4.3

4.y. During the workflow execution the MD will receive notification from the CM about context change. The BP engine will be informed about these changes and the workflow execution will be adapted to the new context.

This is the behaviour of the GASS layer first prototype and it interacts with prototype of WP4.2 layer with respect to notification of context changes and with WP4.3 that provides and manages the execution of the external services, that in Figure 1 are the ones involved in our first scenario: ECG_DV, ECG_DA, ECG_DG and MDL. Of course, depending on the workflow script deployed in the BP engine the steps 4.x and 4.y will be different.

In this first prototype release, we have designed just one workflow template that is the one of the first simplified scenario.

The following section provides details about each single service and functionalities implemented so far.

2. The developed services

2.1. VO management subsystem

2.1.1. BVO Manager

2.1.1.1. Objectives

The Base VO Manager is the key part of policy and authorization enforcement at the top of the Base VO. The Manager interacts as a "go between" in between the participant's registry, A4C server and external requestors. All service requests to the VO pass through the manager and checked against the A4C server and participant's registry by the manager. Based on the response from the A4C server, the Base VO manager makes a decision to either refuse or the request entry to the Base VO. If entry is granted the Base VO Manager creates an OpVO Manager and starts the process to create a new OpVO, at the end the Participants registry is informed of the new participant's credentials

2.1.1.2. Functionality

The Table 1 provides a list of the available methods on each service. The methods have been grouped in macro categories.

Each method will be identified with a summary notation:

E-X-Y-Z

E = External interface

X = Service/group name

Y = Subservice of X

Z = Method name

Table 1 - 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 EPR of a UserAgent and the associated OpVO token to invoke this UserAgent.

2.1.1.3. Involved technologies

The main technologies involved in the implementation of this service are:

- Windows XP and RHEL 3
- Apache Tomcat 5
- GT4
- Ws-core-3.95 (GTK libs)

2.1.2. OpVO Manager

2.1.2.1. Objectives

The Operative VO Manager (OpVO Manager) takes the role of the VO manager during the execution of an Akogrimo application inside an OpVO. Once a requestor has been authenticated by the Base VO it receives a token that is validated by the OpVO manager during the lifetime of the Akogrimo application it is valid for. The token is passed from the User Agent to the OpVO everytime as a parameter and at this stage will be a simple string, the UA itself will pass its token everytime as a SOAP message and at this stage will be a simple XML file.

2.1.2.2. Functionality

The Table 2 provides a list of the available methods on each service. The methods have been grouped in macro categories.

Each method will be identified with a summary notation:

E-X-Y-Z

E = External interface

X = Service/group name

Y = Subservice of X (if applicable)

Z = Method name

Table 2 – 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.
OpVO-JoinReq	
Boolian [] checkToken (String token, String participantId)	
Description	The OpVO manager will receive a request from a User Agent to perform a task in the Base VO, this request is accompanied by the agents token.

2.1.2.3. Involved technologies

The main technologies involved in the implementation of this service are:

- Linux Ubuntu
- Apache Tomcat 5
- GT4
- Ws-core-3.95 (GTK libs)

2.1.3. User Agent (UA)

2.1.3.1. Objectives

This service belongs to VO Management topic and takes care to represent 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 by the outside. The OpVO user always we'll invoke the same method on the UA that will generate in an automate way the invocation for the right service inside the OpVO.

The service is implemented as WSRF service, we will have a dedicated UA for each external user.

2.1.3.2. Functionality

The Table 3 provides a list of the available methods on each service. The methods have been grouped in macro categories.

Each method will be identified with a summary notation:

E-X-Y-Z

E = External interface

X = Service/group name

Y = Subservice of X (if applicable)

Z = Method name

Table 3 – User Agent methods

UserAgent	
Application methods	
Description	This group includes methods it is the method called from outside to ask for a service inside the OpVO
E-UA-InvokeApplication	
public arrayOfXmlElement InvokeApplication(string methodName, arrayOfXmlElement inputMethodParameters)	

UserAgent	
Description	This method allows to start the execution of a specific method on a service inside the OpVO. The OpVOToken is passed in the SOAP message and the User Agent extracts it to manage security accesses. It returns the object with application results or null.
E-UA-GetMethodList	
public string GetMethodList( )	
Description	This method allows to external invoker to retrieve the list of methods available to be invoked inside the OpVO
Configuration methods	
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 to configure the UA with respect to the service methods it has granted access in the OpVO.

2.1.3.3. Involved technologies

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 pipeline in order to extract the VOToken.

Summarizing the main involved technologies are:

- MS Windows 2003 Server
- WSRF.NET
- WS-Security

2.1.4. Service Agents

2.1.4.1. Objectives

This service belongs to VO Management topic and takes care to represent a service inside the OpVO. Moreover it is in charge to manage secure accesses to the service associated to the agent. It implements a general mechanism to check access rights, but for each service typology foresees a specialization in terms of data managed and proxies to access the service.

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

2.1.4.2. Functionality

The Table 4 provides a list of the available methods on each service. The methods have been grouped in macro categories.

Each method will be identified with a summary notation:

E-X-Y-Z

E = External interface

X = Service/group name

Y = Subservice of X (if applicable)

Z = Method name

Table 4 - Service Agent methods

ServiceAgent	
Application methods	
Description	This group includes all the methods that are a replication of the methods exposed on the outside service represented by the SA inside the OpVO. They will be different depending on the outside service, then we will not list them here because they can be found between the Application Support Services
The service agent will be customized on the base of services to invoke. This means that this service will provide the same methods of component services (involved into the workflow) to invoke. Maybe if we will need some additional info in order to manage service agent actions we will add it in the final interface. As soon as possible we will have the component services interfaces we will provide external interfaces.	
Configuration	
Description	This group includes methods to configure the service agent about different aspects: authorization rights, services to be invoked,… At this stage the security set up is not available yet.
E-SA-setEPR	
setEPR (EndpointReferenceType EPR)	
Description	Allows to set the EPR of the outside service to be invoked

2.1.4.3. Involved technologies

The service is implemented using Web Service technology and in particular it will work with SOAP over HTTP.

Summarizing the main involved technologies are:

- SOAP
- WSRF.NET

2.1.5. Participant Registry

2.1.5.1. Objectives

The Participant Registry service is actually constituted by three services that provide different kind 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 to create a WS-Resource associated to each participant and we will store them information related to the specific participant. In fact, it is able to manage information associated to each different participant about:
  - Tokens
  - Roles
  - Agents
  - 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). We will have 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. Retrieving the endpoint reference it is possible to access to the participant related information. This service is a simple Web Service.

2.1.5.2. Functionality

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

Each method will be identified with a summary notation:

E-X-Y-Z

E = External interface

X = Service/group name

Y = Subservice of X

Z = Method name

Table 5 - 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 agent to access a service in an existing participant WS-Resource. The result is the token identifier if operation has success, otherwise "failure"
E-PR-ParticipantInfo-UpdateTokenAgent	
public bool UpdateTokenAgent(TokenAgentType oldEntry, TokenAgentType newEntry)	
Description	Updates a pair token and agent 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 agent data, in particular, the EPR of the agent. Return true if the update has success, false otherwise
E-PR-ParticipantInfo-RemoveTokenAgent	
public bool RemoveTokenAgent(TokenAgentType entry)	
Description	Removes a token and its agent associated to a participant Returns true if peration has success, false otherwise
E-PR-ParticipantInfo-RetrieveTokenList	
public TokenType[] RetrieveTokenList()	

Participant Info	
Description	Retrieves the tokens' list of the participant Returns token type list if has success, null object otherwise</returns>
E-PR-ParticipantInfo-RetrieveAgentList	
public EndpointReferenceType[] RetrieveAgentList()	
Description	Retrieve the agents' 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 agents associated to the participant It returns the Full list if has success, null object otherwise
Profile Management	
Description	The methods in this group manage the profiles associated to each 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 has success, "failure" otherwise
E-PR-ParticipantInfo-UpdateProfile	
public bool UpdateProfile(ParticipantProfileType oldProfile, ParticipantProfileType newProfile)	
Description	Updates an existing profile of the participant It returns true if the operation has success, false otherwise
E-PR-ParticipantInfo-RemoveProfile	
public bool RemoveProfile(ParticipantProfileType profile)	
Description	Removes a participant profile It returns True if the operation has success, false otherwise

Participant Info	
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 has success, 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 allow 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 participantsm, …
E-PR-ParticipantInfo-ConfigureParticipantProperty	
public bool ConfigureParticipantProperty(ParticipantProperty prop)	
Description	Configures property of the participant
E-PR-ParticipantInfo-UpdateParticipantProperty	
public bool UpdateParticipantProperty(ParticipantProperty oldProp, ParticipantProperty newProp)	
Description	Configures properties of a participant It returns true if the update has success, false otherwise</returns>
E-PR-ParticipantInfo-RetrieveParticipantProperty	
public ParticipantProperty[] RetrieveParticipantProperty()	
Description	Retrieves participant properties Returns participant properties array if operation has success, null otherwise

Participant Info	
E-PR-ParticipantInfo-RemoveParticipantProperty	
public bool RemoveParticipantProperty(ParticipantProperty prop)	
Description	Removes property of participant Returns True if deletion has success, false otherwise
E-PR-ParticipantInfo-SetParticipantIdentifier	
public bool SetParticipantIdentifier(string id)	
Description	Set identifier of participant Returns True if setting has success, false otherwise
E-PR-ParticipantInfo-GetParticipantIdentifier	
string GetParticipantIdentifier()	
Description	Returns the participant identifier

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.
E-PR-VOInfo- AddParticipantReference	
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 has success, 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 has success, false otherwise

VOInfo	
E-PR-VOInfo- RetrieveParticipantEpr	
public EndpointReferenceType RetrieveParticipantEpr(string partId)	
Description	It allows to retrieve the participant endpoint reference Returns the participant endpoint reference if has success, null otherwise
E-PR-VOInfo- RemoveParticipantReference	
public bool RemoveParticipantReference(ParticipantMapping partRef)	
Description	Remove participant reference information Returns true if the operation has success, 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 has success, false otherwise
E-PR-VOInfo-GetVOIdentifier	
public string GetVOIdentifier()	
Description	Get the OpVO/BVO identifier Returns the identifier if the operation has success, "failure" otherwise
E-PR-VOInfo-SetManagerIdentifier	
public bool SetManagerIdentifier(string id)	
Description	Set the (BVO or OpVO) Manager identifier Returns True if the operation has success, false otherwise
E-PR-VOInfo-GetManagerIdentifier	
public string GetManagerIdentifier()	

VOInfo	
Description	Get the (BVO or OpVO) Manager identifier Returns the identifier if the operation has success, "failure" otherwise
E-PR-VOInfo-SetManagerEpr	
public bool SetManagerEpr(string manEpr)	
Description	Set the (BVO or OpVO) Manager EndpointReferenceType Returns true if the operation has success, 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 has success, "failure" otherwise

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()	
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	

ParticipantNameService	
public bool RemoveParticipantMappingCollection()	
Description	Removes the collection used to maintain mapping between participant identifier and their endpoint reference Returns true if the operation has success, 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 has success, false otherwise
E-PR-ParticipantNameService-RemoveParticipantMappingEntry	
public bool RemoveParticipantMappingEntry(string participantId)	
Description	Removes a participant from the collection It returns true if the action has result, false otherwise
E-PR-ParticipantNameService-RetrieveParticipantEpr	
public EndpointReferenceType RetrieveParticipantEpr(string participantId)	
Description	Retrieves a document from a collection Returns the document if the operation has success, null otherwise

2.1.5.3. Involved technologies

The service is implemented as service on top of WSRF.NET.

The involved technologies are:

- Windows Server 2003 Enterprise Edition
- WSE 2.0 SP3
- IIS 6.0
- .NET Framework 1.1
- WSRF.NET
- Xindice 1.1b4
- Tomcat 5.0.28

2.2. SLA High Level

2.2.1. SLA-Access

2.2.1.1. Objectives

This service provides all necessary functionalities that other components could require from the document template and contract. Unlike SLA-Translator, SLA-Access does not access directly to 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 what 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 (retrieveInfoMetadata, setInfo 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 them carried out with a support service as the SLA-Translator.

2.2.1.2. Functionality

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

Each method will be identified with a summary notation:

E-X-Y-Z

E = External interface

X = Service/group name

Y = Subservice of X (if applicable)

Z = Method name

Table 6 – 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-retrieveInfoMetadata	
public objMetadata retrieveInfoMetadata(string SLADocId)	

SLA-Access	
Description	Retrieves general information related to a service An object with requested metadata if the operation has success, null object otherwise
E-SLA-Access-setInfo	
public bool setInfo(string SLADocId, objInfo info)	
Description	This method allows to fill specific sections of SLA contract It returns true if the operation has success, false 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

2.2.1.3. Involved technologies

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

Involved technologies:

- Windows Server 2003 Enterprise Edition
- WSE 2.0 sp3
- IIS 6.0
- .Net Framework 1.1
- WSRF.NET v2.1.1
- Xindice 1.1b4
- Tomcat 5.0.28

2.2.2. SLA-Translator

2.2.2.1. Objectives

This service is the only responsible for accessing physically to the documents (SLA-Template and SLA-Contract) and get (or set) information from them. For each document it will know the associated ID and it will contact with the repositories in order to retrieve the appropriate document.

The SLA-Translator has been implemented as a WS-Resource 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 makes possible to choose between different types of results just changing the input parameters of both operations.

2.2.2.2. Functionality

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

Each method will be identified with a summary notation:

E-X-Y-Z

E = External interface

X = Service/group name

Y = Subservice of X (if applicable)

Z = Method name

Table 7 – SLA Translator service methods

SLA- Translator	
SLA contracts and templates management	
Description	This group of methods allows to process the XML document that represents the SLA contract/template
E-SLATranslator- getData	
public string getData (string SLADocId, string rootTag, int occurrence)	
Description	This method will be used for retrieving specific sections from a template or contract. In getData method, the consumer has to specify the tag name involved in the operation by setting the simple tag name or a concrete root tag. Furthermore, it has also to specify how many occurrences it expects from the results that is choosing between the first occurrence found / all the occurrences / the first one and its siblings. 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 bool setData (string SLADocId, objUpdateTag objUpdate)	

SLA- Translator	
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 tag name involved in the operation by setting the simple tag name or a concrete root tag. Furthermore, it has also to fill a specific object type with the data it wants to update.

2.2.2.3. Involved technologies

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

Involved technologies:

- Windows Server 2003 Enterprise Edition
- WSE 2.0 sp3
- IIS 6.0
- .Net Framework 1.1
- WSRF.NET v2.1.1
- Xindice 1.1b4

2.2.3. Contract/Template-Repository

2.2.3.1. Objectives

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

The SLA Template Repository is currently implemented as a couple of regular Web Services deployed in c# (TemplateRepository and ContractRepository) that simply virtualizes a file system, but in future it will be a WS-Resource developed in c# (.NET) that will use Xindice in order to store/retrieve information interacting with this DB.

2.2.3.2. Functionality

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

Each method will be identified with a summary notation:

E-X-Y-Z

E = External interface

X = Service/group name

Y = Subservice of X (if applicable)

Z = Method name

Table 8 - 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)	
Description	Given the name of an SLA Template as templateID, the method GetTemplate() retrieves the document and returns is as 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 is as XmlDocument

2.2.3.3. Involved technologies

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:

- Windows Server 2003 Enterprise Edition
- WSE 2.0 SP3
- .Net Framework 1.1

2.3. BP enactment

2.3.1. WorkFlow Registry

2.3.1.1. Objectives

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

The Workflow Registry contains BPEL templates that correspond to the Business Processes. In addition the Workflow Registry contains what we call "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.

At this stage, the Workflow Registry will be based on MySQL. It will be exposed as a web service that offers methods to upload/retrieve templates. The web service will be implemented in Java.

For the first phase, no semantic will be included. This means that no OWL-S use is foreseen. The template search will be just based on simple keywords.

2.3.1.2. Functionality

The Table 9 provides a list of the available methods on each service. The methods have been grouped in macro categories.

Each method will be identified with a summary notation:

E-X-Y-Z

E = External interface

X = Service/group name

Y = Subservice of X (if applicable)

Z = Method name

Table 9 - WF Registry service methods

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

WF- Registry	
E-BPE-WFRegistry- Store	
workflowID **Store**(wfTemplateFile, annotations[])	
Description	It returns a workflow identifier assigned by database upon success. The parameter is the workflow implementation file and the associated semantic annotation array.
E-BPE-WFRegistry- Get	
WFTemplateFile **Get**(*workflowID*)	
Description	It returns the workflow implementation file upon success. The parameter is the workflow identifier eventually returned by the **searchService** request sent to the GrSDS.
E-BPE-WFRegistry-Update	
Update(workflowID, annotations[], values)	
Description	It returns success or failure. The parameters are the workflow identifier, the semantic annotations to be updated and their new values
E-BPE-WFRegistry-	
Remove(workflowID)	
Description	It returns success or failure. This function removes the specified workflow from the database. The parameter is the workflow identifier.

2.3.1.3. Involved technologies

Involved technologies:

- MySQL
- Basilar Web Service specifications (SOAP, WSDL)
- Java
- Linux Ubuntu

2.3.2. WorkFlow engine

2.3.2.1. Objectives

The Workflow Engine is the component of the Business Process Enactor in charge of enacting specific BPEL processes submitted by the Workflow Manager component. The Workflow

Engine should and will support execution of BPEL processes and calling of services with a WSDL description.

The Workflow 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 canceling a business process, those for inspecting and retrieving I/O and status information of a workflow.

2.3.2.2. Functionality

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

Each method will be identified with a summary notation:

E-X-Y-Z

E = External interface

X = Service/group name

Y = Subservice of X (if applicable)

Z = Method name

Table 10 - WF engine methods

WF- engine	
Workflow instances management	
Description	This group of methods allows to manage the workflow instances living inside the WF engine
E-BPE-WFEngine-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 *D441-GASSlayerArchitecture*
E-BPE- BPEngine-resumeProcess	
resumeProcess (processID)	
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 *D441-GASSlayerArchitecture*
E-BPE-BPEngine-terminateProcess	
terminateProcess **(processID)**	

WF- engine	
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 D441-GASSlayerArchitecture
E-BPE-BPEngine-process_list	
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-BPEngine-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 *D441-GASSlayerArchitecture*
E-BPE-BPEngine-getVariable	
process_variable getVariable**(processID, name)**	
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 D441-GASSlayerArchitecture
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, then we are not going to list here, because they are application specific. In the specific case of the eHealth scenario we will have just one method startFlow.

2.3.2.3. Involved technologies

The workflow engine will be interpreted by the ActiveBPEL engine. This is an opensource implementation of a parser and engine for enactment of BPEL processes. As of now, specification 1.1 of BPEL is fully supported. The ActiveBPEL engine has been installed on a linux machine running the Ubuntu Linux distribution. It is installed as a servlet for Tomcat with Apache as http server.

ActiveBPEL engine is delivered as a standalone solution in which an axis servlet is also included (no external installation is necessary) and for which both a WSDL interface and a programming Java API are provided (the above mentioned methods are directly taken from this engine's wsdl interface).

Involved technologies:

- Apache: any version should be OK as long as Tomcat works fine.
- Tomcat: 5.0.28 (be sure to define the $CATALINA_HOME env variable).
- Axis: 1.2.1 but not necessary. ActiveBpel deploys all necessary jar to Tomcat into $CATALINA_HOME/shared/lib (make sure to add the jars contained in this directory to your CLASSPATH)
- JDK: 1.4.2_09-b05 (don't use 1.5.0! I had many problems with it). Remember to define the $JAVA_HOME env variable and to add $JAVA_HOME/jre/lib/*.jar to your CLASSPATH.
- Ant: 1.6.5 (important, prior versions makes use of web services impossible

2.3.3. WorkFlow Manager

2.3.3.1. Objectives

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 Workflow Engine; part of this transformation includes service instantiation and registration for context changes (with the Context Manager) and SLA violations (with SLA Enforcement). It should also receive context/SLA notifications from the MD and (possibly) interrupt or redirect the current workflow. A Workflow Manager (WFM) is created (by an OpVO Manager, or by another Workflow Manager); thus each instance of the WFM 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.

Any kind of control will not be available.

2.3.3.2. Functionality

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

Each method will be identified with a summary notation:

E-X-Y-Z

E = External interface

X = Service/group name

Y = Subservice of X (if applicable)

Z = Method name

Table 11 - WF manager service methods

WF- Manager

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-WFM-Create	
Boolean CreateWFManager(WFTemplate)	
Description	Create a new Workflow Manager to handle a workflow template (supplied as parameter). (Note: creation of multiple instances implies that this feature should be provided by a "WFM Factory" service, though this will be virtual in the first phase.)
E- BPE-WFM-Start	
Boolian CreateWFManager(WFTemplate, EPR)	
Description	Instruction to a WF 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)

2.3.3.3. Involved technologies

Involved technologies:

- Java
- GT4 core
- WS-Notification

2.3.4. Monitoring Daemon

2.3.4.1. Objectives

This is a component of the Business Process Enactment service (the other components being the Workflow Manager (FWM), 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 WFM for decisions/ action.

2.3.4.2. Functionality

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

Each method will be identified with a summary notation:

E-X-Y-Z

E = External interface

X = Service/group name

Y = Subservice of X (if applicable)

Z = Method name

Table 12 - 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()	
Description	Create a new Monitoring Daemon. We expect this to be invoked by a WFM, so that each WFM has (to all intents and purposes) a dedicated MD (Note: creation of multiple instances implies that this feature should be provided by a "MD Factory" service, though this will be virtual in the first phase.)
E- BPE-MD-GetContext	
Public GetContext(userID)	
Description	To be used by a client to query the last known context for a user ID.

2.3.4.3. Involved technologies

Involved technologies:

- Java
- GT4 core
- WS-Notification

2.4. Application specific services

2.4.1. Objectives

The application specific services include a selected set of services from the eHealth domain that will be the building block to execute the preliminary eHealth scenario. These services are: ECG Data Generator, ECG Data Visualizer, ECG Data Analyzer and Medical Data Logger.

Of course their implementation is in a preliminary stage and the aim is to verify:

- If the available infrastructure is able to support their execution and
- the envisioned working of architectural middleware components in the context of an eHealth environment.

In the following sections we are going to provide a brief description of each service and the related methods.

2.4.1.1. Functionality

The Table 13 provides a list of the available methods on each service. The methods have been grouped in macro categories one for each service.

Each method will be identified with a summary notation:

E-X-Y-Z

E = External interface

X = Service/group name

Y = Subservice of X (if applicable)

Z = Method name

Table 13 – Application Specific services methods

Application Specific Services	
ECG_Data_Generator	
Description	This service functions as a data generator for a patient. It simulates the heart beat of the patient every second based on several parameters (ECG sampling frequency, heart rate mean, etc). The generated ECG data is periodically stored locally in a different file every five seconds. Right after having a new file, a notification is sent to the workflow to transfer the file to the medical data logger and finally to the ECG data analyzer to be analyzed. In the following the list of methods that it provides
generateData	**generateData(String patientName)** This method is used to generate the heart beat of the given patient based on a default parameter value. The generated heart beat is buffered locally. *patientName*: a string indicating the name of the patient
stopDataGeneration	**stopDataGeneration(String patientName)** This method stops the generation of the ECG data for the given patient. *patientName*: a string denoting the name of the patient.
MedicalDataLogger	

Application Specific Services	
Description	This service is used to wrap the generated ECG data in a standard format. The data is copied to the Medical Data Logger by the Data Manager from the ECG Data Generator. The wrapped data is stored temporarily before being sent to the Data Manager.
wrapData	**String resultAbsolutePath wrapData(String sourceAbsolutePath)** This method wraps the data from the ECG Data generator in a standard format. *sourceAbsolutePath*: The absolute path of the ECG data, which resides in the ECG Data Generator's location. *resultAbsolutePath*: The absolute path of the wrapped ECG data, which resides in the Medical Data Logger's location.
EcgDataAnalyzer	
Description	This service is used to analyze the heart beat of a patient and predict whether the patient will have a probability to get a heart attack. In case the patient will get a heart attack, the *EcgDataAnalyzer* sends a notification to the workflow engine to start the heart attack handling process. Furthermore it starts forwarding the current data to a cardiologist (when there is a heart attack) or a big display (when the context changes).
analyzeData	**analyzeData (String patientName)** This method is used to analyze the ECG data of the patient to find out whether the patient will suffer a heart attack. When *analyzeData* method is invoked, a thread is created to perform the analysis of the ECG data for the given patient. It is continuously running in the background and invokes the workflow to start the heart attack handling process in case a heart attack symptom is detected. *patientName*: a string which is the name of the patient.
registerForwardData	**String ack registerForwardData(String targetName)** This method forwards the current data to the cardiologist (when there is a heart attack) or a big display (when the context changes) by invoking the data forwarding process. *ack*: is a string informing whether the data transfer to the target destination (such as cardiologist or a big display) is success or fail.
EcgDataVisualizer	

Application Specific Services	
Description	This service visualizes the ECG data of a patient in a detail or a coarse manner. It can be used by the cardiologist, patient, or clerk to look into the actual status of the heart beat of the patient.
visualizeSimpleData	**visualizeSimpleData(String patientName)** This method visualizes the ECG data of the given patient in a coarse manner due to the limitation of the display. *patientName*: a string which indicates the name of the patient.
visualizeDetailData	**visualizeDetailData(String patientName)** This method visualizes the ECG data in detail. *patientName*: a string indicating the name of the patient.

2.4.1.2. Involved technologies

Involved technologies:

- Java
- GT4
- Ubuntu

3. Conclusions

The goal of this firs prototype has been to have preliminary feedbacks about some technological aspects that constitute an innovative approach introduced by Akogrimo but, at the same time, they can be an issue that could arise problems to achieve the final objectives.

The focus of this implementation has been on the following main points:

- ***Identification of possible integration and interoperability issues***: in Akogrimo we have planned to use different hosting platform (Microsoft and Linux) running Web Service and WS-Resource implemented on top of different frameworks (based on Java or C#). The integration results of the first prototype show as the solution of interoperability and interaction issues in such environment is not so simple as expected using standard specifications and protocols. We have derived an important lesson learnt from this preliminary prototype because we have identified and solved some common problems in order to achieve communication between GT4 and WSRF.NET based platform. We would underline as the main of these problems are related to minor differences in the standard formats used to exchange information (e.g. SOAP header contents, WSDL, XSD,…), then the main effort has been spent in the identification of the issues but not in the solution implementation. The gained experience constitutes a solid basis and this topic can be considered in an advanced status, otherwise the solution of this kind of issues is a pre-requirement for the following implementation phase.
- ***Authentication from Network to Grid layer***: one of the most relevant point of Akogrimo is that network is not there to be used by the grid service requestor without considering the involvement of this use. Then as first step we have focused on how to use a grid service integrating network and grid level authentication. The prototype has been useful as proof of concept with respect to the use of the A4C and SAML server to store and authenticate incoming request both to Network and Grid layer. In particular, with respect to the GASS layer the future developments will require the full adoption of WS specification in order to manage authentication token.
- ***Proof of concepts related to context changes notification***: the context status awareness is a very important objective in Akogrimo then it is clear that this prototype had to provide preliminary feedback in relation to the feasibility of context information transfer from the lower layer to the GASS layer. Of course the prototype provides limited capabilities (e.g. limited number of available context types, limited number of context changes management capabilities,…) but it proofs the feasibility of Akogrimo approach. Other work needs to be done in order to extent the current status at the aim of managing more complex context change situations.
- ***Proof of concepts related to business process adaptation depending on the context changes***: this is the most challenging objective and it is strictly related to the previous one. Also in this case, the prototype shows the integration between the notification of context changes and the consequent modification of workflow execution depending on the value of the context. In this first version we have tested the behaviour in a specific case (i.e. finding out the presence of a big display to provide a detailed data visualization) but we have to improve this capability in order to be able to manage:
  - Adaptation in case of faults
  - Lack of agreed QoS
  - Several kind of context changes

Furthermore at this stage the adaptation is managed inside the workflow script but in the future work it will be directly managed by the "intelligent" component of the BP enactment subsystem, that is the WorkFlow manager.