ICT 269978

Integrated Project of the 7th Framework Programme

COOPERATION, THEME 3
Information & Communication Technologies
ICT-2009.5.3, Virtual Physiological Human

Work Package: WP2
Data and Compute Cloud Platform
Deliverable: D2.3

First Prototype of the Cloud Platform

Version: 1v3

Date: 19-Mar-12

Document Information

IST Project Num	FP7 – ICT - 269978	**Acronym**	VPH-Share
Full title	Virtual Physiological Human: Sharing for Healthcare – A Research Environment		
Project URL	http://www.vphshare.org		
EU Project officer	Joël Bacquet		

Work package	**Number**	2	**Title**	Data and Compute Cloud Platform
Deliverable	**Number**	2.3	**Title**	First Prototype of the Cloud Platform

Date of delivery	**Contractual**	29-Feb-12	**Actual**	19-Mar-12
Status	Version 1v3		Final ☐	
Nature	Prototype ☒ Report ☐ Dissemination ☐ Other ☐			
Dissemination Level	Public (PU) ☐ Consortium (CO) ☒		Restricted to other Programme Participants (PP) ☐ Restricted to specified group (RE) ☐	

Authors (Partner)	CYFRONET, UCL, UvA, AOSAE			
Responsible Author	Piotr Nowakowski		**Email**	p.nowakowski@cyfronet.pl
	Partner	CYFRONET	**Phone**	+48600280105

Abstract (for dissemination)	This document details the features and technologies used in the implementation of the initial prototype of the VPH-Share cloud management platform. It lists the status of each component produced by WP2, ongoing work in each technical task and the specifics of integration with external Work Packages.
Keywords	cloud computing, data storage federation, high performance computing, hybrid clouds

The information in this document is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability. Its owner is not liable for damages resulting from the use of erroneous or incomplete confidential information.

Version Log			
Issue Date	**Version**	**Author**	**Change**
30.01.2012	0.1	Piotr Nowakowski	Initial draft
6.02.2012	0.2	Piotr Nowakowski, David Chang, Spiros Koulouzis, Rodrigo Diaz Rodriguez	Inclusion of selected descriptions of technical tasks
14.02.2012	0.9	Piotr Nowakowski, Jan Meizner, Tomasz Gubała, Tomasz Bartyński, Marek Kasztelnik	Preparing the deliverable for internal review: inclusion of additional descriptions of technical tasks; writing the summary and conclusions; proofreading and formatting
17.02.2012	0.95	Piotr Nowakowski, Marian Bubak	Additional updates and revisions
20.02.2012	1.0	Piotr Nowakowski, Marian Bubak	Internal review release
12.03.2012	1.1	Piotr Nowakowski, Susheel Varma, Dario Ruiz Lopez, Tomasz Gubała, Jan Meizner	Additional changes following internal review and the Consortium Meeting in Krakow (5-6 March 2012)
16.03.2012	1.2	Susheel Varma, Piotr Nowakowski	Final check
19.03.2012	1.3	PMO	Submission Version

Contents

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

This document presents the current state of development of Work Package 2 tools and services in the context of the VPH-Share project. The goal of this document is to summarize the existing features of the Cloud application development, enactment and data storage platform, explain how the prototype can be used to deploy existing WP5 workflow services and present possible use cases which are already supported by the prototype. In addition, the document outlines future development plans in Work Package 2.

This deliverable is meant as a follow-up to Deliverable 2.2, published at the end of Project Month 6. Where D2.2 focused on the design of individual application components, this document is meant as a report on how this design translates into practical solutions and deployable software. Any deviations from the initial design, whether due to evolving user requirements or discovery of additional technical possibilities, are listed and explained, as is their impact on the overall Project architecture.

The following topics are addressed in this document:

- Implemented components of the platform, including cloud computing infrastructure and management tools, data storage federation, HPC frontend and data validation components;
- Available functionality (from the user's perspective);
- Technologies applied in the implementation of each component and tool;
- Software engineering methods used.

This document is divided as follows:

- Section 1 introduces the WP2 prototype by briefly explaining the goals of WP2 and referencing its design document (Deliverable 2.2). An updated architecture diagram is presented and discussed. In addition, this section contains short, introductory descriptions of the work performed in each of technical tasks of WP2.
- Section 2 contains in-depth descriptions of the implementation progress achieved in each technical task of WP2 as well as the features supported by the first prototype (released in Project Month 12). A subsection has been prepared for each clearly identifiable technical component of WP2, explaining its features, status of prototype releases and ongoing work. In addition, any deviations from the original design (if present) are highlighted and explained.
- Section 3 describes the status of integration of WP2 with external tasks and Work Packages, focusing on the availability of end-user interfaces (which are covered by WP6) as well as the progress in identifying, preparing, registering and exposing Atomic Services (ASs) obtained from WP5 workflow.
- Section 4 contains closing remarks.

1 Status of the VPH-Share WP2 Architecture

As explained in the Project's Technical Annex [1] and discussed in detail in deliverable D2.2 [2], the goal of Work Package 2 is to provide a backbone for the VPH-Share computational and data storage services, enabling the application workflow representatives from Work Package 5, as well as the data storage and management teams from Work Package 4 to deploy their respective applications and data sources within a Cloud-based distributed computing infrastructure. Specifically, Work Package 2 fulfils the following goals:

- Integrate and oversee the Cloud-based computing and data storage resources made available to the Project;
- Expose a unified, heterogeneous, secure Cloud computing platform where application services and binary data repositories can be deployed and made available to authorised users without the need to install high-performance computing hardware at said users' local sites;
- Facilitate the development and deployment of the VPH-Share Atomic Services (please refer to deliverable D2.1 [3] for an explanation of what constitutes an Atomic Service and what features it offers to end users) on the Cloud resources managed by WP2;
- Provide extensions for traditional (Grid-based) high performance computing resources wherever required by application services;
- Facilitate access to potentially sensitive data sets by managing them with the use of the available Cloud data storage and monitoring their integrity, consistency and availability with automated, configurable tools;
- Collaborate with application and data storage providers as well as with partner projects (specifically, p-medicine) on providing an open, extensible infrastructure into which additional application services can be imported with minimum effort.

Whereas deliverable D2.2 provides an in-depth description of each technical component of the Work Package, the goal of this section is to explain which features (presented in D2.2) have been implemented and are integrated in the first prototype. In order to focus this discussion, we would like to refer the reader to an updated version of the Work Package 2 architecture diagram, which is presented in Figure 1.

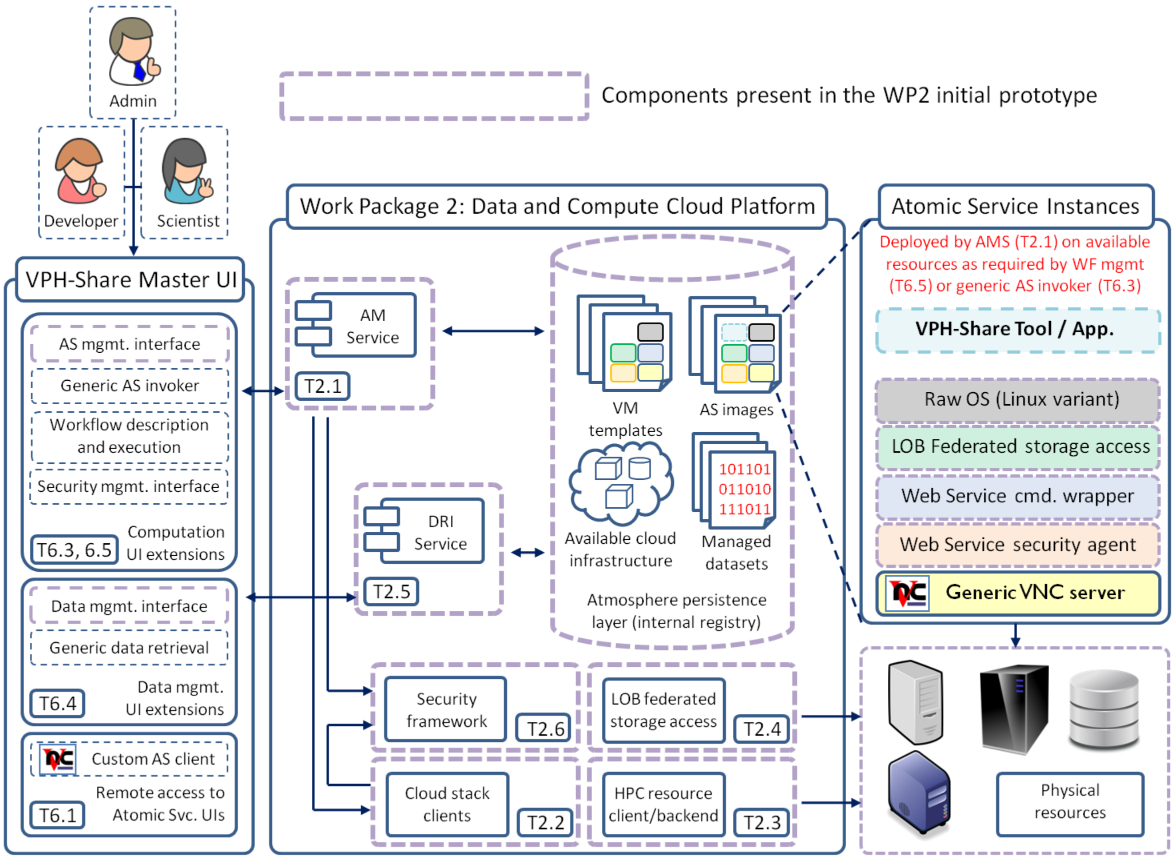

Admin
Developer
Scientist
Components present in the WP2 initial prototype
VPH-Share Master UI
AS mgmt. interface
Generic AS invoker
Workflow description and execution
Security mgmt. interface
T6.3, 6.5
Computation UI extensions
Data mgmt. interface
Generic data retrieval
T6.4
Data mgmt. UI extensions
Custom AS client
T6.1
Remote access to Atomic Svc. UIs
Work Package 2: Data and Compute Cloud Platform
AM Service
T2.1
DRI Service
T2.5
VM templates
AS images
Available cloud infrastructure
Managed datasets
Atmosphere persistence layer (internal registry)
Security framework
T2.6
LOB federated storage access
T2.4
Cloud stack clients
T2.2
HPC resource client/backend
T2.3
Atomic Service Instances
Deployed by AMS (T2.1) on available resources as required by WF mgmt (T6.5) or generic AS invoker (T6.3)
VPH-Share Tool / App.
Raw OS (Linux variant)
LOB Federated storage access
Web Service cmd. wrapper
Web Service security agent
Generic VNC server
Physical resources

Figure 1: VPH-Share Work Package 2 architecture as of Project Month 12 (conceptual view).

The central part of the diagram lists the WP2 application components and explains how these components map to technical tasks in WP2, according to the Project’s Description of Work [1].

Following the initial design period, concluded in Month 6, the technical tasks have commenced implementation of their assigned components, as shown in Figure 1. While the following chapter will contain further information regarding the status of development within each technical task, a general outline is presented below:

- The Cloud Resource Allocation Management framework, known as Atmosphere, is now capable of managing images of virtual machines and Atomic Services and it can also deploy Atomic Service Instances to the cloud resources provided by Task 2.2. Currently we support the OpenStack middleware platform. Support for EC2-compatible cloud stacks is under development.
- A basic Cloud Execution Environment is in place at CYFRONET, consisting of 16 four-CPU nodes to which VPH-Share Atomic Service Instances can be deployed. The newest version of the OpenStack Nova has been deployed and is available for development and testing purposes. In addition, an attached mass storage directory is provided and can be interfaced with the use of the OpenStack Swift protocol.

- A prototype service-based deployment of the HPC framework for VPH-Share has been deployed, based on the Application Hosting Environment (AHE) suite, as described in deliverable D2.2.
- The data access framework, known as LOBCDER, is now capable of exposing attached storage resources (including Swift-managed mass storage) as a simulated WebDAV (Web-based Distributed Authoring and Versioning) directory, which can in turn be mounted and accessed on the virtual machines used to host Atomic Service Instances.
- The Data Reliability and Integrity tool, in its prototype version, can register and periodically monitor managed datasets in the WP2 framework, ensuring their consistency and availability.
- Finally, an initial version of the WP2 security framework has been implemented – this includes Service Proxies capable of intercepting incoming calls and invoking an external policy decision point (PDP) to determine whether a given call should be cleared for execution.

Section 2 provides more information regarding the status of work in each of the tasks listed above.

1.1 End-User Functionality

From the perspective of the end user, the WP2 prototype provides the ability to do the following:

- A Virtual Machine can be spawned from one of the available templates. Atmosphere takes care of instantiating the Virtual Machine upon one of the available Cloud hosts, and returns to the developer a list of credentials that they can use to interact with this machine and deploy production services.
- Once the developer installs the services, the Virtual Machine can be registered as an Atomic Service in Atmosphere. Upon doing so Atmosphere takes over management of this VM, enabling it to provision the functionality of domain services when requested by the scientist or by automated tools (such as the workflow management tools developed in WP6) acting on behalf of the scientist.
- Atmosphere is capable of replicating Atomic Service Instances and spawning dedicated instances when requested by the developer.
- A basic version of the LOBCDER storage federation is in place, enabling authorised users to download and upload domain-specific data to Cloud data repositories by means of a WebDAV interface. This interface can be further mounted on the Virtual Machines (and, consequently, on Atomic Service Instances) as an element of the local file system, thus permitting application services to directly interact with Project data.
- An AHE frontend is supplied, enabling high-performance computational jobs to be submitted. In essence, the AHE frontend mimics an Atomic Service and can be managed by Atmosphere.
- The Data Reliability and Integrity service can be directed to inspect and validate datasets stored in Cloud resources and registered with the Project (using the initial prototype of the Atmosphere Internal Registry, which has also been deployed). At present, notifications

are dispatched to administrators in the form of e-mail messages, though in the future a dedicated notification system is expected to be built into the Master Interface.

- A basic version of the AS Security Proxy is in place. While a SSO (single sign-on) Project-wide security system has not yet been deployed, the Security Proxy enables Atomic Service Instances to be invoked in a secure manner. In addition, TLS (Transport Layer Security) is also provided.

1.2 Software Engineering Aspects

As far as software engineering methods are concerned, we follow the rapid prototyping approach, with short development cycles and frequent validation of results against the expectations of their intended users (whether the WP5 workflow teams or other technical tasks of VPH-Share). Progress reports are collected from WP2 developers on a monthly basis, and corrective actions introduced whenever requested by WP2 management. In addition, CYFRONET operates a WP-wide code repository at https://gforge.cyfronet.pl where the source code of individual components can be collected and managed. We intend to carry on with this scheme in the second year of the Project, focusing in particular on the specifics of end-user interfaces (which are developed as part of WP6, but which involve WP2 to a significant degree).

Table 1 presents a generalised list of technologies exploited by WP2 components. A more detailed technical description of each component can be found in deliverable D2.2 [2], while any discrepancies or deviations from the WP2 design documentation will be explained in Section 2 of this deliverable.

Table 1: List of technologies applicable to each component of the WP2 architecture.

Component	Applicable technologies
Cloud Resource Allocation Management (Atmosphere core services)	Java SE6 components deployed as OSGi bundles to an Apache Karaf container. Nagios framework for instance monitoring. Integration layer based on Apache Camel.
Atmosphere Internal Registry	Semantic registry layer implemented in Ruby on top of MongoDB storage. REST-based service endpoint and custom jQuery-based GUI.
Cloud Execution Environment	OpenStack (Diablo release) private cloud stack deployed at CYFRONET (HP ProLiant BL2x220c G5, 2 x Intel Xeon L5420, 16GB RAM, 120 GB internal HDD); 3 TB attached storage (NFS).
High Performance Execution Environment	Middleware connections for QcG (using the QcG Java SDK), Unicore (using the UCC java library) and Globus (using the JGlobus library). SOAP-based service endpoint.
Data Storage Federation	Support for OpenStack Swift storage resources; data exposed by a WebDAV-like API (mimicking a true WebDAV server). Service-based control endpoint for management actions.
Data Reliability and Integrity	Standalone Java-based service, integrated with OpenStack storage. Managed dataset metadata stored in AIR. Service-based control endpoint for management actions.
WP2 security components	Java-based Security Proxy implemented as plugin for the Apache server framework, deployable on any Atomic Service template.

2 FEATURES OF THE FIRST WP2 PROTOTYPE

The goal of this section is to detail the features of each of the technical components which comprise the WP2 platform, present their integration with external components and outline ongoing development activities in each technical task.

2.1 Cloud Resource Allocation Management

At the core of the Cloud resource management infrastructure lies the Atmosphere Management Service (AMS). AMS is responsible for optimising utilisation of computational resources and will be used by:

- Application providers, who need to select an appropriate virtual machine template, install and configure their software and save the Virtual Machine as an Atomic Service.
- Scientists, who will specify their requirements in terms of the required functionality (as a list of Atomic Services and their configurations) and will be provided with a pool of resources that is monitored and managed automatically by AMS. This includes interaction with single Atomic Service Instances as well as running workflows which access a set of Atomic Service Instances.

For a more detailed description of AMS features please refer to Section 4.1.1 of deliverable D2.2 [2].

The internal architecture of AMS and its interactions with other WP2 subsystems are illustrated in Figure 1. The main component of AMS is the Manager, which supervises the process of preparing an optimal deployment plan and provisioning resources. It exposes the AMS functionality to the Cloud Facade and accepts requests from clients. The Manager is also responsible for maintaining a representation of the infrastructure, available Atomic Services and AS Instances in a dedicated registry called the Atmosphere Internal Registry (AIR), described in Section 2.2. When a new request arrives, the Manager queries AIR for available resources and invokes the Optimizer to prepare a deployment plan that will ensure optimal resource allocation. The Manager then enacts the deployment plan by using the Cloud Client to manage resources in the Cloud Execution Environment (CEE), and the Proxy Controller Client to register Atomic Service Instances in the HTTP reverse proxy. A detailed description of this process can be found in Section 4.1.7 of deliverable D2.2 [2].

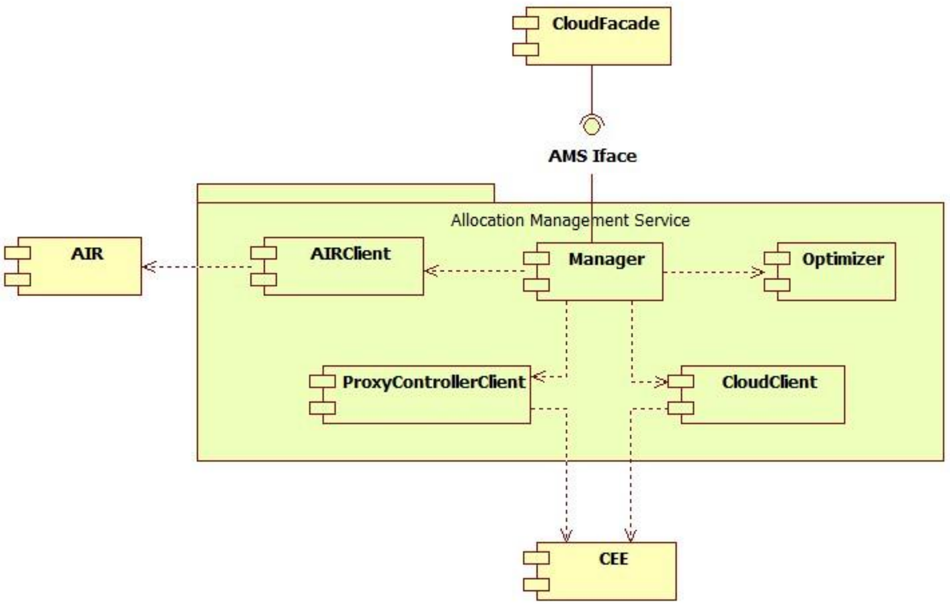


Figure 2: Architecture diagram of Allocation Management Service in the scope of WP2 tools. The light green box indicates internal AMS components while yellow boxes indicate other WP2 subsystems that AMS depends on (i.e. the Atmosphere Internal registry and the Cloud Execution Environment) as well as the Cloud Facade which exposes AMS functionality through REST and Web Service interfaces.

2.1.1 Implementation status

The initial prototype of the framework includes all Allocation Management Service components depicted in Figure 2. The candidate technologies mentioned in Section 4.1.8 of deliverable D2.2 [2] are being used to implement the actual prototype. All AMS components are implemented in Java as OSGI [4] bundles and deployed in a Karaf [5] container. Additionally, AIR and Proxy Controller Clients employ the Camel integration framework [6] that facilitates communication with external services. The Cloud Client is based on the JClouds [7] library. The Optimizer component implements a simple optimisation algorithm in Java that relies on the number of clients using a specific Atomic Service Instance.

The Allocation Management Service prototype provides all the features required by users to start interacting with the WP2 Data and Compute Platform. It implements end-user services (see Section 4.1.1 of deliverable D2.2 [2] for a full list of AMS features), such as:

- Browsing the available templates in AIR
- Instantiating Virtual Machines from selected templates
- Saving Virtual Machines as Atomic Services
- Requesting instances of specified Atomic Services

The prototype supports all the identified use cases, albeit in a very minimalistic manner. This can be explained by considering the fourth item in the above list. The AMS prototype is able to start Atomic Service Instances only in a private Cloud infrastructure, and register them in an HTTP proxy to make them publicly accessible. It does not yet collect monitoring data from Atomic Service Instances and does not take into account required data replication and security constraints – thus, the deployment plans prepared by the Optimizer are far from being optimal and Atomic Service Instances are not subject to automatic scaling.

2.1.2 Allocation Management Service metadata schema

In order to fulfil its responsibilities, AMS maintains a representation of the Cloud infrastructure, Atomic Services and Atomic Service Instances. A simplified view of this internal representation is depicted in Figure 3. At the topmost level lies the **ExecutionEnvironment**, composed of **DataSources** and **ComputeSites**. The former represent data storage services such as Amazon S3 or OpenStack Swift, while the latter model commercial data centres, private cloud deployments, and HPC infrastructures. Each **DataSource** and **ComputeSite** may be associated with a **CostModel**, which defines the expenses for storing and transferring data or running virtual machines of a specific size. **ComputeSites** are composed of **Hosts** which, in the case of private cloud providers, represent physical machines and their capacity. A **Host** should be treated as a physical computational resource which is capable of hosting VPH-Share Atomic Services. Since information about physical machines is not available for public Cloud infrastructures each **ComputeSite** of this type will have only one **Host** that represents the resources available for single cloud client (i.e. Atmosphere). **Hosts** run virtual machines with VPH-Share applications (Atomic Service Instances) represented by **Appliances**. Virtual Machines are characterised by **Load** (CPU, memory and disk usage statistics) and **Performance** (duration of a single request or number of requests per second). Each **Appliance** is assigned a certain amount of resources (CPU, memory and disk storage, expressed as Cloud instance size) represented by **ResourceAllocation**. **Appliances** may be instantiated from and saved as **ApplianceTemplates**. **AvailableAppliance** is a base class for **Appliance** and **ApplianceTemplate** classes. It encapsulates attributes and methods common for both of these classes. Examples of such common attributes include **ApplianceState**, which represents the condition of a resource (whether it is running, stopped etc.) and **ApplianceType**, determined on the basis of what type of Application (i.e. arbitrary VPH-Share process) is installed. An **Appliance** requires an **InitialConfiguration** which contains the initialisation context for a given Virtual Machine (for instance a root public SSH key), data required to properly configure and start the hosted **Application** as well as a list of **DataSources** that will be used by the application (required for deployment plan optimisation). Applications consume and produce data represented by **LogicalData** items that are grouped in **DataSets** retrieved from/stored in **DataSources**. Applications and **LogicalData** may have **SecurityConstraints** associated with them that will – for instance – restrict the list of **CloudSites** at which sensitive data may be processed. **DependentWorkflow** models any application that requires the availability of specific Atomic Service Instances.

Classes that are coloured light green are persisted in the Atmosphere Internal Registry.

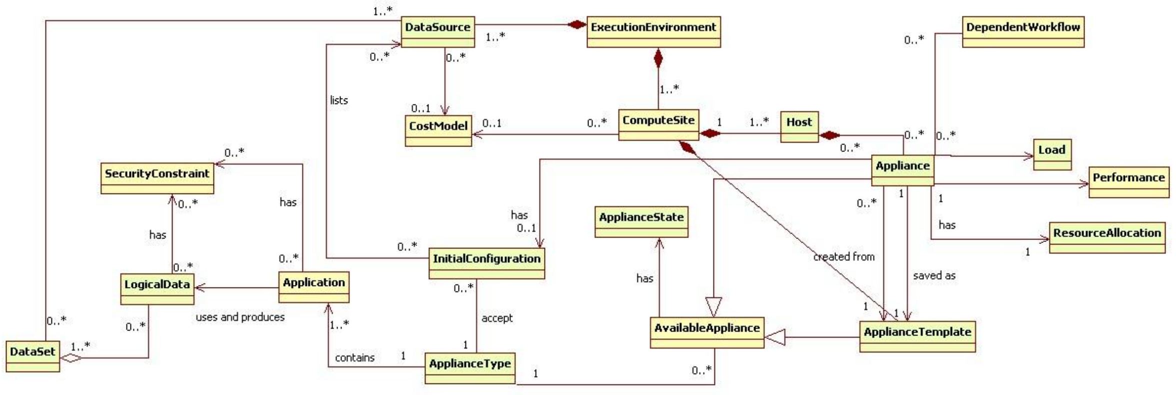


Figure 3: UML class diagram of the internal representation of the Cloud infrastructure, Atomic Services and Atomic Service Instances used by the Allocation Management Service. Classes coloured light green are persisted in the Atmosphere Internal Registry.

2.1.3 Deviations from proposed design

In the original design (see Section 4.1.2 in deliverable D2.2 [2]) it was stated that AIR is a component of the Allocation Management System; however we have since discovered that AIR functionality is also useful for other WP2 tools, such as DRI. Therefore, AIR has been extracted from AMS as a standalone subsystem and will be separately described in the next section of this deliverable. It was also assumed that AMS would expose its interface as a REST or Web Service endpoint. During development it turned out that it is more convenient to provide a single entry point for all WP2 remote endpoints and publish it through the Cloud Facade. Hence, AMS will only provide a local OSGI [4] service for the Cloud Facade.

2.1.4 Ongoing development and future work

During the coming months work on AMS will focus on collecting online monitoring data about the availability and load of Virtual Machines, and storing such data in AIR. A dedicated AMS component, called the Monitoring Controller Client will be developed. It will be responsible for registering and unregistering Atomic Service Instances in the Nagios monitoring framework deployed in CEE. We will also focus on enhanced optimisation algorithms, taking into account the load of Atomic Service Instances, rather than just their availability.

2.2 Atmosphere Internal Registry

The Atmosphere Internal Registry (AIR) is responsible for delivering the persistence layer and inter-component exchange mechanism for metadata regarding the Project's computing and data cloud. This information is stored, shared and consumed by components of the VPH-Share software; particularly by elements of the Atmosphere cloud computing and data provisioning platform (see Section 2.1.2 for details). Conceptually, AIR should be treated as part of Task 2.1; however due to the fact that it is implemented as a standalone component and its functionality is not limited to supporting the AMS service, we have decided to describe it in a separate top-level subsection of this deliverable.

The main duties of AIR (see Section 2, Figure 2 in deliverable D2.2 [2]) involve establishing a common domain model for VPH-Share metadata exchange, providing a suitable persistence mechanism to securely store and manage that metadata and exposing a set of appropriate API (Application Programming Interface) operations for the other entities in VPH-Share to use that functionality for their purpose.

The following sections describe in detail the state of the AIR prototype, deployed as an internal VPH-Share Consortium tool, as it existed during the publication of this deliverable.

2.2.1 Description of the first AIR prototype

The initial AIR prototype was deployed in late 2011. The development plan for AIR assumes tight, "small-delta" incremental development cycles with a SaaS-like deployment and management model. Each cycle consists (with occasional deviations) of the following steps:

1. Going through all use cases and user stories described so far as external requirements for AIR; filtering out and choosing the most important, relevant and urgent cases for the present cycle.
2. Communication with relevant parties who authored the request (which may include the user community, application developers or other tool/platform developers) in order to adjust the use case to the current state of affairs and to address all issues in detail so it is ready for implementation.
3. Designing the changes required for AIR to support the cases and implementing the changes; updating the online documentation where needed.
4. In-situ deployment of the new version of AIR and broadcast communication with the Consortium to announce the new version and stress the most important changes.
5. Direct communication with the parties mentioned in point 2 of this list in order to obtain acceptance for the changes made.

The initial prototype follows these assumptions and has already gone through two development cycles (one related to Data Source metadata and another for Appliance and Atomic Service metadata).

2.2.2 Architecture and Deployment Mechanism

Regarding the internal architecture of AIR, there were no deviations with respect to what was planned during the design phase (see Section 4.1.2 of deliverable D2.2 [2]). The prototype consists of three main building blocks (see Figure 4).

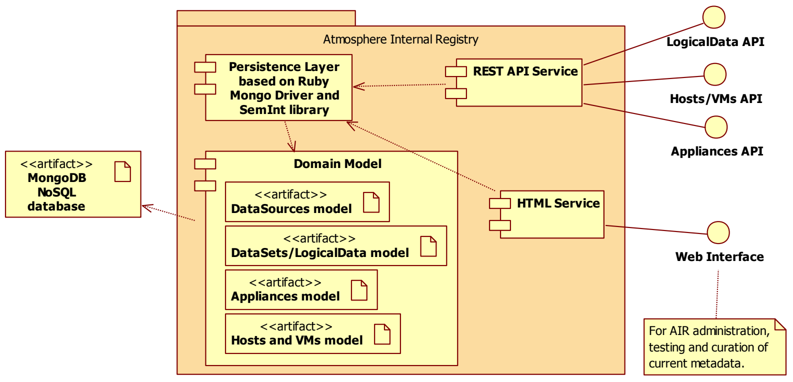

Atmosphere Internal Registry
Persistence Layer based on Ruby Mongo Driver and SemInt library
REST API Service
LogicalData API
Hosts/VMs API
Appliances API
<<artifact>> MongoDB NoSQL database
Domain Model
<<artifact>> DataSources model
<<artifact>> DataSets/LogicalData model
<<artifact>> Appliances model
<<artifact>> Hosts and VMs model
HTML Service
Web Interface
For AIR administration, testing and curation of current metadata.

Figure 4: Deployment model used for the first prototype of AIR. During subsequent development cycles both the Domain Model and the API are expected to grow considerably, while other elements should remain relatively stable.

Currently, the prototype stores and publishes information about four elements of the Atmosphere resource pool. The data management part is described with **DataSources**, **DataSets** and **LogicalData** concepts (see Section 2.5) while the Cloud computing elements are described with **Host**, **VM** (Virtual Machine), **ApplianceType** and **InitialConfiguration** concepts (see Section 2.1.2).

The set of technologies used is also in line with the initial assumptions made at the design stage. The persistence layer is provided with an installation of the MongoDB schemaless database, while the domain models, defined using the Semantic Integration [8] methodology, provide semantic abstractions (concepts) over the data inside that database. The *controller* layer (according to the Model-View-Controller principle; not shown in Figure 4 for clarity) manages those abstractions for the sake of both HTTP APIs and Web-based GUIs.

2.2.3 Interfaces of the AIR Prototype

The prototype provides two types of interfaces – a HTTP API through which other applications and clients can use AIR automatically, and a basic Web UI, targeted for human users.

The current set of API calls supported by the AIR prototype is given in Table 2.

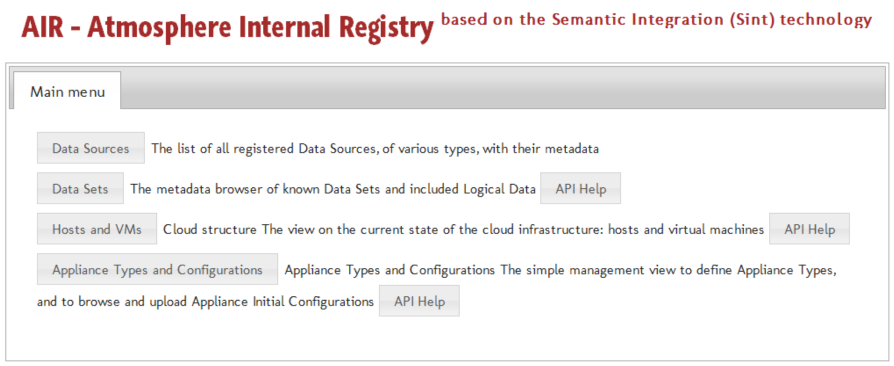
Table 2: Complete list of all API calls available in the first prototype of AIR (February 2012). A detailed manual on the supported API operations is available online, together with the prototype.

AIR model	API operation call header	Meaning
DataSource management	`GET: get_managed_data_sets`	Get all Data Sets that have the flag is_managed set.
	`GET: get_logical_data_for_data_set`	Retrieves all Logical Data items registered for given Data Set of ID dsid. Please use the internal AIR ID for the Data Set.
	`POST: update_dri_checksum`	Updates the value of the Logical Data item checksum computed by DRI (the dri_checksum field).

AIR model	API operation call header	Meaning
Appliance management	`GET: get_appliance_config`	Downloads the payload of the configuration file with configuration ID equal to conf_id. The AIR internal identification system is used for configuration ID.
	`GET: get_vms_by_appliance_type`	Gets the JSON array of VMs which are currently set to be of certain appliance type. The type is determined by the type of the appliance configuration identified with the configuration ID equal to conf_id.
	`POST: upload_appliance_config`	Uploads a configuration string as a new appliance configuration.
	`GET: get_running_vm_config`	Returns the JSON-encoded configuration object with the 200 code on success or a 400 and an error message on failure. The returned object structure is given below.
	`GET: get_appliance_type_for_config`	Gets the JSON structure with all Appliance Type fields. The Appliance Type is chosen according to the Initial Configuration ID passed as the conf_id parameter.
	`GET: get_appliance_type_for_vm`	Gets the JSON structure with all Appliance Type fields. The Appliance Type is chosen according to the VM identified with the vms_id parameter.
Host and VM management	`POST: add_host`	'Upserts' another host to the topology - adds it if no such host is found in the data or updates altered fields if the host was registered before.
	`POST: add_vms`	'Upserts' another VM server or template to the topology - adds it if no such VM is found in the data or updates altered fields if the VM was registered before.
	`GET: get_running_vm_specs`	Generates a JSON file with detailed list of hosts and running VM, along with the specification and used/free resources.
	`DELETE: remove_vms`	Removes the VM metadata for appliance identified by vms_id.
	`DELETE: remove_host`	Removes the host metadata from the registry, where the host is identified by host_id.

Note that the list in Table 2 is not closed – rather, it is expected to grow and change during future implementation cycles of AIR, according to the evolution of VPH-Share user requirements and software. All the listed operations are secured with basic (HTTP) authentication protocol at the time of publication of this deliverable. Future AIR versions are expected to be secured with specific VPH-Share security mechanisms developed in Task 2.6.

The second type of interface is a Web GUI (see Figure 5 and Figure 6 for sample screenshots).



AIR - Atmosphere Internal Registry based on the Semantic Integration (Sint) technology

Main menu

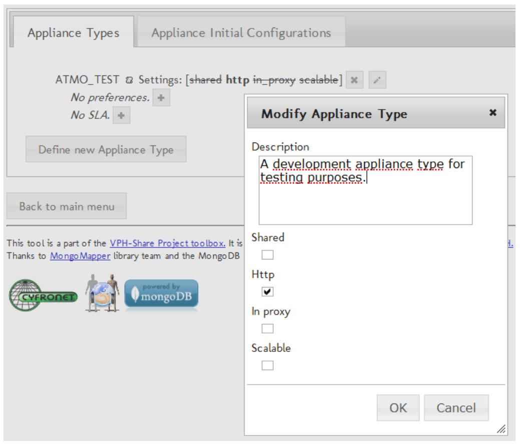
Data Sources The list of all registered Data Sources, of various types, with their metadata

Data Sets The metadata browser of known Data Sets and included Logical Data API Help

Hosts and VMs Cloud structure The view on the current state of the cloud infrastructure: hosts and virtual machines API Help

Appliance Types and Configurations Appliance Types and Configurations The simple management view to define Appliance Types, and to browse and upload Appliance Initial Configurations API Help

Figure 5: First AIR prototype – main menu of the Web user interface.



Appliance Types | Appliance Initial Configurations

ATMO_TEST Settings: [~~shared~~ **http** ~~in_proxy~~ ~~scalable~~]

No preferences.

No SLA.

Define new Appliance Type

Back to main menu

This tool is a part of the VPH-Share Project toolbox. It is

Thanks to MongoMapper library team and the MongoDB

Modify Appliance Type

Description

A development appliance type for testing purposes.

Shared ☐

Http ✓

In proxy ☐

Scalable ☐

OK Cancel

Figure 6: First AIR prototype – sample administrator dialog for management of registered Appliance Types.

The AIR Web user interface is not meant to be used by the actual end users in the VPH-Share community, i.e., the scientists. Instead, the aim is to provide application developers and system administrators (see p.11 in deliverable D2.2 [2] for user group definitions) the ability to register new entities inside AIR, check if the metadata stored is valid and not outdated, and optimise the mechanisms of the Atmosphere cloud provisioning platform according to user requirements (such optimisation can be effected e.g. through careful definition of the available Appliance Types; see Figure 6).

2.2.4 Cloud Data Source harvesting

The final feature provided with the first prototype of AIR is the cloud data sources harvesting ability. This feature is available offline, as a set of dedicated scripts that can be used by Atmosphere system administrators (due to the powerful impact of such an action on the metadata stored in AIR, it is not meant to be available online).

The purpose of those harvesting scripts is to connect to a predefined Cloud data source instance, use the provided credentials to list all the **DataSets** and **LogicalData** objects stored there, then transfer any useful metadata about those objects back to AIR. By periodically executing this harvester, Atmosphere administrators may easily keep the internal registry in sync with any external Cloud data sources and may also add new data sources to the platform.

Currently, the harvesting scripts support OpenStack Swift servers as well as Amazon EC2 data catalogues. Upon executing them, the user has to supply the credentials required by AIR to access such sources, as no such credentials are presently stored in AIR.

2.2.5 Future Development Plans

The current AIR prototype is already integrated with AMS (Allocation Management Service) and DRI (Data Reliability and Integrity) services. Future development plans for AIR are as follows (in no particular order):

- Integration with LOBCDER (Large OBject Cloud Data storage fedERation) to establish a common metadata space for **DataSources** used in the Project.
- Integration with the main Metadata Management platform (WP6) in order to integrate the concepts and terms used inside the Atmosphere platform with the semantic terminology used by the VPH-Share end user communities.
- Development of more sophisticated use cases for both AMS and DRI components.

Carrying out those plans is expected to follow the same tight-cycle development methodology as was used to provide the current prototype of AIR (see Section 2.2.1).

2.3 Cloud Execution Environment

2.3.1 Structure of the existing Cloud platform

The Cloud Execution Environment (CEE) constitutes a specialised platform, which is required by:

- The Atmosphere component described in Section 2.1, to enable low-level operations on Virtual Machines, such as managing the VM lifecycle or preparing snapshots that can be used as templates.
- The WP3 tools, as a Private Cloud storage backend used by such tools to store and manipulate data.

CEE is composed of several components, shown in Figure 7, some of which are off-the-shelf while others are custom-developed.

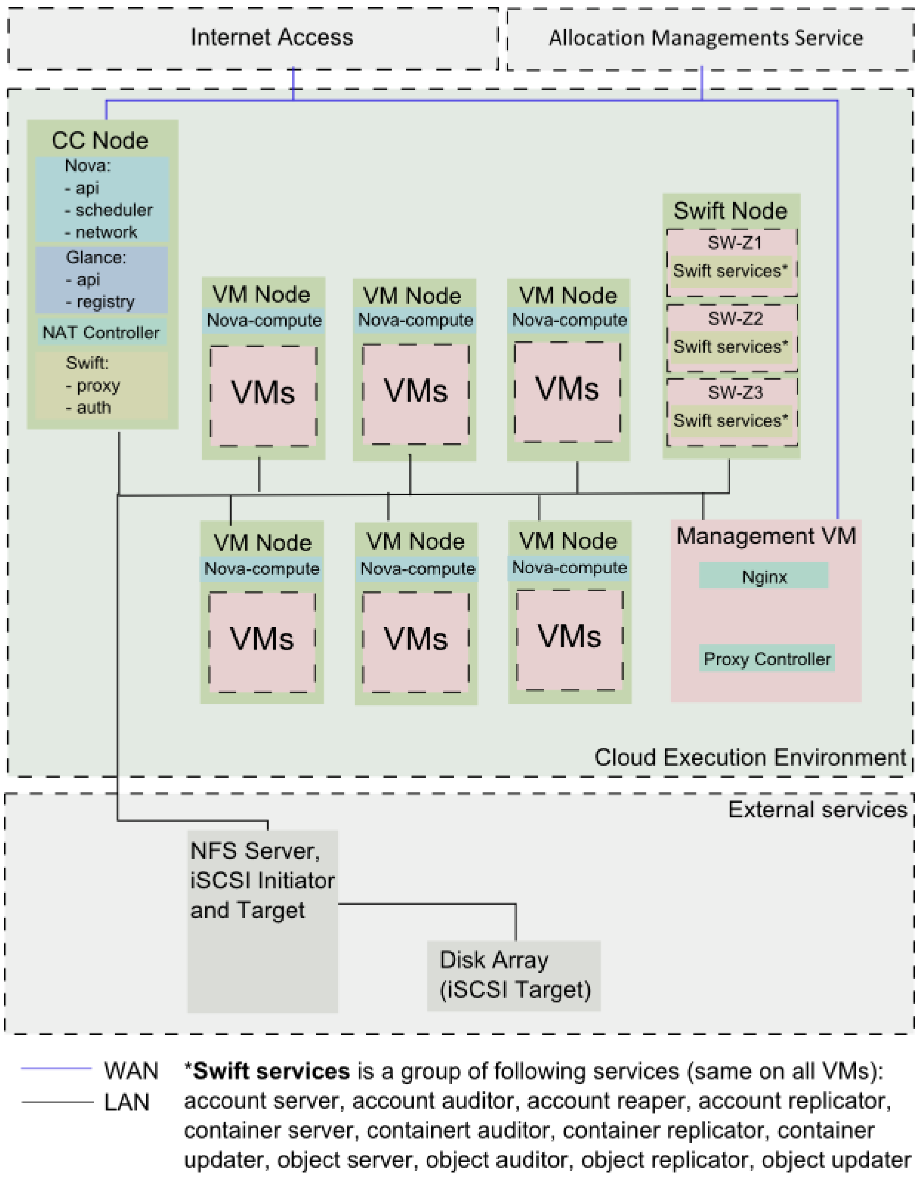

Internet Access
Allocation Managements Service
CC Node
Nova:
- api
- scheduler
- network
Glance:
- api
- registry
NAT Controller
Swift:
- proxy
- auth
VM Node
Nova-compute
VMs
Swift Node
SW-Z1
Swift services*
SW-Z2
Swift services*
SW-Z3
Swift services*
Management VM
Nginx
Proxy Controller
Cloud Execution Environment
External services
NFS Server, iSCSI Initiator and Target
Disk Array (iSCSI Target)
WAN
LAN
*Swift services is a group of following services (same on all VMs): account server, account auditor, account reaper, account replicator, container server, containert auditor, container replicator, container updater, object server, object auditor, object replicator, object updater

Figure 7: Architecture of the Cloud Execution Environment (light green area in the middle), including OpenStack nodes (CC, VM, Swift), additional management VM (for the Nginx-based reverse proxy) as well as external components.

The main components are elements of the OpenStack [9] middleware suite (described in more detail later on in this document) responsible for providing computational and storage features for the Private Cloud installation.

For the computational part, we have thus far deployed 7 physical nodes with identical hardware specifications (HP ProLiant BL2x220c G5, 2 x Intel Xeon L5420, 16GB RAM, 120 GB internal HDD). One of them acts as the Cloud Controller (CC) and the rest are used to run VMs. All nodes have access to approximately 3 TB of external shared storage space (NFS on iSCSI volume) backed by a disk array with fast (15000 RPM) SAS hard drives. This shared space is used to store VM templates and images of running Atomic Services. We have also deployed infrastructure monitoring tools which allow us to judge that, at the current stage of the project, the presented resources are more than sufficient; however we are ready to extend our resource pool if demand increases. All mentioned nodes are connected to an Ethernet switch with support for 802.1Q VLANs using 1 Gbps ports (switch uplink port is 10 Gbps). As such, the physical network layout is consistent with the description and diagram presented in Section 4.2.3 of deliverable D2.2 [2]. According to the description contained in that document, this layout enables us to choose any network mode available for Open Stack. As a result, we have decided to use the most powerful VLAN-based network setup. At present we use 2 VLANs – one for physical nodes and one for VPH VMs, which provides L2 level separation. However, if required (e.g. for security reasons), this deployment allows us to provide additional VLANs for multiple VM groups that should be separated. Finally, in addition to internal LANs (using private IP addresses) the CC Node is connected to the Internet (WAN) and has a public IP address. Thus, the CC node can act as a NAT-enabled router for the remaining physical nodes and VMs, including both SNAT (for all IP outbound traffic) and DNAT (for some inbound traffic on predefined TCP/UDP ports) between cloud VMs in a private LAN and the Internet.

The DNAT (Destination Network Address Translation) mechanism is fully remotely configurable via the **NAT Controller** – a lightweight service written in Python, exposing a RESTful API, and is used by **AMS** to provide TCP/UDP port redirection for any network service (other than Atomic Service Instance HTTP(s) endpoints) such as SSH or VNC. ASI, on the other hand, required a more intelligent solution enabling redirection of traffic based on deeper packet inspection (at the application layer) for which we have proposed a solution based on the **Nginx** [10] server acting as a reverse proxy. It is configured using the **Proxy Controller**, a lightweight service written in Ruby that exposes a RESTful API (with the help of the Sinatra library). The Proxy Controller is used by the **Proxy Controller Client** (part of AMS) to register and unregister ASIs. All components of the proxy are deployed on a dedicated VM (marked in the figure as “management VM”), directly connected to both the Internet (public IP) and all LANs (private IPs). Both mechanisms (NAT and the proxy) facilitate conservation of the public address space which should not be wasted needlessly, especially given the exhaustion of IANA’s IPv4 pool and near-exhaustion of RIRs pools, such as RIPE NCC responsible for allocations in Europe [11]. Unfortunately, direct use of an IPv6-only network (with public addresses) for our Cloud installation is not possible as IPv6 support in the public Internet is still lagging; however OpenStack already supports this protocol and it remains a possible option for the future. Note that the use of dual-stack (private IPv4 network and public IPv6 addresses) solutions would preclude direct access from IPv4 clients (NAT/Proxy mechanisms still would be required).

The storage component of our private Cloud is currently deployed on two physical nodes. Its frontend is deployed on the CC node, while storage nodes are provided as three separate VMs on a dedicated physical node (Swift Node with the same specification as computational nodes). Each VM has access to ca. 500 GB space on a dedicated (not shared) volume, backed by the aforementioned external disk array. As suggested by OpenStack developers, the XFS file system is used on those volumes. As data is stored by our installation of Open Stack in a triple-redundant manner, ca. 500 GB of space is available; however, just like the computational part of our Cloud, the storage space can be flexibly increased if required.

2.3.2 Cloud middleware stack

As foreseen during the design phase, we have deployed the OpenStack middleware suite in our private Cloud infrastructure. The currently installed release is Diablo, which at the time of preparation of this document, is the most current stable version of OpenStack.

More specifically, we have deployed all three "core" components of this release:

- Computing (Nova) – directly responsible for managing the lifecycle of VMs and ASIs. It provides an external REST-based OpenStack Compute API (a.k.a. Nova API, currently in version 1.1) which is used by the **Cloud Client** component of Atmosphere (by way of the JClouds [7] library).
- Image Service (Glance) – this enables storage of VM templates used to instantiate Atomic Service Instances. It also provides a REST-based API (called the OpenStack Image Service API; a.k.a. Glance API, currently in version 1.0) however this API is only used internally by the stack and not exposed to the rest of the platform as it is not required by external clients.
- Object Storage (Swift) – for the storage part of the Private Cloud used by WP3 tools and WP5 services. Like other parts it also provides a REST-based "OpenStack Object Storage" API (a.k.a. Swift API, currently in version 1.0), which is exposed for external clients and can be interfaced by the LOBCDER storage federation described in Section 2.5.

We have performed standard manual software installation using official package repositories as described in the project documentation. In accordance with this description we have used Ubuntu (10.04 LTS) on all nodes.

2.3.3 Infrastructure provisioning status

We have thus far provisioned several VMs to serve the needs of VPH application and service developers. Table 3 summarises our activity in this regard.

Table 3: Virtual Machines provisioned to VPH-Share developers as of Project Month 12.

VM no.	Allocated resources	Recipient	Purpose
1	1 VCPU, 2 GB RAM, 5 GB HDD	David Chang [UCL]	VM for hosting AHE ACD software
2	1 VCPU, 2 GB RAM, 5 GB HDD	Breanndan O'Nuallain [UVA]	VM for development and testing of the ViroLab workflow
3	1 VCPU, 2 GB RAM, 5 GB HDD	Martin Steghöfer [UPF]	VM for the Taverna Server

VM no.	Allocated resources	Recipient	Purpose
4	1 VCPU, 2 GB RAM, 5 GB HDD	Xavier Planes [UPF]	Non-interactive part of the @neurIST workflow
5	1 VCPU, 2 GB RAM, 5 GB HDD	Xavier Planes [UPF]	Interactive part of the @neurIST workflow
6	1 VCPU, 2 GB RAM, 10 GB HDD	Eric Kerfoot	VM for Atomic Services used as part of the euHeart workflow
7	1 VCPU, 2 GB RAM, 25 GB HDD	Xavier Planes [UPF]	VM for the ANSYS simulation package
8	1 VCPU, 2 GB RAM, 10 GB HDD	[ATOS]	euHeart workflow VM clone to enable development and testing of the security framework

In addition to these resources we have also allocated space on storage part of the cloud for:

- Testing the WP3 tools during their development process.
- Storage of data required to test the @neurIST workflow.

2.3.4 Ongoing work

Overall, the deployment of Cloud middleware was smooth; however we had to overcome several minor glitches, while other issues may still require some work.

The issues already addressed include:

- Lack of a built-in mechanism for redirecting TCP/UDP ports – there is a mechanism for redirection of public IPs through DNAT, however, as already explained, we wanted to conserve public IPs. As a result, we have developed the previously described NAT Controller.
- Minor incompatibilities between the OpenStack DB and management tools – this required manual updates of the network-related portion of the DB to allow proper extraction of VM IPs through the OpenStack API using JClouds.
- JClouds bends the HTTP protocol (by providing an "Accept" header in a format not defined by the standard); however the Nova client responsible for handling these requests performs very strict checking, which, in turn leads to errors which prevent destroying and restarting VMs. A fix for this problem was proposed to (and accepted by) the developers and a patched version has been installed (cf. ticket number 794 in JClouds issue tracker – https://code.google.com/p/jclouds/issues/detail?id=794).

One additional problem which still needs to be resolved relates to errors in handling VM snapshot requests issued to the Nova API by JClouds (this, however, works well when using the official client and hence may be a JClouds issue).

Additionally, we plan to deploy a dedicated Nagios-based monitoring infrastructure for the VMs and ASIs, as well as a **Monitoring Controller** that would enable AMS to register/unregister ASIs.

2.4 High-Performance Execution Environment

The Application Hosting Environment (AHE) 3.0 is a lightweight middleware suite which virtualises grid applications and exposes their features as RESTful web services. Grid middleware tools are complicated applications with a steep learning curve. This is often a troublesome issue for scientific end users. AHE attempts to hide the complexity of the underlying HPC resources by providing a lightweight layer between the grid middleware and the user. An expert user is required to setup AHE with information on how and where to execute an application. Once this is completed, the scientific end user can execute the application through a set of simple RESTful web service commands.

The Audited Credential Delegation application (ACD) provides a secure solution that audits, authenticates and authorises user commands. It also provides virtual organisation management as well as credential delegation features. ACD is implemented as a RESTful web service and, in conjunction with AHE, provides a simple way to handle grid credentials, and manage data and computational jobs.

AHE and ACD can be accessed using RESTful web service commands. AHE will access the underlying grid middleware using a number of Java middleware client libraries, including UCC (unicore), JGlobus (Globus) and QcG to launch applications that have already been installed in the relevant grid infrastructure. AHE will use ACD to authenticate and authorise the user and generate proxy certificates on behalf of the user.

2.4.1 Current Status of the Prototype

A prototype of AHE has been implemented using Java. The current prototype can be deployed in two different versions: as a standalone Java application using an embedded Jetty Server, or as a servlet that can be deployed on a servlet-compliant server such as Apache Tomcat. AHE 3.0 has the following features implemented: AHE Core library, workflow engine, middleware connectors and RESTful web services.

The AHE core library consists of the core data structures and libraries that implement AHE functionality – these include user, security, application, resource, and workflow management. AHE is currently able to create and edit user information and security credentials. This allows AHE to authenticate or map the user to ACD for authentication and authorisation. It is also able to set up a credential to connect to the underlying grid middleware. An application and resource registry has been implemented in AHE using the Hibernate ORM (Object-Relational Mapping) framework, which allows AHE to set up virtual applications on corresponding resources. These applications translate into VPH-Share Atomic Services and can be managed by Atmosphere. The main features enabling AHE to create and initiate virtual applications have also been implemented. This allows the user to find the application they wish to launch and submit it, along with the required input data, to the resource they desire. When the user initiates a virtual application or workflow, the persistent workflow engine starts a new workflow and proceeds through successive workflow stages. The AHE workflow engine enables simple job submission with pre- and post-processing stages to be implemented, it also allows more complicated workflows for virtual applications (such as error recovery) to be created.

AHE currently includes a number of middleware connectors: QcG (using the QcG Java SDK), Unicore (using the UCC java library) and Globus (using the JGlobus library). AHE is able to generate a submission object compliant with those connectors from the application execution details and configurations provided by the user.

A RESTful web service for AHE has been implemented. The interface of this service is presented in Table 4.

Table 4: AHE service API.

Command	HTTP Method	Resource	Comment
`AddUser`	Post	CMD	Add User
`EditUser`	Post	User	Edit user
`ListUsers`	Get	CMD	List all user
`RemoveUser`	Delete	User	Remove User
`AddCredential`	Post	CMD	Add credential
`EditCredential`	Post	Credential	Edit credential
`ListCredential`	Get	CMD	List credentials
`RemoveCredential`	Delete	Credential	Remove credential
`AddUserCredential`	Post	User	Add Credential to user
`RemoveUserCredential`	Delete	User	Remove credential from user
`Status`	Get	CMD	Get status of App Instance
`SetDataStaging`	Post	AppInstance	Set Data staging for both stage in and out
`SetStageIn`	Post	AppInstance	Set Stage In command
`SetStageOut`	Post	AppInstance	Set Stage out command

Command	HTTP Method	Resource	Comment
`GetDataStaging`	Get	AppInstance	Get data staging information
`GetStageIn`	Get	AppInstance	Get date stage in information
`GetStageOut`	Get	AppInstance	Get data stage out information
`Prepare`	Post	CMD	Prepare virtual application
`Start`	Post	AppInstance	Start application / workflow
`Terminate`	Delete	AppInstance	Terminate
`ListJobs`	Get	CMD	List all jobs
`SetProperty`	Post	AppInstance	Set Application instance property
`GetProperty`	Get	AppInstance	Get application instance property
`ListProperties`	Get	AppInstance	List all application instance properties
`CreateResource`	Post	CMD	Create a resource in the resource registry
`EditResource`	Post	Resource	Edit a resource in the registry
`ListResource`	Get	Resource	List all resources in the registry
`CreateApp`	Post	CMD	Create an application in the application registry
`EditApp`	Post	AppReg	Edit an application in the application registry
`ListApp`	Get	AppReg	List all applications in the registry

The Audited Credential Delegation (ACD) application has also been implemented as a RESTful web service. Currently, ACD can be deployed as a standalone application using an embedded Jetty server or as a servlet container deployable on servlet compliant servers such as Apache Tomcat.

ACD is currently able to audit commands issued by the user. It is also able to set up virtual organisations (VO) including member management and group certificate setup, allowing ACD to generate proxy certificate for each member of the VO. Finally, ACD is able to authenticate and authorise the user based on their roles (administrator or scientist). It supports a Shibboleth authentication mechanism, as well as local username/password database authentication.

A summary of ACD RESTful web service commands can be found in Table 5.

Table 5: ACD service API.

Command	HTTP Method	Resource	Comment
`createlocalACDAccount`	Post	User	Create New ACD account
`createNewVO`	Post	CMD	Create New Virtual Organisation
`generateProxies`	Post	CMD	Generate Proxy
`registerUser`	Post	CMD	Register User to VO
`getACDAudit`	Get	CMD	Get ACD adiut
`getCertificateDetail`	Get	VO	Get certificate details
`getRoleAssignment`	Get	ACD Credential	Get role assignment
`getuserACDAccounts`	Get	User	Get User ACD account
`getUserVOs`	Get	ACD Credential	Get user VO
`viewAllRoles`	Get	CMD	View all roles
`viewCurrentVOs`	Get	CMD	View current VO
`assignP12CertToVO`	Post	VO	Assign P12 certificate to VO
`assignUserToRole`	Post	ACD Credential	Assign user to role

Command	HTTP Method	Resource	Comment
`assignUserToVO`	Post	VO	Assign user to VO
`changelocalACDPassword`	Post	Login	Change user local ACD password
`updateP12VOCert`	Post	VO	Update P12 certificate of VO
`updateRegisteredUserDetails`	Post	User	Update user details
`resetACDPassword`	Post	Login	Rest user ACD password
`RevokeUserRole`	Delete	ACD Credential	Revoke user role
`deactivateACDUser`	Delete	Login	Deactivate ACD user
`Deactiveteall`	Delete	User	Deactivate all
`removeUserVOs`	Post	VO	Remove user from VO

2.4.2 Deviations from Proposed Design

Several changes in AHE have occurred since the publication of Deliverable 2.2. First, JavaGAT will not be used in the Connector module as AHE does not currently require the features provided by JavaGAT or general-purpose grid SDK libraries. Instead, it is simpler to use UCC for Unicore, JGlobus for Globus and additional middleware Java client libraries as required.

Several components have been added as extensions to the presented design, particularly in the context of user credential management. This is due to the fact that AHE may have to be deployed without ACD. Proxy Delegation support has also been added to AHE, enabling AHE to automatically obtain proxy certificates on behalf of the user.

Another change relates to interoperation between AHE and GridSpace, which is no longer required.

In ACD, one major change is that AHE will no longer intercept commands from AHE and authorise them. Instead, all commands will be authorised by AHE, while ACD will be used to authenticate the user and – if necessary – generate proxy certificates.

2.4.3 Ongoing Work and Future Plans

Currently the most notable issue affecting AHE are Java dependency conflicts caused by the large number of external libraries used. One example of this is when different versions of the same library are found in multiple external packages. This may cause some packages to function incorrectly. One possible solution is to implement AHE in an OSGI framework, such as Eclipse Equinox or Apache Felix runtime.

The current development plan for AHE is to implement several use cases to ensure that AHE are able to run real-world scenarios. As ascertained in the course of discussions with WP5 representatives, this includes using GROMAC for HIV studies in ViroLab, and euHeart simulations.

AHE will also implement more complex workflow support, enabling error recovery, batch job submissions and application scheduling. Error recovery will allow users to reconfigure a virtual application when AHE has discovered an error during job submission: once an error is detected, AHE waits for the user to correct the error or terminate the application. Batch job workflows will allow users to run simulations which exploit the same application but with a different range of variables per submission. The job scheduler will allow users to run applications concurrently, creating a coupled simulation. We will also investigate more intuitive complex workflow management methods and integration with the workflow engine.

A number of other aspects of AHE will be subject to improvements. These include AHE response messages (more intuitive information passed from the underlying middleware to the user). Additionally, AHE will be integrated with external grid libraries, including Steering, SPRUCE emergency submission and advance reservation functions. We will also be adding more connectors for different middleware and data transfer components, such as the VPH-Share data transfer API (see Section 2.5 for details).

Currently, there are no major issues with ACD. However, we are investigating the implementation of OpenID in ACD for authentication as well as implementing database access using Hibernate ORM. This will allow ACD to be more easily deployed.

2.5 Data Access for Large Binary Objects

The Large OBject Cloud Data storagE fedeRation (LOBCDER) is a service that aims to provide reliable, managed access to large binary objects stored in various storage frameworks and providers. LOBCDER transparently integrates multiple autonomous storage resources, and exposes all available storage as a single name space. The LOBCDER service is divided into three main layers. Figure 8 shows a conceptual design of LOBCDER, which includes the frontend, resources and the backend.

The frontend's purpose is to provide access and control through standardised well known interfaces. This removes the need for developing and maintaining specialised clients. LOBCDER's frontend is a Web-based Distributed Authoring and Versioning (WebDAV) servlet, which presents the entire available storage space via HTTPs.

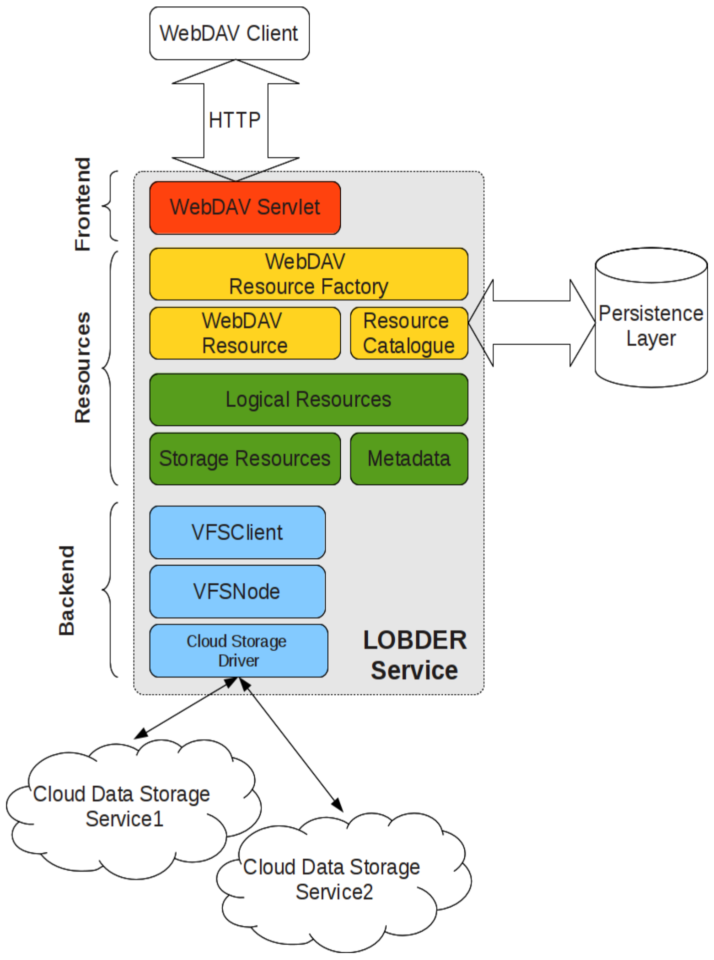


Figure 8: Conceptual design of LOBCDER.

The purpose of the resource layer is to create a logical representation of the physical storage space. The first part of the resource layer is the WebDAV resource and the resource catalogue. The WebDAV resource implements the WebDAV specifications while the catalogue is responsible for querying the persistence layer for logical resources. The logical resources are middle layers that bind the WebDAV resources with the physical data. This is achieved with the help of the storage resource, where is holds a representation of a physical storage location. Logical resources also hold metadata that contain content types, sizes, creation dates, permissions, etc. Such metadata is stored in AIR on behalf of LOBCDER.

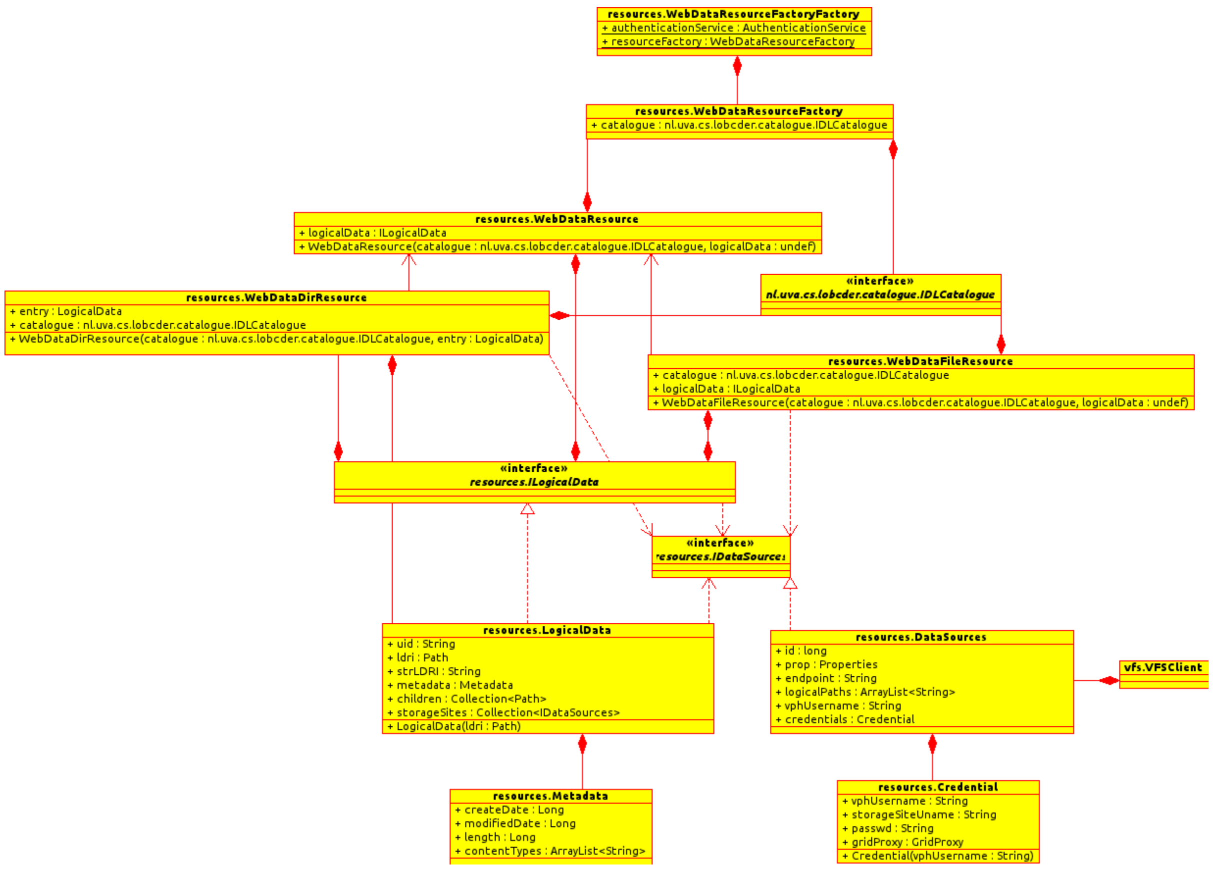
Finally, the backend layer provides the necessary abstraction for uniformly accessing physical storage resources. The main component is a Virtual Resource System Client which is able to access any physical resource system, thus providing a uniform API to the components above it.

2.5.1 Status of the Prototype

As described in deliverable D2.2 [2], the frontend would enable the following operations on the WebDAV resources:

- **Copy** creates a duplicate of a source dataset (identified by its Logical Data Resource Identifier or LDRI, which is an URI (Universal Resource Identifier) pointing to the dataset in LOBCDER), at a destination data resource.
- **Delete** removes a data resource.
- **Lock/Unlock** locks and unlocks access to a data resource while its owner is modifying it.
- **Make Collection** creates a new collection (or bucket) at the location of the specified LDRI.
- **Move** moves a data resource to the location specified by an LDRI.
- **Get Properties** retrieves properties for a data resource such as mime type, and length.
- **Upload/Download**

Currently LOBCDER is able to provide all of these operations, except the lock/unlock operation. More specifically, the current state of the LOBCDER prototype is depicted in the class diagram shown in Figure 9. Note that at this stage the LOBCDER prototype is not connected to the persistence layer, i.e. the Atmosphere Internal Registry (AIR). Instead, it runs its own persistence mechanism through a local database. At this point we will provide a description of the classes listed in Figure 9.


resources.WebDataResourceFactoryFactory
+ authenticationService : AuthenticationService
+ resourceFactory : WebDataResourceFactory
resources.WebDataResourceFactory
+ catalogue : nl.uva.cs.lobcder.catalogue.IDLCatalogue
resources.WebDataResource
+ logicalData : ILogicalData
+ WebDataResource(catalogue : nl.uva.cs.lobcder.catalogue.IDLCatalogue, logicalData : undef)
«interface»
nl.uva.cs.lobcder.catalogue.IDLCatalogue
resources.WebDataDirResource
+ entry : LogicalData
+ catalogue : nl.uva.cs.lobcder.catalogue.IDLCatalogue
+ WebDataDirResource(catalogue : nl.uva.cs.lobcder.catalogue.IDLCatalogue, entry : LogicalData)
resources.WebDataFileResource
+ catalogue : nl.uva.cs.lobcder.catalogue.IDLCatalogue
+ logicalData : ILogicalData
+ WebDataFileResource(catalogue : nl.uva.cs.lobcder.catalogue.IDLCatalogue, logicalData : undef)
«interface»
resources.ILogicalData
«interface»
resources.IDataSources
resources.LogicalData
+ uid : String
+ ldri : Path
+ strLDRI : String
+ metadata : Metadata
+ children : Collection<Path>
+ storageSites : Collection<IDataSources>
+ LogicalData(ldri : Path)
resources.DataSources
+ id : long
+ prop : Properties
+ endpoint : String
+ logicalPaths : ArrayList<String>
+ vphUsername : String
+ credentials : Credential
vfs.VFSClient
resources.Metadata
+ createDate : Long
+ modifiedDate : Long
+ length : Long
+ contentTypes : ArrayList<String>
resources.Credential
+ vphUsername : String
+ storageSiteUname : String
+ passwd : String
+ gridProxy : GridProxy
+ Credential(vphUsername : String)
Figure 9: LOBCDER class diagram.

WebDataResourceFactoryFactory: As the name suggests, this class is responsible for creating WebDataResourceFactory instances. The **WebDataResourceFactoryFactory** class is instantiated when a new request comes from client through the **WebDavServlet**.

Moreover, this class is meant for configuring and instantiating the **WebDataResourceFactory** class.

WebDataResourceFactory: This class's role is to create **WebDataResource** resources, and return them to the **WebDavServlet**. Thus, the main method of this class is **getResource**. This method locates an instance of a **WebDataResource** for a given URL (Universal Resource Location). Additionally, this class creates the **IDLCatalogue** instances.

IDLCatalogue: This class is created when the **WebDataResourceFactory** receives a request. The role of this class is to query the persistence layer for **LogicalData** and **DataSource** entries. Two of the most frequently used methods of this class are **registerResourceEntry** and **getResourceEntryByLDRI**. **registerResourceEntry** will register a **LogicalData** instance in the persistence layer only if there is no duplicate entry present. Otherwise this method will throw an exception. **getResourceEntryByLDRI** is quite straightforward: given a Logical Data Resource Identifier (LDRI), the method should return a single one LogicalData entry.

WebDataResource: This is a superclass for all the WebDAV resources (file and folder). The methods not currently implemented in this class are **authenticate**, **authorise** and **checkRedirect**. WebDAV provides the ability to authenticate and authorise each resource separately, providing better granularity for user permissions. These two methods will be implemented in order to reflect the permissions each member of the VPH-Share community has with respect to data resources. Additionally, the **checkRedirect** method will be implemented at a later stage, when LOBCDER is deployed on multiple hosts. Under this setup LOBCDER will be able to redirect incoming calls to the nearest LOBCDER instance.

WebDataDirResource/WebDataFileResource: These two classes are subclasses of **WebDataResource**, and provide a representation of the logical and physical data to a WebDAV client.

LogicalData: This class encapsulates the logical entries held on the persistence layer together with the physical data stored on the cloud services. Hence, this class holds a LDRI, which uniquely identifies the logical resource, and a set of **DataSources** that hold replicas of the physical data. When a new **LogicalData** object is created, it is not necessary to have any physical data associated with it either because the client may create an empty file, or because this instance represents a folder or a collection. Moreover, this class holds a **Metadata** member that provides creation and modification dates as well as mimetypes.

DataSource: This class is a representation of the storage resource that holds the actual data. Since there are many different storage resources this class needs to have a **Credential** member that will provide access to storage resources. This class can hold credentials such as passwords, certificates, etc.

VFSClient: In order to be able to interact with the physical storage resources, each **DataSource** uses a **VFSClient**. This class provides numerous methods for manipulating and managing data on the remote storage resource.

To obtain a better understanding of how LOBCDER works, the specific sequence of operations involved in processing user requests is presented below.

1. A WebDAV client sends a request to the **WebDavServlet**.
2. The **WebDavServlet** calls the **WebDataResourceFactoryFactory** to create a **WebDataResourceFactory** instance.
3. A **WebDataResourceFactory** instance is returned to the **WebDataResourceFactoryFactory**.
4. The **WebDataResourceFactory** class creates an **IDLCatalogue** in order to be able to query the requested resource.
5. The **WebDavServlet** will now call the **getResource** method from the **WebDataResourceFactory** class.
6. The **WebDataResourceFactory** will use its **IDLCatalogue** to query the requested resource from the persistence layer.
7. The returned entry is an instance of **LogicalData** class which contains a **DataSource** that indicate the physical location of the requested resource.
8. At this moment the **WebDataResourceFactory** will get the **DataSource** from the **IDLCatalogue** the requested user has access to.
9. The **WebDataResourceFactory** will instance the **WebDataResource**, and set its DataSource.
10. Finally, the **WebDataResourceFactory** will return the requested **WebDataResource** back to the **WebDavServlet**, where it will respond to the WebDAV client.

2.5.2 Ongoing and Future Work

2.5.2.1 Performance

Since the design goal for LOBCDER is to federate data located in the cloud, we will optimise our implementation to handle extensive datasets before we start developing distributed solutions in order to address scale issues. We will also investigate possible optimisation techniques such as asynchronous requests, parallel streams and caching, to better alleviate the effect of backend drivers on LOBCDER performance.

2.5.2.2 Authentication

In parallel with this effort we are developing a security mechanism that protects LOBCDER resource handles against unauthorised access. This implies that for each request, we need to know who is accessing a given resource (credentials) and make a decision on whether the requestor has sufficient rights (permissions). For a typical set of credentials consisting of a username and a password, the HTTPS authorisation a scenario is depicted in Figure 10.

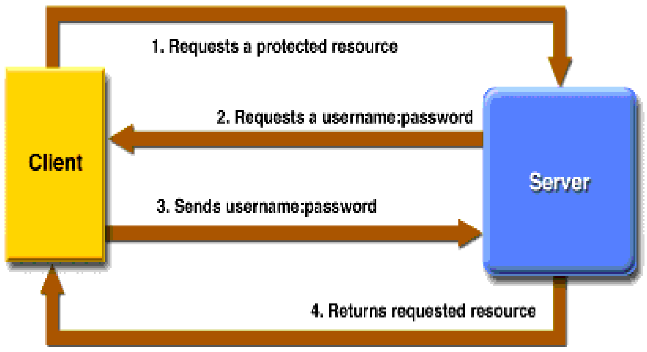

1. Requests a protected resource
2. Requests a username:password
3. Sends username:password
4. Returns requested resource
Client
Server

Figure 10: HTTP basic authentication.

Each LOBCDER user is authenticated by their VPH username. We also assume that an Authentication Service (*AS*) is available where the user is authenticated prior to accessing LOBCDER. As a result of this authentication, the user receives a security token. Later this token is included in the password field whenever LOBCDER needs to authenticate the user. If the user is not considered to be authenticated, LOBCDER calls *AS* passing this token as a parameter. If the token is valid (i.e. was issued to the user by *AS* during the authentication procedure and has not expired) then *AS* returns a positive answer and the list of roles assigned to the user. This list is placed in the request context for later authorisation.

2.5.2.3 Authorisation

When an authenticated user tries to access a LOBCDER resource, the system has to decide if this user is authorised. Any logical path in LOBCDER (**LogicalData** resource) shall have an Access Control List (ACL) attached to its metadata. ACL consists of the list of the roles associated with a resource and permissions set by the resource owner for these roles. There are two pseudo roles called "owner" and "others" respectively. If an authenticated user is the owner of a resource (the resource was created by him), he is automatically assigned this role. Any authenticated user is a member of the "others" pseudo-role (Figure 11). If the user has more than one role, the least restrictive role is used for ACL permissions. To manipulate the content of ACL, LOBCDER may require an additional extension, going beyond its data access (WebDAV) interface. This issue will be investigated further.

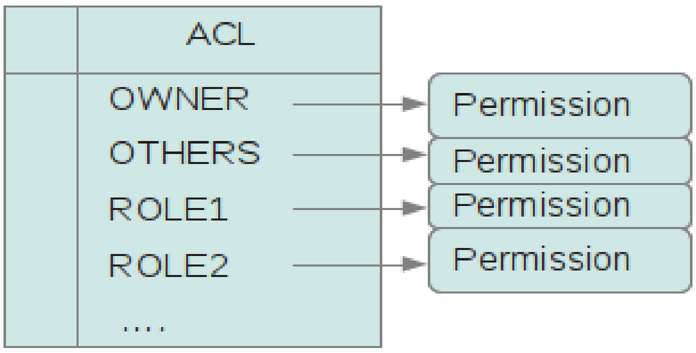


Figure 11: LOBCDER access control list.

2.6 Data Reliability and Integrity

As presented in Deliverable 2.2 [2], the goal of the Data reliability and integrity (DRI) tool is to ensure reliable use of the biomedical datasets manipulated with the use of VPH-Share applications and tools. Simulations results and inferred medical outcomes must be based on reliable data. Due to the large size and long-term persistence of medical data files, special reliability and integrity mechanisms should be enforced on top of Cloud storage. Thus, the infrastructure developed in Task 2.5 needs to be able to perform the following tasks:

- periodic integrity checks on data objects with the use of hash algorithms,
- facilitating storage of multiple copies of data on various Cloud platforms,
- tracking the history and origin of binary datasets.

2.6.1 Status of the Prototype

The DRI Runtime is responsible for enforcing Task 2.5 data management policies. It keeps track of managed components and periodically verifies the accessibility and integrity of the managed data. As designed, the Runtime assumes the form of a generic (i.e. non-application-specific) Atomic Service in the WP2 infrastructure. Thus, it can be managed and deployed by Atmosphere tools, just like any other type of Atomic Service. For scalability purposes, multiple instances of DRI Runtime may coexist in the system, integrated into a coherent platform by sharing a common registry (the WP2 persistence layer), although at present only a single instance of the prototype service has been deployed on the computing resources contributed by CYFRONET (see Section 2.3 for details).

In line with the Atomic Service specification, the DRI service has been deployed into a virtual machine and further registered with Atmosphere mechanisms for automatic management. It is currently able to monitor and validate the data sets registered with the Atmosphere Internal Registry and present in the Swift data storage that is part of the VPH-Share cloud federation. Figure 12 presents the architecture of DRI Runtime.

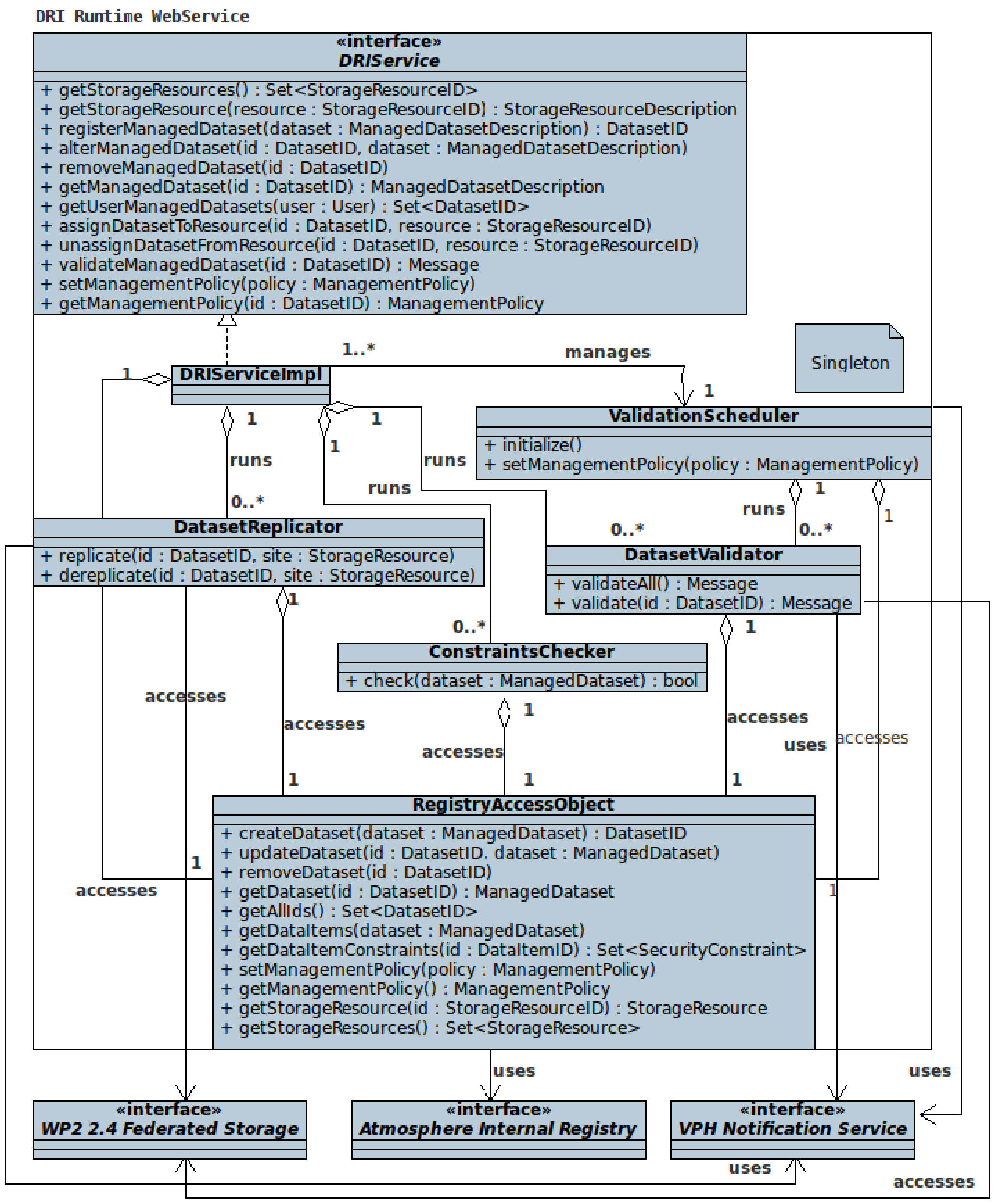

DRI Runtime WebService
«interface»
DRIService
+ getStorageResources() : Set<StorageResourceID>
+ getStorageResource(resource : StorageResourceID) : StorageResourceDescription
+ registerManagedDataset(dataset : ManagedDatasetDescription) : DatasetID
+ alterManagedDataset(id : DatasetID, dataset : ManagedDatasetDescription)
+ removeManagedDataset(id : DatasetID)
+ getManagedDataset(id : DatasetID) : ManagedDatasetDescription
+ getUserManagedDatasets(user : User) : Set<DatasetID>
+ assignDatasetToResource(id : DatasetID, resource : StorageResourceID)
+ unassignDatasetFromResource(id : DatasetID, resource : StorageResourceID)
+ validateManagedDataset(id : DatasetID) : Message
+ setManagementPolicy(policy : ManagementPolicy)
+ getManagementPolicy(id : DatasetID) : ManagementPolicy
1..*
manages
Singleton
1
DRIServiceImpl
1
1
1
1
runs
runs
runs
runs
0..*
0..*
0..*
ValidationScheduler
+ initialize()
+ setManagementPolicy(policy : ManagementPolicy)
1
1
DatasetReplicator
+ replicate(id : DatasetID, site : StorageResource)
+ dereplicate(id : DatasetID, site : StorageResource)
DatasetValidator
+ validateAll() : Message
+ validate(id : DatasetID) : Message
1
0..*
1
ConstraintsChecker
+ check(dataset : ManagedDataset) : bool
1
accesses
accesses
accesses
accesses
uses
accesses
1
1
1
RegistryAccessObject
+ createDataset(dataset : ManagedDataset) : DatasetID
+ updateDataset(id : DatasetID, dataset : ManagedDataset)
+ removeDataset(id : DatasetID)
+ getDataset(id : DatasetID) : ManagedDataset
+ getAllIds() : Set<DatasetID>
+ getDataItems(dataset : ManagedDataset)
+ getDataItemConstraints(id : DataItemID) : Set<SecurityConstraint>
+ setManagementPolicy(policy : ManagementPolicy)
+ getManagementPolicy() : ManagementPolicy
+ getStorageResource(id : StorageResourceID) : StorageResource
+ getStorageResources() : Set<StorageResource>
1
accesses
1
uses
uses
«interface»
WP2 2.4 Federated Storage
«interface»
Atmosphere Internal Registry
«interface»
VPH Notification Service
uses
accesses

Figure 12: DRI Runtime architecture.

At the core of the prototype lies the **DatasetValidator** class, which performs periodic validation of data items represented in the Atmosphere Internal Registry. This class is configurable by a dedicated set of parameters stored in the Registry. As part of our ongoing development work, we are implementing an end-user interface that will enable administrators to manage the runtime parameters of the DRI service. Furthermore, the DRI prototype provides a service frontend that can be used to register, unregister and query the status of managed datasets.

Figure 13 presents the persistence schema upon which DRI Runtime bases its operation. This schema is managed by the Atmosphere Internal Registry which thus provides a uniform persistence layer for DRI (as well as for other core components of WP2).

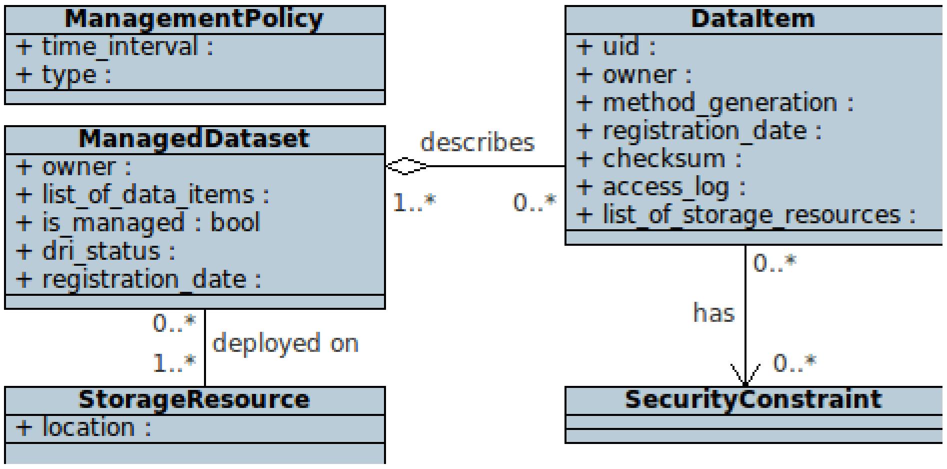


Figure 13: DRI data model.

DataItem is a virtual concept shared between DRI and LOBCDER. It can describe either a single file or a collection of files. A **DataItem** is optionally associated with a **ManagedDataset** object, which implies that it needs to be validated by DRI, in accordance with the parameters stored within that instance of **ManagedDataset**. **ManagementPolicy** specifies the general operating characteristics of DRI and can be configured by system administrators.

The DRI Runtime prototype is developed in Java. The service is deployed to a an application server residing on a virtual machine currently provided by CYFRONET and exposes a RESTful Web Service API (Jersey implementation) backed by the DRI component internally. Quartz Scheduler [12] is utilised for scheduling validation routines periodically.

2.6.2 Ongoing and Future Work

Having deployed the first prototype of DRI our focus will now shift from low-level conceptual and implementation-oriented details to providing a more robust set of interfaces and measuring the performance of various data validation algorithms. We intend to deploy a notification service (in collaboration with WP6) where any emerging problems could be communicated to system administrator and resource owners, either synchronously (e.g. via e-mail) or through notifications stored in AIR and displayed in the Master Interface whenever an authorised user has logged in.

The service is also in the process of being secured with authenticity tokens provided by task T2.6. When deployed in production mode, it will contact the common Policy Decision Point to guard against unauthorised access.

2.7 Security Framework

The security framework is responsible to ensure that all Atomic Services in the system are properly secured. To do so, any communications with these services are properly encrypted and have their authorisation checked before actually invoking any of the services.

2.7.1 Security requirements

2.7.1.1 Scope

In order to explain the scope of the requirements of the security framework, it is important to remember that VPH-Share is a platform that integrates a dynamic set of heterogeneous applications, each of them with its own security requirements. As such, the security framework does not aim to address all the security requirements that are specific to each integrated application, especially when new applications with new requirements can be added dynamically.

Hence VPH-Share, and more specifically – its security framework, will deal with the security issues resulting from the integration of these applications into a common and publicly accessible framework, but the intrinsic security requirements which are specific to each application will remain as the responsibility of that application and associated application developer.

VPH-Share will provide a generic mechanism to enable application developers to define the security constraints for the accessibility of each application based on information relative to the user, which in turn will be stored and retrieved from a common authentication platform.

2.7.1.2 Authentication

One of the goals of VPH-Share is to help augment the VPH initiative by providing a reliable and consolidated infostructure. To help foster and cement trust relationships with existing VPH users, the system will try to reuse the same authentication platform used for the BiomedTown portal. This will provide a single sign-on mechanism for the users of both BiomedTown and VPH-Share applications.

2.7.1.3 Security perimeter

As explained in Section 2.7.1.1, the security framework will deal with the security issues resulting from the integration of applications. This, in practice, means that the security framework will protect the integrated platforms from unauthorised access from outside the VPH-Share platform.

2.7.1.4 Atomic services

VPH-Share applications, also referred to as Atomic Services in this document, will be supported with:

- **Privacy assurance** by encrypting all incoming and outgoing communication to/from the VPH-Share platform.
- **Specific authentication** by integrating user attributes with a common identity provider (BiomedTown)
- **Authorisation support**:
  - Basic role-based authorisation by checking role attributes stored by the common shared identity provider
  - Customisable policy system able to define complex access rules based on user attributes stored by the common identity provider.

The security framework aims to transparently incorporate these features as a wrapper around the Atomic Services. Note, however, that both the security roles and the access rules are specific to each Atomic Service and hence it is the responsibility of the application developers to define such rules and the user attributes needed to validate these rules. The task of surveying and the definition of such rules will be performed during the second year of the Project.

2.7.1.5 LOBCDER / WebDAV

The LOBCDER service provides a standardised WebDAV interface to unified distributed file system and with respect to security, it will at least provide:

- **Privacy assurance** by encrypting all incoming and outgoing communication to/from the platform;
- **Basic user/password authentication** – LOBCDER cannot reuse any of the complex authorisation features provided by the Atomic Services because the WebDAV standard does not support them. Thus, LOBCDER will alternatively use basic user/password mechanisms to ensure compatibility with the WebDAV standard and third-party WebDAV clients.

Although LOBCDER authentication requirements differ slightly from the approach suggested for Atomic Services, a common model can be applied by taking these inconsistencies into account.

2.7.1.6 Workflow Management / Taverna

The Taverna workflow system will have the same security requirements as the Atomic Services, with the only addition that applications may be launched from a command line instead of a web client. It is acceptable for the authentication process to be performed from the command line, by having the Taverna workflow system supply authentication parameters to the identity platform through a secure API.

2.7.1.7 Performance

The security layer should not introduce significant overhead within the system or, at least, it should not be larger than its other working components within ASI. The initial goal for the security proxy execution time is set on 1 second, subject to server workload and the impact of other system components.

2.7.1.8 External services

VPH-Share includes a search engine to look for additional external services to be used conjunctly with VPH-Share Application Services. However, VPH-Share has no way to guarantee the security of these external services at all. Therefore, VPH-Share will follow a classic trust/deny mechanism, meaning that external services will be initially labelled as untrusted, and access to them banned from the firewall configuration of the system. However, the system will also maintain a list of trusted external services. Hence, the correct procedure for accessing these services will be to ask the system administrator for access to a given service. An administrator will be responsible for granting such access (or denying it) and configuring the firewall appropriately.

2.7.1.9 Security policies

As explained above, the system will provide the ability to define access criteria for services based on the attributes assigned to the user. These policies can be as simple as a role-based mechanism or as complex as a combination of policy rules depending on more complex attributes.

An example of a simple role-based policy could be: "**If** the user **has the role** *paediatrics* assigned, **then** they should be granted access to the Paediatrics database service".

An example of a complex policy rule could be: "**If** the user **has at least** a bachelor's degree in medicine **and has** signed a confidence agreement **then** grant access to all medical stores".

Security policies can be a very powerful mechanism for developers to express accessibility constraints via rules for services. However, there is no way that the VPH-Share platform can know them beforehand, as they are specific to each service. As a result, it will be the responsibility of application developers to infer these criteria from the application requirements and user attributes provided by the Master Interface. These security policies will be defined and implemented during the second year of the Project.

The administration of the security system will have to be integrated with the Atmosphere Internal Registry via the Master Interface, allowing application developers to update the policy rules associated with their services (even already instantiated ones). It will therefore be possible to maintain and update the access criteria for services without the need to define a new service template in each case.

2.7.2 Security for Atomic Service invocations: the Security Proxy

The Security Proxy, which is preinstalled on any Virtual Machine which hosts an Atomic Service works jointly with the Reverse Proxy, which is in charge of forwarding all incoming requests from a public IP address to private addresses and balancing the workload of the system. Note, that, as mentioned in Section 2.3.1, the actual Atomic Service Instances sit behind an IP forwarding Reverse Proxy so that the ASIs do not expose their private IP addresses.

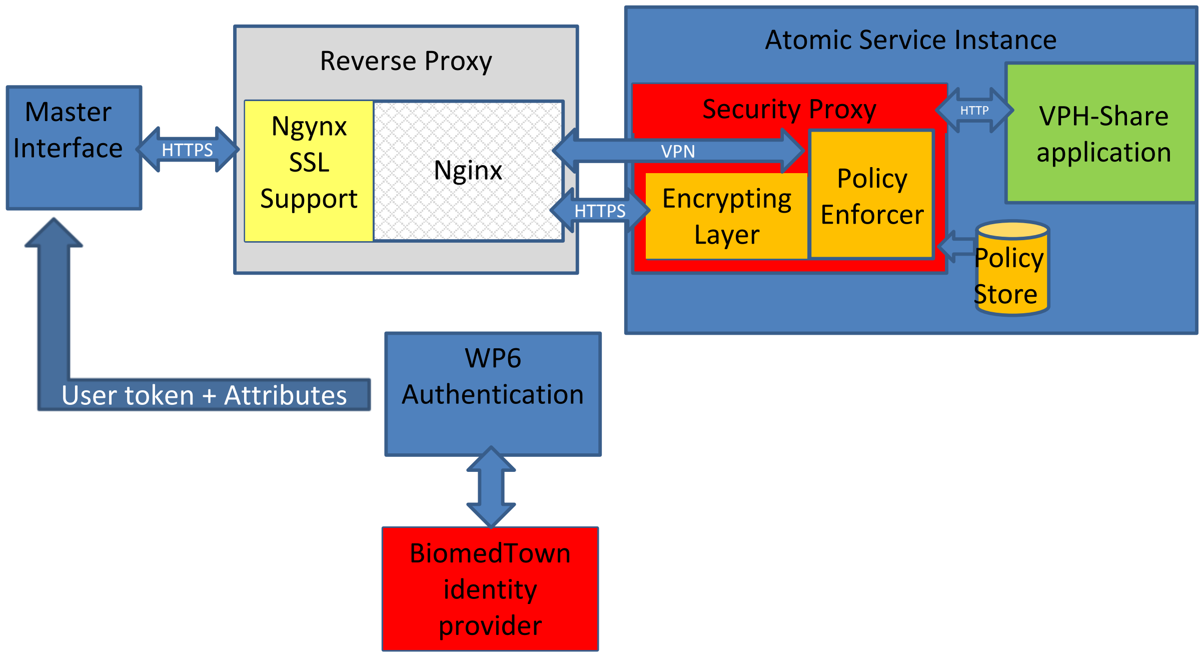


Figure 14: Overview of WP2 security mechanisms.

Figure 14 presents a schematic view of how the Security Proxy integrates with the Reverse Proxy, external clients, Atomic Services and WP6.

As the first step, the user authenticates via the Master Interface, which performs the necessary actions, including forwarding the authentication request to the BiomedTown identity provider and retrieving all user attributes. Subsequently, this authentication information (user + attributes) is signed by the Master Interface and included in the password field of the header of the HTTP request, which will travel through the whole invocation.

To further secure data transmission, all communication between the different servers is encrypted at the HTTP level by using the SSL (Secure Socket Layer) protocol over HTTP (HTTPS). This implies that the Reverse Proxy itself decrypts the message, performs any calculations required for message forwarding, and encrypts it again prior to contacting the service recipient. Dealing with SSL can be easily performed on the Nginx server by configuring the nginx HTTP SSL module (see the nginx HTTPS Module [13] for details).

On the Atomic Service Instance side, all HTTP requests are intercepted by the Security Proxy, which listens to incoming requests on configurable ports and performs two tasks: decrypting messages and checking for authorisation for a given service based on the attributes of the user provided along with the request into the VPH-Share platform. It is capable of extracting user attributes from the HTTP header and confirming their integrity by checking the signature of the field containing it. These user attributes are then used jointly with the security policies stored in the policy store to decide whether to grant or deny access to the service.

If the request is granted, the invocation is forwarded to the service on a local host address. Therefore, all services (including the Apache server) are required to communicate through the local host. An advantage of this design is that it also allows different services to invoke each other on different networks, by passing through the Nginx server responsible for properly forwarding the request. Figure 15 shows how such communication can be accomplished.

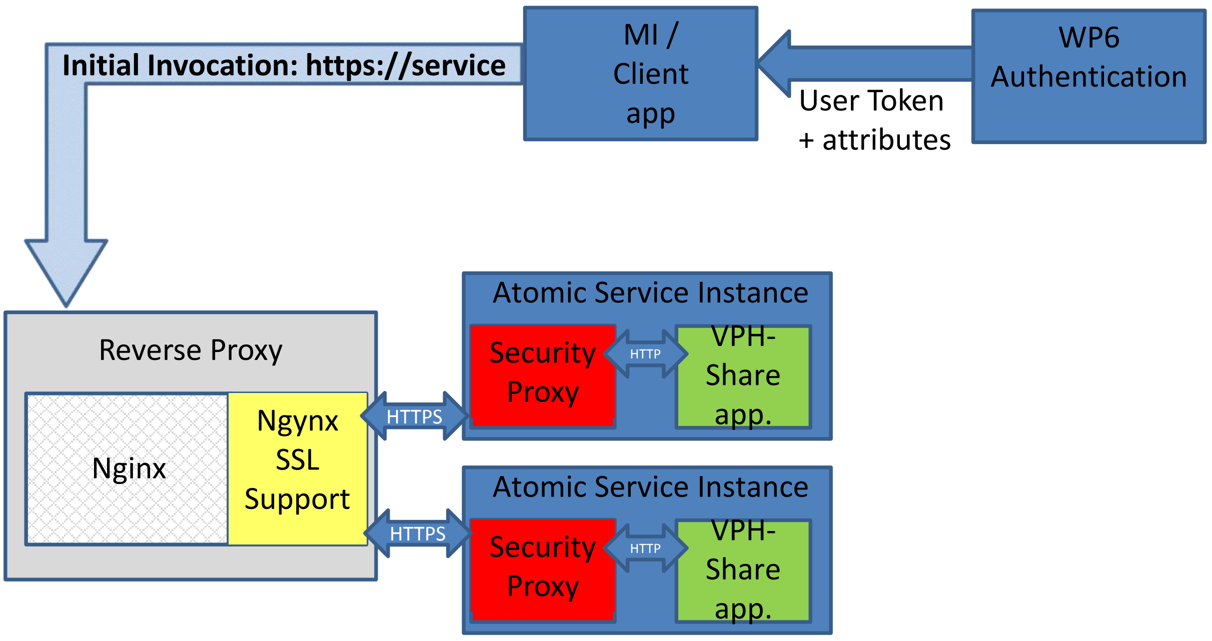


Figure 15: Inter-service communication with Nginx.

The sequence of steps a request takes within the VPH-Share platform is shown in Figure 15. The initial request to invoke any Atomic Service will first be authenticated in Work package 6, which will properly include the user token and attributes. This user token and attributes then need to travel along with the request to the Security Proxy in the HTTP header, and will reach the corresponding VPH-Share application service. When the VPH-Share application wishes to invoke another Atomic Service, it will invoke its corresponding public IP address and follow the same path again without the WP6 authentication, with the original user token travelling through all the invocations.

The presented architecture allows any HTTP-based services, including REST and SOAP services to be secured in a transparent manner. Regarding the Reverse Proxy, the design facilitates the required encryption by simply adapting the configuration files of the Nginx server.

Regarding the security policies, they will be stored locally in the Atomic Service template, which implies that its configuration will be copied to all instances of the Atomic Service upon deployment. However, Atomic Service policies can be modified either when defining the template or after it has been instantiated through the Atmosphere Internal Registry management console. To do so, each Atomic Service Instance will have a security agent installed on it, which will securely accept configuration files from the management console and perform basic syntactic validation prior to accepting them. This allows the application developer to update policy rules and define access criteria for the security proxy on a given machine. Figure 16 presents this process.

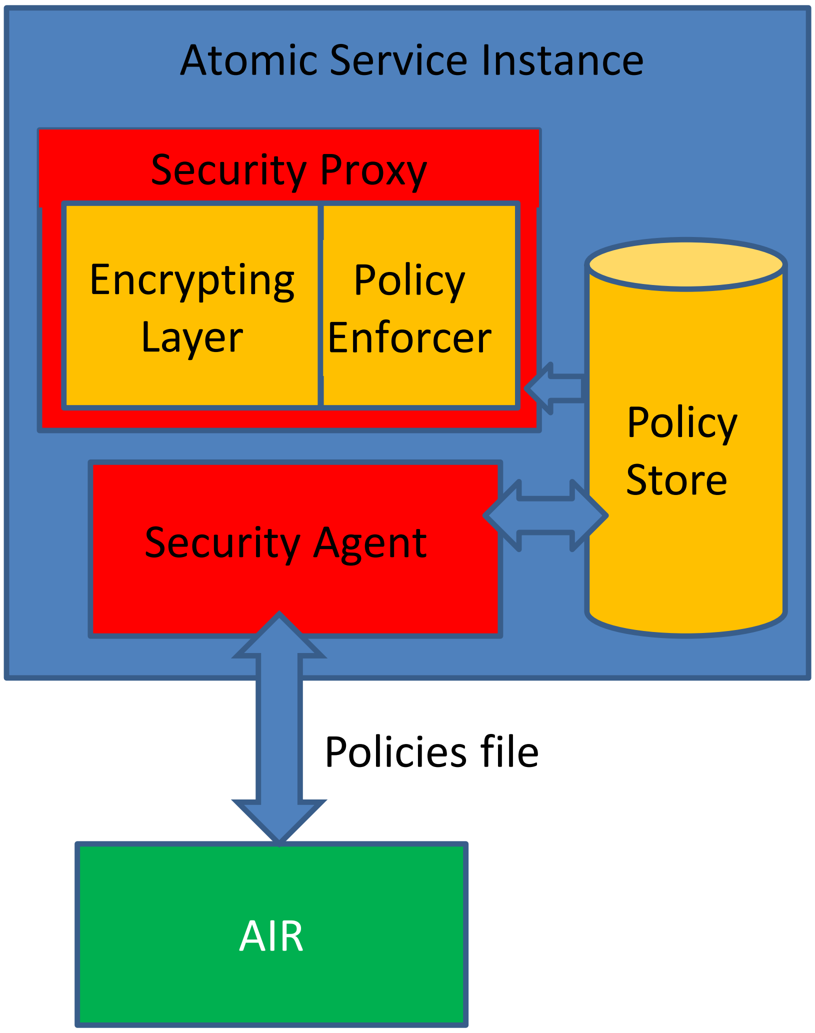


Figure 16: Updating policy rules of the Security Proxy.

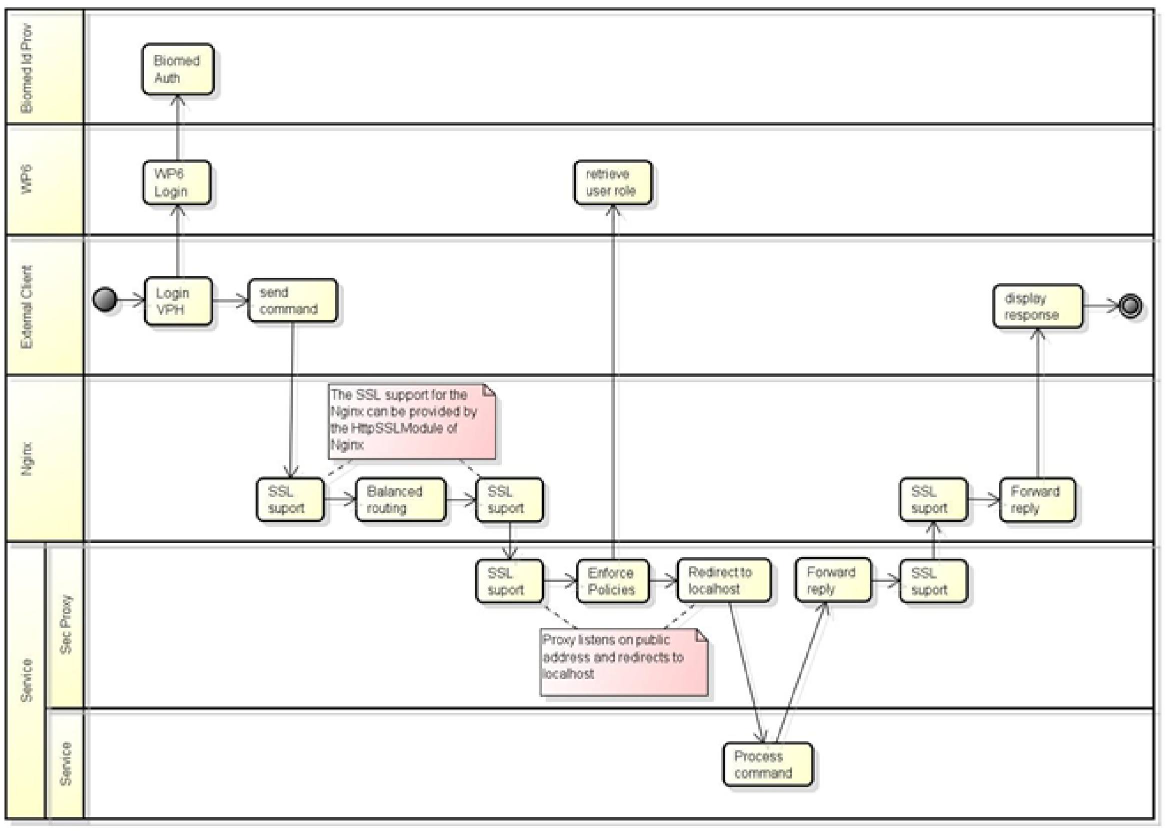


Figure 17: User authentication and authorisation in the context of invoking Atomic Services deployed in a private network.

Figure 17 shows the sequence diagram involved in communication with an Atomic Service, for an arbitrary process launched in any external client application (or within the context of the Master Interface).

The process starts with the external client (or the service which launches a given application in a terminal session on the service host). The client logs into VPH-Share and subsequently issues a service invocation request to a public IP address, which is properly encrypted with SSL. The Nginx proxy decrypts the request, selects the appropriate private IP of the Atomic Service Instance and forwards the request, having again encrypted it with SSL.

Once the request arrives at the local (instance-bound) Security Proxy, it is decrypted, and its security attributes used to decide whether the user is authorised to perform the given operation. If so, the HTTP request is redirected to a local host address of the Atomic Service. Once the service produces a result, it is properly encrypted and sent back to the reverse proxy, which can finally deliver it to the client application.

2.7.3 Status of the prototype and ongoing work

The initial prototype of the security framework includes the Security Proxy, which has been deployed on the first Atomic Services produced by WP5 and can be used to contact said services. Thus, service communication proceeds in a secure manner and the Security Proxy can be replicated to additional Atomic Services (by inclusion in the host VM).

There is ongoing work on integration of the WP2 security mechanisms with the user identity and credential management tools provided by WP6, which will be delivered in its first prototype (due by Month 15).

During the next 12 months, work will focus on:

- Negotiation and definition of an initial set of policy rules (in collaboration with application developers) to be used in the deployment of the first Atomic Services
- Detailed design of the policy engine
- Initial implementation of the policy engine
- Study on collaboration with p-medicine

3 INTEGRATION WITH EXTERNAL WORK PACKAGES

This section presents the core issues pertaining to integration with external WPs, especially in the context of user interfaces and deployment of application-specific Web Services.

3.1 WP2 End-user Interfaces

3.1.1 General description of WP2 user interfaces

WP2 user interfaces will consist of two main parts. The Cloud Management Portlet handles the lifecycle of Atomic Services which includes instantiating basic templates from which Atomic Services can be created, listing the active Atomic Services which can be accessed by the current user and invoking their operations. The second part – Data Management Portlet – provides views for managing data sources and files in the VPH-Share federated storage infrastructure. As schematically depicted in Figure 18, both components are part of the Master Interface (developed by WP6) and share a common dependency upon the Cloud Facade module. This is dictated by the requirement for a uniform interface to the cloud infrastructure, enabling different clients (not just a single web application) to obtain access to VPH-Share resources by interfacing directly by the core services exposed by WP2.

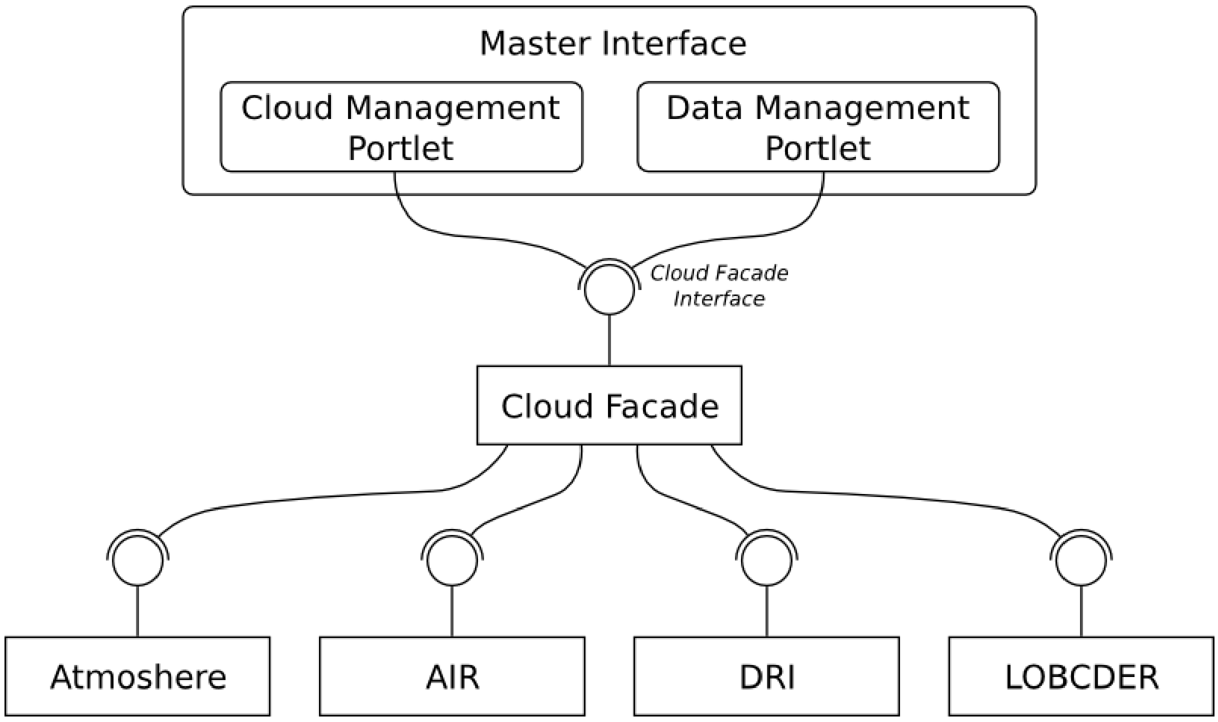


Figure 18: Architecture diagram of WP2 user interfaces and dependencies.

The Cloud Facade exposes its interface using well-known standards such as WS (Web Services) and REST, which are utilised by WP2-specific portlets and can be conveniently adapted by other clients. The underlying cloud and data components include the following:

- Atmosphere – manages the lifecycle of virtual machines (VMs) and Atomic Services (ASs), provides methods for starting, stopping or pausing VMs and AS Instances,
- AIR – a common registry which stores information about the state of Atomic Services and data sources. This is the internal WP2 registry referenced in Figure 1.
- DRI – a standalone service which ensures data integrity and reliability, configured by VPH administrators,
- LOBCDER – VPH-Share data federation facade, used for file payload transfers.

Operations invoked on the components listed above are synchronised by the cloud facade which delivers a coherent and convenient way of accessing the entire VPH-Share cloud infrastructure.

3.1.2 Ongoing development

The Portlet specification (JSR-286) was chosen as a base for implementation of WP2 user interfaces. The Jetspeed-2 Portal [14], with its flexible customisation mechanisms, was approved as the portlet container used to host WP2 portlets (individual user interface views). Currently, the flow of the Cloud Management Portlet is fully implemented on top of mock data as the Cloud Facade dependencies are not yet operational. The flow includes listing user's active Atomic Services, instantiating VM template images and Atomic Service Instances, registration of new Atomic Services and presenting credential information that allows users to log into the virtual machines.

Integration with the Master Interface (MI) is handled by embedding individual portlet views via mash-up techniques. To keep the overall look-and-feel consistent, separate CSS (Cascading Style Sheets) may be applied to pages integrated within the MI. For development purposes the portal may also act as a standalone web application, with its own layout, at least until integration with the Master Interface has been concluded.

In terms of deployment, installation is divided between two servers: the portlet container (responsible for serving the user interface views) and the Cloud Facade service (deployed inside an OSGI container [15] as a SOAP or REST web service). Such separation facilitates integration with dependent WP2 components and makes the facade endpoint operational and accessible by different clients independently of the portal.

3.2 Status VPH-Share Atomic Services

This section briefly describes the status of Atomic Services, which have been made available for testing as of the publication date of this deliverable. Note that these are application components correctly deployed and tagged as Atomic Services – additional components are in the pipeline, but until they have been registered with WP2, Atmosphere will be incapable of managing their status.

Details of each of the Atomic Services currently known to WP2 are listed in Table 6. Please note that as, in many cases, the services themselves remain under development by the respective workflow teams, WP2 maintains a set of "persistent" instances that are not shut down whenever a given service is not performing calculations. The purpose of this temporary arrangement is to facilitate development and testing of Atomic Services by their creators and maintainers.

Table 6: List of Atomic Services provided to and registered by WP2 by Project Month 12.

WP5 Workflow	**Atomic Service Name**	**Deployment status**
EUHeart	VTK 2 Ex Format Convertor	Up and running on http://149.156.10.131:55000/vtk2ex/?wsdl
EUHeart	Meshing Service (Ongoing)	Up and running on http://149.156.10.131:55000/ex2vtk/?wsdl
EUHeart	Meshing Tools	Up and running on http://149.156.10.131:55000/heartgen/?wsdl
@neurist	Meshing Service (May be split into separate tools)	http://149.156.10.131:33502/axis2/services/wsGimias?wsdl (VNC interaction required), http://149.156.10.131:38529/axis2/services/wsGimias?wsdl
@neurist	Image Segmentation Service	http://149.156.10.131:33502/axis2/services/wsGimias?wsdl (VNC interaction required), http://149.156.10.131:38529/axis2/services/wsGimias?wsdl

4 SUMMARY

The features chosen for implementation within the initial development period enable us to deploy a coherent prototype, which can then be used to deploy and run sample applications, based on the services provided by WP5. While out of scope of this document, it should be noted that WP2 representatives continue to liaise with the WP5 application teams in order to help them extract the core functionality of their application workflows and deploy such functionality in the form of Atomic Services, in line with the requirements imposed by the Project architecture (and specifically by WP2). During the Consortium Meeting held in Barcelona in September 2011 we jointly prepared a list of prospective Atomic Services and data sets that would be contributed to the Project by each workflow team. Implementation and integration of these Atomic Services is ongoing and the results can be directly used to showcase the features of WP2 tools. As time goes by, we expect to be able to present additional workflow scenarios, acting upon real-life datasets and accessed with the use of user interfaces provided by WP6.

5 References

1. VPH-Share Project Consortium. Grant Agreement for Collaborative Project "VPH-Share", Annex I: Description of Work, 2nd revision. 2011..

2. VPH-Share Work Package 2. Deliverable D2.2: Design of the Cloud Platform. Internal Project Deliverable. ACC CYFRONET AGH; 2011.

3. VPH-Share Project Consortium. Deliverable D2.1: Analysis of the State of the Art and Work Package Definition. Internal Project Deliverable. ACC CYFRONET AGH; 2011.

4. OSGi Alliance Homepage. [Online].; 2011. Available from: http://www.osgi.org/Main/HomePage.

5. Apache Karaf OSGi Container. [Online].; 2011. Available from: http://karaf.apache.org/.

6. Apache Camel Integration Framework. [Online].; 2011. Available from: http://camel.apache.org/.

7. JClouds. [Online].; 2011. Available from: http://code.google.com/p/jclouds/.

8. Sloot PMA, Gubała T, Bubak M. Semantic Integration of Collaborative Research Environments. Handbook of Research on Computational Grid Technologies for Life Sciences, Biomedicine and Healthcare. 2009: p. 514-530.

9. OpenStack Project Home. [Online]. Available from: http://openstack.org/.

10. Nginx Project Home. [Online]. Available from: http://wiki.nginx.org/Main.

11. IPv4 Exhaustion. [Online]. Available from: http://www.ripe.net/internet-coordination/ipv4-exhaustion.

12. Quartz Scheduler homepage. [Online].; 2012 [cited 2012 2 29. Available from: http://quartz-scheduler.org/.

13. Nginx HTTPS Module. [Online]. Available from: http://wiki.nginx.org/HttpSslModule.

14. Jetspeed 2 Web Portal. [Online].; 2012. Available from: http://portals.apache.org/jetspeed-2.

15. Alliance O. OSGI Web Portal. [Online]. Available from: http://www.osgi.org/Main/HomePage.

16. Davidoff F, Godlee F, Hoey J, Glass R, Overbeke J, Utiger R, et al. Uniform requirements for manuscripts submitted to biomedical journals. JAOA: Journal of the American Osteopathic Association. 2003; 103(3).

LIST OF KEY WORDS/ABBREVIATIONS

ACD	Audited Credential Delegation
ACL	Access Control List
AHE	Application Hosting Environment
AIR	Atmosphere Internal Registry
AMS	Atmosphere Management Service
AS	Atomic Service
AS	Authentication Service
ASI	Atomic Service Instance
API	Application Programming Interface
CC	Cloud Controller
CEE	Cloud Execution Environment
DNAT	Destination Network Address Translation
DRI	Data Reliability and Integrity
EC2	Amazon Elastic Computing Cloud v2
HPC	High Performance Computing
HTTP	HyperText Transfer Protocol
LDRI	Logical Data Resource Identifier
LOBCDER	Large OBject Cloud Data storagE fedeRation
MI	Master Interface
NAT	Network Address Translation
ORM	Object-Relational Mapping
OSGI	Open Service Gateway Initiative
REST	REpresentational State Transfer

SaaS	Software as a Service
SDK	Software Development Kit
SOAP	Simple Object Access Protocol
SSH	Secure SHell
SSL	Secure Socket Layer
SSO	Single Sign-On
TLS	Transport-Layer Security
URI	Universal Resource Identifier
URL	Universal Resource Location
VM	Virtual Machine
VCN	Virtual Network Computing
WebDAV	Web-based Distributed Authoring and Versioning
WP	Work Package