



ICT 269978

Integrated Project of the 7th Framework Programme

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



VPH-Share

Work Package: WP2
Data and Compute Cloud Platform
Deliverable: D2.2
Design of the Cloud Platform
Version: 1.3
Date: 31/08/2011



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.eu		
EU Project officer	Joël Bacquet		

Work package	Number	2	Title	Data and Compute Cloud Platform
Deliverable	Number	2.2	Title	Design of the Cloud Platform

Date of delivery	Contractual	2011-08-31	Actual	2011-08-31
Status	Version 1.3		Final <input checked="" type="checkbox"/>	
Nature	Prototype <input type="checkbox"/> Report <input checked="" type="checkbox"/> Dissemination <input type="checkbox"/> Other <input type="checkbox"/>			
Dissemination Level	Public (PU) <input type="checkbox"/> Restricted to other Programme Participants (PP) <input type="checkbox"/> Consortium (CO) <input checked="" type="checkbox"/> Restricted to specified group (RE) <input type="checkbox"/>			

Authors (Partner)	Marian Bubak, Tomasz Bartyński, Marek Kasztelnik, Maciej Malawski, Jan Meizner, Piotr Nowakowski (CYFRONET) Spiros Koulouzis (UvA) David Chang, Stefan Zasada (UCL) Enric Sarries (AOSAE)			
Responsible Author	Piotr Nowakowski		Email	p.nowakowski@cyfronet.pl
	Partner	CYFRONET	Phone	n/a

Abstract (for dissemination)	This deliverable constitutes the design document of Work Package 2 of the VPH-Share project, devoted to designing, implementing and deploying the Cloud management platform and services for application deployment and execution. It covers the implementation details and technology-related information for a number of WP2 components, including the resource management layer, application execution services and tools for uniform data access and integrity monitoring.
Keywords	Cloud platforms, PaaS, IaaS, distributed systems, Cloud resource management, distributed data access



Version Log			
Issue Date	Version	Author	Change
2011-07-19	0.1	Piotr Nowakowski	First draft
2011-08-02	0.2	Piotr Nowakowski	Extended draft
2011-08-09	0.3	Piotr Nowakowski with contributions from CYF and UCL	Extended draft
2011-08-10	0.4	Piotr Nowakowski with contributions from AOSAE and UvA	Extended draft
2011-08-15	0.5	Piotr Nowakowski with contributions from UvA	Extended draft
2011-08-16	0.6	Piotr Nowakowski, Marian Bubak	Further refinements and updates as requested by WP2 management
2011-08-22	0.9	Piotr Nowakowski	Updated version following internal review
2011-08-29	1.0	Piotr Nowakowski, Marian Bubak, David Chang, Blanca Jordan Rodriguez, Maciej Malawski, Xavier Planes, Enric Sarries, Dmitry Vasunin	Additional updates following internal review
2011-08-30	1.2	Piotr Nowakowski	Additional updates and revisions
2011-08-31	1.3	PMO	Final proof read



Contents

Executive Summary	9
1 Introduction.....	9
2 VPH-Share User Groups and User Requirements Related to WP2	11
3 Work Package 2 Architecture Description	13
4 Detailed Design of Work Package 2 Components	16
4.1 Cloud resource allocation management	16
4.1.1 Functionality.....	17
4.1.2 Architecture	18
4.1.3 Deployment plan.....	20
4.1.4 Data stored in Atmosphere Internal Registry	21
4.1.5 Provided interfaces.....	22
4.1.6 Dependencies.....	23
4.1.7 Control flow.....	23
4.1.8 Candidate technologies	26
4.2 Cloud application deployment and execution	27
4.2.1 Functionality.....	27
4.2.2 Atomic Service Instance.....	29
4.2.3 Architecture	31
4.2.4 Provided interfaces.....	35
4.2.5 Dependencies.....	35
4.2.6 Control flow.....	35
4.2.7 Candidate technologies	41
4.3 Access to high-performance computing environments.....	42
4.3.1 Component description	42



4.3.2	How HPC fits into the overall architecture of WP2 and VPH-Share	44
4.3.3	Detailed Design	45
4.3.4	Audited Credential Delegation Security Component	53
4.3.5	Components.....	54
4.3.6	Interfaces	58
4.3.7	AHE/ACD implementation technologies.....	60
4.4	Access to large binary data in the cloud.....	60
4.4.1	Introduction	60
4.4.2	System Overview	62
4.4.3	Frontend	63
4.4.4	Operations Backend	64
4.4.5	Access Backend	65
4.4.6	Data persistence	65
4.4.7	Connection Module.....	66
4.4.8	Scaling.....	66
4.4.9	Usage scenarios	67
4.4.10	Implementation Technologies	68
4.5	Data reliability and integrity.....	68
4.5.1	Structure of a Managed Dataset.....	68
4.5.2	Tagging datasets	70
4.5.3	The DRI Runtime service.....	70
4.5.4	DRI Interfaces.....	72
4.5.5	Implementation Technologies	73
4.6	Security for Cloud applications	74
4.6.1	Architectural Design	75



4.6.2	Creation and Instantiation of Virtual Appliances	83
4.6.3	Invocation of Atomic Service.....	85
4.6.4	Policy Administration Point	86
4.6.5	Policy Information Point	88
4.6.6	Auditing and Logging	89
5	Implementation Methodology.....	90
6	Conclusions.....	92
7	References.....	93
8	List of Key Words/Abbreviations	97

LIST OF FIGURES

Figure 1: WP2 in the VPH-Share architecture	13
Figure 2: Overall architecture of the VPH-Share Data and Compute Cloud Platform (Work Package 2) and its relation to external Project components.	14
Figure 3: Use case diagram illustrating the roles of the application provider, the Atomic Service Cloud Facade (part of the WP6 Master Interface) and Atomic Service Instances accessing the features of the Allocation Management Service subsystem. The diagram also depicts indirectly used features.....	18
Figure 4: Architecture of the Allocation Management Service and its functional dependencies.....	19
Figure 5: The architecture and elements of the Atmosphere Internal Registry along with its interactions.....	20
Figure 6: Creation of a new Atomic Service Instance by the application provider. All interactions between the end user, the Master Interface, AMS and the Cloud Execution Environment are illustrated in chronological order.	25
Figure 7: Actors of the Cloud Execution Environment, namely the Allocation Management Service, the Atomic Service Cloud Facade and the application provider. Features of the system that are not directly accessed by users, but are required to provide CEE functionality, are also presented.	29



Figure 8: Structure of an Atomic Service Instance. The virtual machine (with a selected OS) hosts a VPH-Share application with a REST/SOAP interface exposed using wrapper mechanisms which involve a security module.	30
Figure 9: Cloud Execution Environment architecture overview. Components marked in light green are either developed within Task 2.2 or existing solutions that will be deployed and configured. Components that are marked in other colours are external to CEE but are significant to explain its architecture.	32
Figure 10: Architecture of a private Cloud deployment. Types of nodes (Cloud controller and compute) as well as network configuration are presented.	34
Figure 11: Creating a new Atomic Service. Interaction of the end user (application provider) with the infrastructure (Cloud Manager) is presented.....	38
Figure 12: Invocation of an Atomic Service Instance.....	39
Figure 13: Control flow involved in deploying an Atomic Service Instance.	40
Figure 14: VPH-Share Overview. The main aim of Task 2.3 is to provide access to grid infrastructure and tools to the VPH-Share infostructure.....	44
Figure 15: An overview of Task 2.3 within Work Package 2.	45
Figure 16: A typical AHE workflow.....	46
Figure 17: The AHE Job lifecycle state diagram.....	48
Figure 18: AHE Runtime module UML diagram.....	49
Figure 19: AHE Engine module UML diagram	49
Figure 20: A Simple JBPM workflow document example created using the Eclipse JBPM editor. JBPM supports complex processes which include human interaction, event handling as well as rules.	50
Figure 21: AHE Connector Module UML diagram.....	51
Figure 22: AHE API module UML diagram.....	52
Figure 23: AHE security module UML diagram.....	52
Figure 24: AHE storage module UML diagram	53
Figure 25: ACD wrapper paradigm	54
Figure 26: A typical ACD workflow. All commands are intercepted by ACD and checked before being sent to AHE.	55



Figure 27: ACD Authentication Service UML diagram	56
Figure 28: ACD Authorisation Service UML diagram.....	57
Figure 29: ACD Credential Repository Service UML diagram	58
Figure 30: LOBCDER interactions with application developers, end-users, and administrators	62
Figure 31: LOBCDER architecture.....	63
Figure 32: End users and developers can use standard clients and APIs to mount LOBCDER	64
Figure 33: StorageResource structure	65
Figure 34: DataResource structure.....	66
Figure 35: Distributed LOBCDER architecture	67
Figure 36: Schematic representation of a VPH-Share Managed Dataset.....	69
Figure 37: Autonomous operation of the DRI Runtime.....	71
Figure 38: Overall architecture of the VPH-Share Security Components	76
Figure 39: Security Proxies - Encrypting/Decrypting the AS requests and responses	77
Figure 40: Security Agent – Components Diagram and interfaces	78
Figure 41: Securing Atomic Services	84
Figure 42: Spawning a Secure Atomic Service Instance	85
Figure 43: Security Agent – Authorizing a request to an Atomic Service.....	86
Figure 44: Activation of PDP policies.	88
Figure 45: Security Audit – Use case for both the Management and the Agent’s event logging	90



EXECUTIVE SUMMARY

This deliverable constitutes the design document of Work Package 2 of the VPH-Share project, devoted to designing, implementing and deploying the Cloud management platform and services for application deployment and execution. The tools deployed by WP2 will constitute the VPH infostructure upon which domain-specific services can be provisioned to researchers and medical practitioners from the VPH consortium, in line with the Project's goal (1).

The role of this document is to provide an in-depth overview of how each component of the WP2 architecture is designed, how it is going to be implemented and deployed and how it is expected to integrate with other WP2 components (and with the VPH-Share project architecture in general). To this end, the document includes a summary section where the overall WP2 architecture is presented and each of the participating user groups is discussed, along with the ways in which these groups are expected to interact with the system. This general description is followed by specific technical details related to the implementation of WP2 subcomponents, including:

- deployment and execution of applications in Cloud infrastructures
- access to high performance computing (non-Cloud) infrastructures
- access to large binary data in the Cloud
- data integrity, availability and retrievability
- security aspects related to Cloud computations

This deliverable should be treated as a follow-up to the preceding WP2 document, namely the Analysis of the State of the Art and Work Package Definition (D2.1), published at the end of Project Month 3. The recommendations identified in the course of our research of the state of the art in the area of Cloud system management, distributed application deployment and distributed data storage translate into the design choices presented in this deliverable. User requirements were taken into account by means of a selection of detailed questionnaires distributed among and collected from the leaders of all four participating workflow teams. We further intend to coordinate our development efforts with users in the course of implementation and deployment of WP2 solutions. To this end, personal contacts have been established between WP2 members and user team representatives.

This document is meant as a live deliverable – should additional technologies become relevant to WP2 development at the implementation stage we intend to further address the topics discussed here when preparing subsequent WP2 deliverables. This document will also be extended as part of our consecutive prototype releases. WP2 periodic reports and prototype descriptions will therefore take into account any ongoing developments.

1 INTRODUCTION

The goal of Work Package 2 (Data and Compute Cloud Platform) is to develop, integrate and maintain an environment which will enable the VPH-Share workflows, as well as any



application making use of VPH-Share resources, to operate on top of the Cloud and high-performance computing infrastructure provided by the project.

In order to fulfil its goal, Work Package 2 needs to deliver a **consistent service-based system** that enables end users to deploy the basic components of VPH-Share application workflows (known as **Atomic Services**) on the available computing resources and then enact workflows using these services. The end-user interfaces (and – by extension – the Work Package 2 services which support them) must cater to each group of users expected to interact with the system. This division of responsibilities will be further elaborated upon in Section 2.

Given the above requirements, the primary aim of this document is to constitute an in-depth presentation of the structure and interactions of tools which, taken together, constitute the WP2 architecture. The document is structured as follows:

- Section 2 details the characteristics of each group of users who will interact with the WP2 platform (and with the VPH-Share system in general), listing their specific requirements and the ways in which such requirements impact the architecture of the Data and Compute Platform. It also presents some generic use cases, further explaining the relationships between application providers, end users and administrators, as well as the functionality which needs to be provided to each of these groups, as identified on the basis of our discussions with VPH-Share workflow developers and application providers.
- Section 3 is meant as a generalised overview of the WP2 architecture. It does not include detailed descriptions of individual components; rather, it serves as a “big picture” introduction to the way in which Work Package 2 is structured and the interactions between its constituent parts.
- Section 4, the most extensive part of this deliverable, is meant as an in-depth description of each of the components identified in the preceding section. For each of the Work Package 2 technical tasks, a thorough discussion of the implementation concepts and technology choices is provided. This discussion is meant to address the following issues (on a per-component basis):
 - component description (How does the component work? How does it fit into the overall architecture of VPH-Share and, specifically, of WP2?);
 - detailed design (A textual description illustrated by UML class/sequence diagrams);
 - interfaces (What interfaces will the component provide to other components? What interfaces will it require of other components?);
 - Implementation technologies (Which technologies will be used to implement the component?),
- Section 5 focuses on development methodologies, laying out the blueprint for the implementation of the initial prototype of the WP2 platform which is due by Project Month 12.
- Section 6 summarises the presented descriptions and contains general conclusions.



2 VPH-SHARE USER GROUPS AND USER REQUIREMENTS RELATED TO WP2

The goal of VPH-Share is to develop the organisational fabric (the *infostructure*) and integrate optimised services to expose and share data and knowledge, jointly develop multiscale models for the composition of new VPH workflows and facilitate collaborations within the VPH community. Thus, the Project should enable groups of users to gain authorised access to a variety of computational and data services, deployed on distributed hardware resources. Most of the technologies presented with the data and compute platform are exclusively implemented as services – applications or other necessary pieces of logic that encapsulate the data they operate on and provide secure interfaces to access them. In this sense, the function of WP2 is to provide a Cloud and HPC platform on which to deploy, instantiate, access and manage VPH-Share services (which are understood as applications, or components thereof, fulfilling specific needs of researchers). This approach allows each application to evolve at its own pace thereby reducing the side effects traditionally seen in former enterprise applications. The starting point for the development of VPH-Share solutions is a selection of standalone applications derived from four participating workflows: @neurist, EUHeart, ViroLab and VPHOP. Each of these projects operates a selection of software tools, presently provided only to its consortium members. With the aid of VPH-Share these tools are meant to be exposed to a wider community of users and potential collaborators. The applications will need to be prepared for deployment in a distributed Cloud environment and a set of interfaces will need to be provided for end users, enabling them to interact with the exposed tools in a secure and convenient way.

As a consequence of the above, and also with respect to the Project's Description of Work (1) and basing on the workflow questionnaires distributed and collected by WP2 during this preparatory phase, three specific groups of users were identified in the context of VPH-Share. These are as follows:

- 🌐 **Application providers** (also called **developers**): These are the people responsible for developing and installing scientific applications and software packages, as well as provisioning input data required by such applications to operate. Typically, this group would comprise IT experts who collaborate with domain scientists and translate their requirements into executable software. Within the context of VPH-Share developers are tasked with installing pre-existing applications and components on the virtualised hardware resources provided by the Project so that these applications can be provisioned to domain scientists (see below).
- 🌐 **Domain scientists**: This group comprises the actual researchers (belonging to the VPH-* project community) who stand to benefit from access to scientific software packages provided to them by means of the VPH-Share platform. To some extent the entire VPH-Share infrastructure exists to support and provide added value to these users and is one of the determining factors by which the success of the project may be judged. In general, the scientists will require the ability to access the applications in a secure and convenient manner, making use of graphical interfaces that will be provided through WP6.
- 🌐 **System administrators**: A group of privileged users who will be able to manipulate and assign the available hardware resources to the Project and define security/access policies for other groups of users. Administrators will be tasked with making sure the platform



remains in an operational state and that no unauthorised or harmful activity is effected with the use of VPH-Share resources. Administrators will also be able to monitor system usage statistics and respond to emerging issues by taking advantage of notification mechanisms built into the system.

The following table summarises the mapping between user groups and technical WP2 tasks, to better illustrate how the proposed architecture of WP2 corresponds to the user requirements. For specific use cases and their descriptions, please refer to Section 4 which contains in-depth presentation of each technical task of WP2.

Technical task	Targeted use cases
Task 2.1: Cloud Resource Allocation Management	<p>Application providers: Deploy and register Atomic Services;</p> <p>Domain scientists: Browse available Atomic Services;</p> <p>System administrators: Browse and manage available Cloud computing resources; register new resources; set allocation policies;</p>
Task 2.2: Cloud Application Deployment and Execution	<p>Application providers: Request deployment of Atomic Service Instances for development and testing purposes;</p> <p>Domain scientists: Request access to specific Atomic Services via workflow management tools or directly (with the use of APIs/GUIs embedded in the Master Interface);</p> <p>System administrators: Set deployment properties for each Atomic Service;</p>
Task 2.3: Access to High-Performance Computing Infrastructures	<p>Application providers: Request execution of HPC tasks for development and testing purposes;</p> <p>Domain scientists: Request access to specific HPC-based Atomic Services via workflow management tools or directly (with the use of APIs/GUIs embedded in the Master Interface);</p> <p>System administrators: Manage HPC resources attached to the Project; review logs and monitoring data;</p>
Task 2.4: Access to Large Binary Data in the Cloud	<p>Application providers: Query for and store binary data generated by VPH-Share Atomic Services;</p> <p>Domain scientists: Download and utilise the binary data produced by VPH-Share application workflows;</p> <p>System administrators: Manage VPH-Share data storage resources;</p>
Task 2.5: Data Reliability and Integrity	<p>Application providers: Tag datasets for automatic reliability/accessibility monitoring; set monitoring, validation and replication policies;</p> <p>Domain scientists: Access verified datasets regardless of location in the VPH-Share data federation;</p> <p>System administrators: Receive notifications in case of access problems or policy violations;</p>
Task 2.6: Security	<p>Application providers: Safely deploy applications and</p>

	expose them to selected (authorised) groups of users; Domain scientists: Access any VPH-Share application component with common credentials; System administrators: Manage access rights; add/remove users; set user attributes.
--	--

The next section explains how the system intends to cater to each of these user groups and how the individual components of Work Package 2 come together to enable provisioning of integrated services to all types of VPH-Share users.

3 WORK PACKAGE 2 ARCHITECTURE DESCRIPTION

An overview of Work Package 2 and its relationship with the overarching VPH-Share architecture is presented in Figure 1. It should be noted that this is strictly a conceptual view as this deliverable focuses on detailing the internal architecture of WP2. The Project Consortium plans to release a separate document by the end of Project Month 8, where the overall architecture of the entire Project will be presented in detail. Specific information regarding the integration of WP2 tools with other components (external to WP2) can be found in the relevant subsections of Section 4.

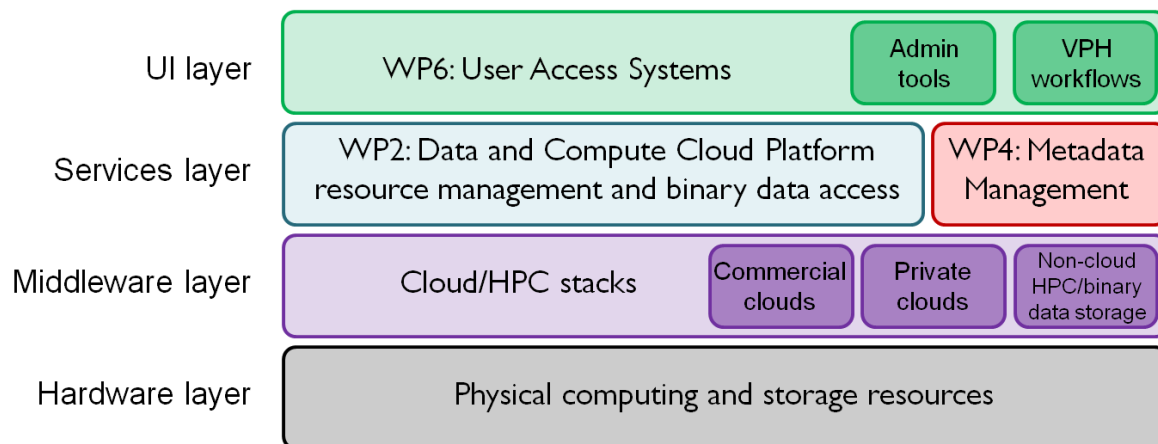


Figure 1: WP2 in the VPH-Share architecture

The projected architecture of Work Package 2 reflects the structure of the WP in the Project’s Description of Work (1). Each of the technical tasks of WP2 translates into either a specific component of the proposed architecture, or into several such components. It should also be noted that since the Cloud deployment and execution platform (being implemented within WP2) is a crucial element of the VPH-Share architecture, significant attention will be devoted to description of inter-component interfaces and relation to other tools and services provided by the Project.

In light of the above, a view of the WP2 architecture is presented in Figure 2. The diagram covers the basic components of WP2, along with graphical representations of inter-

component interactions and positioning of WP2 with relation to other Work Packages of the Project (WP6 in particular).

Several observations of a general nature should be made at this point. First, the notion of the Atomic Service, as presented in the Description of Work (and further explained in Deliverable 2.1 (2)) is central to understanding the features and modes of operation of the designed platform. It is important to note that each application (or component thereof) needs to be treated as a service if it is to be managed by WP2 components and deployed in the Cloud. Along with a schematic depiction of the basic building blocks of the Atmosphere framework, Figure 2 also presents the structure of an individual Atomic Service, listing the libraries and tools which will be prepared by WP2 and preinstalled on all virtual machines hosting Atomic Services within the context of VPH-Share. The specific features and structure of each of these components will be discussed in Section 4.

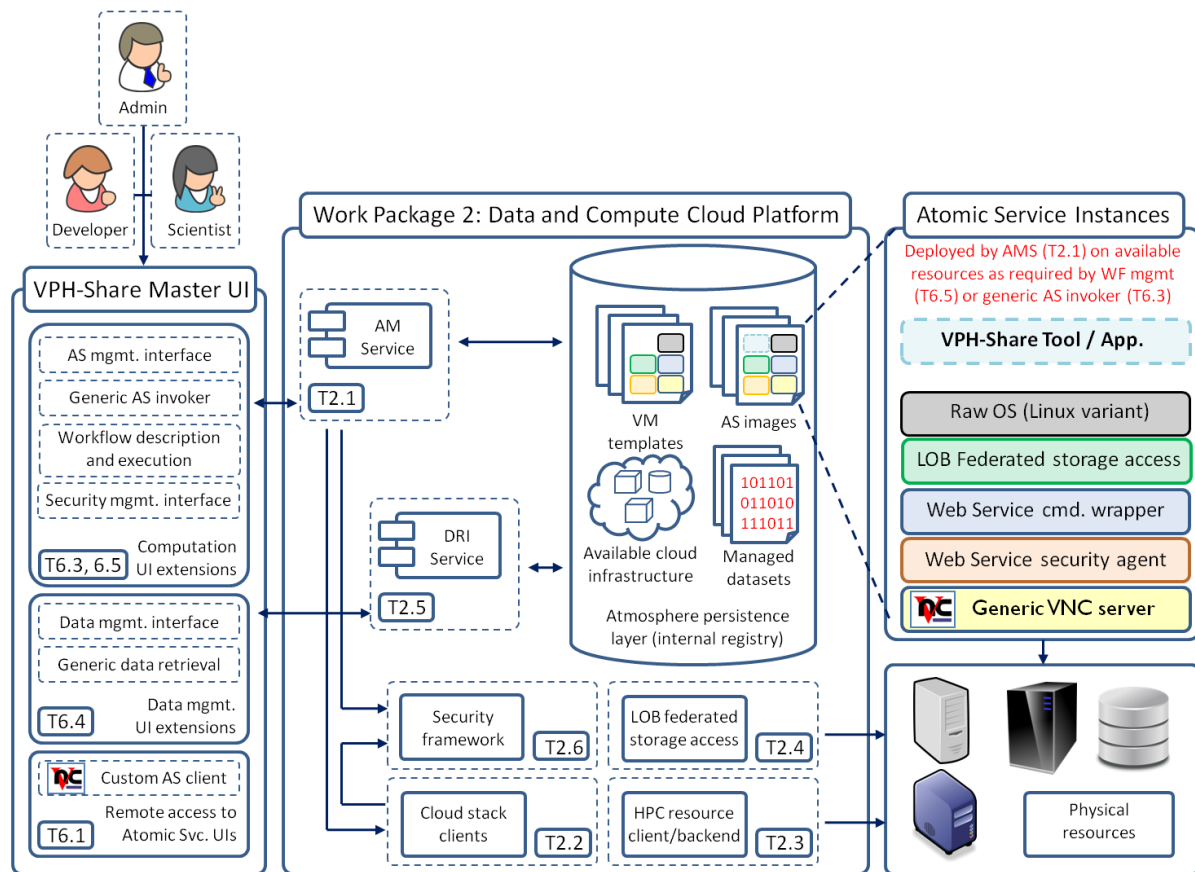


Figure 2: Overall architecture of the VPH-Share Data and Compute Cloud Platform (Work Package 2) and its relation to external Project components.

In light of the above requirements, several operations have to be performed in any standalone, command-line-based application before it can become part of the VPH-Share framework:



- As mandated by the Description of Work (1), the application needs to expose a remote interface based upon the Web Services technology. Either the application is already engineered to present such an interface, or it must be reengineered to comply with this requirement. Naturally, when legacy applications are concerned (and VPH-Share will involve numerous such applications), it cannot be expected that developers will reimplement them only to suit the requirements of VPH-Share. Thus, a different procedure is planned, where the legacy application is instead *wrapped* by an external client which provides the required Web Service interface to the WP2 tools (and to other components which wish to interact with the application), while internally invoking the command-line interface(s) that the application provides. A generic description of how a legacy application can be wrapped to comply with Atomic Service requirements can be found in Section 4.1.7. Work Package 6 will then attempt to use the WP2 tools enable components of pilot workflows (derived from ViroLab, VPHOP, @neurist and EUHeart projects) to function within the VPH-Share infrastructure.
- The Web Service enabled application is then installed onto a virtual machine image provided to the developers by the Atmosphere component, which implements the functionality associated with Task 2.1 of the Project. Taking advantage of virtualisation technologies and OS independence features of modern Cloud solutions, Atmosphere does not need to enforce a specific programming environment or operating system – instead, a selection of virtualised platforms will be offered to developers. The developer will need to select a specific OS template, which will come with preinstalled components enabling VPH-Share Atomic Services to function. Having made this choice, the developer will be presented with a persistent instance of the template, deployed upon Cloud resources, where the application (or parts thereof) can be installed. In fact, the virtual machine provided to developers can be directly used to wrap application components into Atomic Services, with no further hardware requirements. While installing and testing their application, developers may log in to the virtual machine directly, via the SSH protocol, with credentials supplied to them by Atmosphere (which is also responsible for instantiating and managing the virtual machine in question).
- Following installation of the application “payload” (i.e. the application component depicted in the top right-hand corner of Figure 2), the Atomic Service can be registered and stored in the Atmosphere internal registry. This operation can be performed by interacting with the dedicated Atmosphere portlet which will be embedded in the VPH-Share Master UI and will provide access to specific features of the Atmosphere component. Upon registration, Atmosphere will store a copy of the Atomic Service virtual machine image and will later use it to instantiate Atomic Service Instances that correspond to the specific application (or application component). The developer’s work concludes at this point as the Atomic Service is now ready for use and can be dynamically instantiated and served to end users of the Project (i.e. researchers). Note that should further development work become necessary (for instance to upgrade the Atomic Service, or to resolve issues/fix bugs), the developer may again check out the specific VM instance and perform the required actions before committing the updated resource back to the Atmosphere storage layer.

An interesting issue arises with respect to applications that need to provide a graphical user interface. Naturally, an application that is not directly input-driven, does not normally expose



command-line interfaces, and hence cannot be easily wrapped and deployed as an Atomic Service. In such cases WP2 aims to preserve the existing features of the application by enabling the virtual machine on which it is hosted to expose a remote GUI with help from a remote desktop-type mechanism, specifically the Virtual Network Computing (3) interface. The virtual machine (on which applications are deployed) includes a generic VNC (Virtual Network Computing) server while the client front-end (embedded in the VPH-Share Master Interface) includes a portlet that exposes the application's GUI in the browser window and enables clients to directly interact with the service.

4 DETAILED DESIGN OF WORK PACKAGE 2 COMPONENTS

4.1 Cloud resource allocation management

Resource allocation management is required to ensure that all computational tasks are assigned an appropriate share of underlying cloud or HPC resources. As Atomic Service Instances (ASI) will perform processing tasks, efficient management of those instances is the main goal of Task 2.1. An Atomic Service Instance is a computer system that:

- has a VPH-Share application installed
- uses wrapping mechanisms provided by Task 2.2 to expose the application as a HTTP service (either through SOAP or REST Web Service protocols)
- secures access to the application using tools provided by Task 2.6
- has VPH-Share federated data storage access tools installed
- optionally includes tools to directly connect to the machine (such as SSH or VNC server)
- is hosted in the cloud infrastructure (either private or commercial installation) or in an HPC infrastructure, depending on resource requirements of the application

A detailed description of Atomic Service Instances is presented in Section 4.2.2.

Commercial cloud providers employ the pay-per-use model while private infrastructures have limited amounts of resources. Submitting a job to HPC infrastructures usually requires waiting in a queue and consumes computational grants. The usage demand for most Atomic Service Instances will be dynamic and may frequently be prone to spikes in demand. Matching capacity to actual Atomic Service Instance usage make dynamically optimised deployment of instances a necessity. Adjusting the computational environment and providing dynamic and on-demand features required by end users involves:

- shutting down instances of a given type (if there are too many instances of this specific type or this type of instance is not required at the moment)
- initializing instances of similar or another type to handle incoming traffic
- configuring instances
- enabling application providers to install their software on machines with operating systems of their choice
- monitoring the cloud infrastructure and the performance of instances
- controlling and minimizing the cost of hosting instances



It is extremely difficult – if not outright impossible – for system administrators to manually harness such a dynamic and complex environment, composed of private and commercial clouds as well as HPC resources. Therefore Task 2.1 aims to develop and deploy a computer system – the Allocation Management Service – to assist system administrators in performing their assigned tasks.

4.1.1 Functionality

The Allocation Management Service (AMS) is a subsystem of the VPH-Share Data and Compute Cloud Platform. Its functionality is depicted in Figure 3. AMS is accessed by three classes of actors:

1. **VPH-Share application providers** (already discussed in Section 2) who will be able to use graphical tools exposed by the Master User Interface to:
 - a. Browse available virtual machine templates of raw operating systems that can be used to create new Atomic Services. The application provider may wish to use a specific distribution of an operating system, depending on the requirements of their application. Some of the major issues which must be taken into account when choosing OS distribution include availability of libraries and tools, robustness, system overhead and individual preferences.
 - b. Create a new virtual machine based on the selected raw OS distribution. The application provider does not need to know where and how a virtual machine will be created on the underlying infrastructure. From the user's perspective, this will be a single-click operation that will return the IP address of the created virtual machines and credentials necessary to log into each machine. Connecting to a virtual machine and installing applications (creating an Atomic Service) is described in Section 4.2.
 - c. Save the newly configured virtual machine with the VPH-Share application installed as a new Atomic Service. This operation can also be invoked using a graphical tool embedded in the Master User Interface and must require a description of the newly created Atomic Service. AMS will interface the underlying layers in order to actually save the virtual machine with the preinstalled VPH-Share application as a new template and register a new Atomic Service in the WP2 Internal Registry.
2. **The Atomic Service Cloud Facade**, acting on behalf of end users (scientists), either by invoking the functionality of a single Atomic Service or executing a more complex workflow that involves multiple calls to a range of Atomic Services. In both cases, AMS will be responsible for ensuring that the required Atomic Service Instances are configured, deployed and monitored properly. Furthermore, AMS will try to allocate resources in a way that maximises application performance and minimises costs. Despite the fact that the AMS is unnoticed by the end users, it will perform complex tasks to deliver this functionality. It has to provision Atomic Service Instances on demand, in an optimal manner, on the basis of an optimal deployment plan enacted using the Cloud Execution Environment (CEE). In order to develop such a plan AMS needs to query CEE for monitoring data describing infrastructure load and performance of ASIs, and then collate this data with CEE specific policies and ASI-specific resource demands and usage costs. For more details about the deployment plan please refer to Section 4.1.2.

3. **Generic subsystems or components developed or deployed within WP2** (including Atomic Service Instances and the Data Reliability and Integrity runtime – see Section 4.5). A good example of such an actor is an Atomic Service Instance that queries AMS for configuration parameters required to initialise and properly customise itself. Atomic Service Instance configuration may contain information regarding security, data sources that should be accessed or any other application-specific data required to initialise the service. Another example is the Data Reliability and Integrity service (see Section 4.5.4) that stores and access metadata describing managed datasets (see Section 4.5.1).

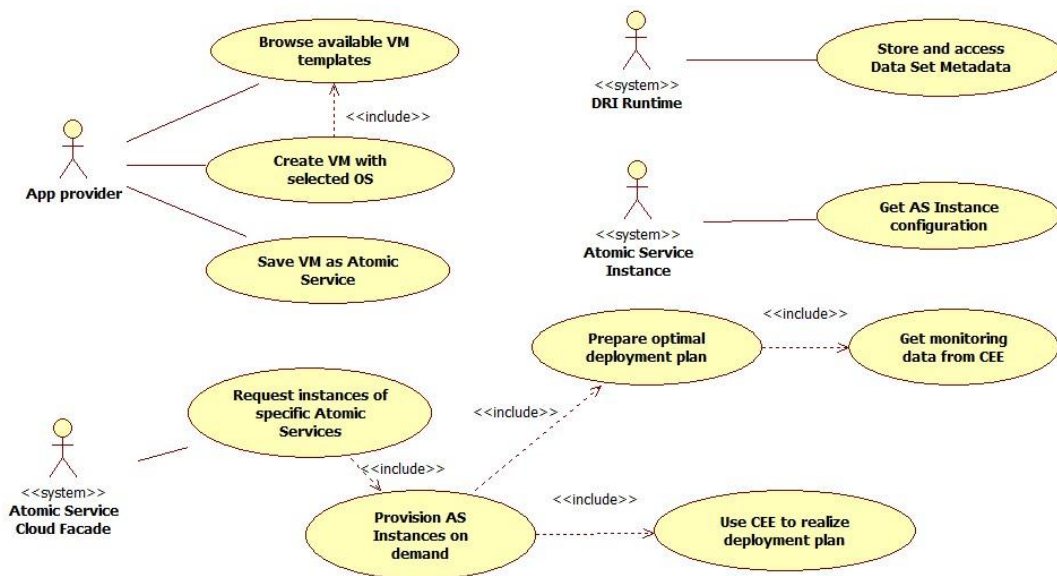





Figure 3: Use case diagram illustrating the roles of the application provider, the Atomic Service Cloud Facade (part of the WP6 Master Interface) and Atomic Service Instances accessing the features of the Allocation Management Service subsystem. The diagram also depicts indirectly used features.

4.1.2 Architecture

The Allocation Management Service subsystem is part of the Data and Compute Cloud Platform. It is subdivided into components dedicated to specific features. Modularisation allows independent development of components implementing separate aspects of the system’s functionality. The AMS architecture is illustrated in Figure 4. Three key components can be distinguished:

-  Manager
-  Optimiser
-  Atmosphere Internal Registry (AIR)

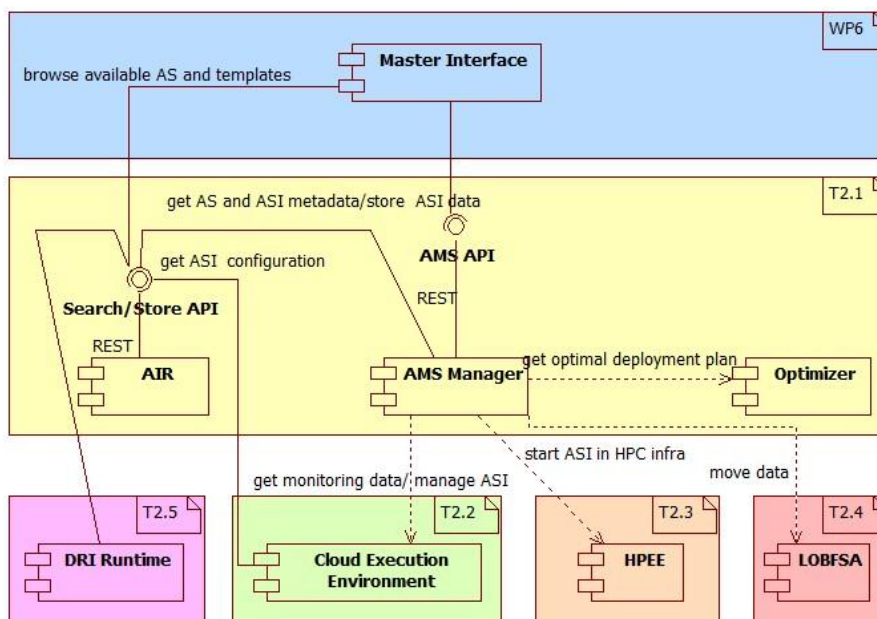


Figure 4: Architecture of the Allocation Management Service and its functional dependencies.

The Manager is a central component of the AMS subsystem which will supervise the process of preparing the optimal deployment plan. It will also provide a remote REST interface that accepts requests from the Atomic Service Cloud Facade regarding currently required Atomic Service Instances, and requests from application providers to create or save new Atomic Service Instances. The Manager will interface the Cloud Execution Environment to obtain the current status of the underlying infrastructure and dispatch deployment plans that will result in starting or stopping instances. If the deployment plan involves execution of applications on an HPC infrastructure, the Manager will contact the High Performance Execution Environment. The Atmosphere Internal Registry (AIR) will be used as the Manager’s persistence layer; thus the AMS subsystem will be able to survive a crash or reboot and maintain control over the underlying resources.

Optimisation logic will be encapsulated in a separate module. The Optimiser will implement the process of preparing an optimal deployment plan. There are many factors which might influence how Atomic Service Instances should be located and how much resources they should consume (for a comprehensive list of factors that will be taken into account please refer to Section 4.1.2). Thus, an approach based on multiple criteria must be applied. We intend to experiment with various tools and techniques, including the Modelling Language for Mathematical Programming (AMPL) (4) combined with the DONLP2 (5) solver, constraint satisfaction programming using CHOCO (6) or finding Pareto-optimal solutions and normalizing them using objective functions (for a detailed description of these tools and techniques please refer to Deliverable 2.1 (2)). It is expected that various optimisation policies will be applied and tested throughout the lifecycle of the VPH-Share project – hence, the Optimiser component must be easily replaceable. To achieve this goal, the Optimiser will only be used by the Manager component and will not have any dependencies on other

components. Each invocation of the Optimiser (by the Manager) will include all data necessary to perform optimisation.

The Atmosphere Internal Registry (hereafter also referred to as the Atmosphere Registry, the AIR component or simply the Registry) is a core element of the Atmosphere platform, delivering persistence capabilities. Its components and interactions are depicted in Figure 5. The main function of AIR is to provide a technical means and an API layer for other components of Atmosphere to store and retrieve their crucial metadata. Having a logically centralised (though physically dispersed, if needed to meet high availability requirements) metadata storage component is beneficial for the platform, as multiple elements may use it not only to preserve their “memory” but also to persistently exchange data. This is facilitated through the well-known database sharing model where the data storage layer serves as a means of communication between autonomous components, making the Atmosphere Internal Registry an important element of the platform.

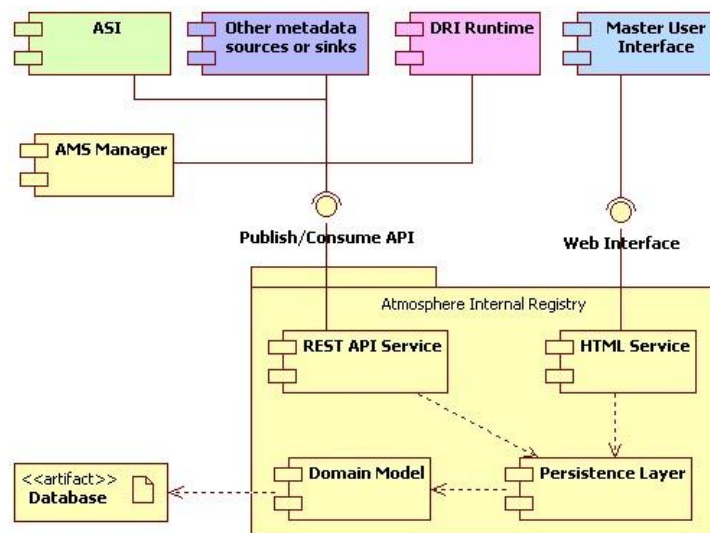


Figure 5: The architecture and elements of the Atmosphere Internal Registry along with its interactions.

4.1.3 Deployment plan

The deployment plan is the most significant concept of the AMS subsystem. All user requests (except requests for browsing available templates and obtaining instance configuration parameters) will result in preparing a new deployment plan that will be dispatched to the Cloud Execution Environment. This takes place automatically, based on platform requirements and the information available in the Atmosphere Internal Registry (see previous section). The deployment plan is a formal description of actions that are required at a specific point in time to provision Atomic Service Instances implementing the features requested by end users. Such a plan needs to be optimal in terms of performance of Atomic Service Instances and costs of computation, data storage and transfer. The deployment plan will specify:

- 🌐 which Atomic Service Instances should be available at the given moment



- where each Atomic Service Instance should be initialised (partner’s private Cloud site, commercial Cloud, hybrid installation or HPC infrastructure)
- the quantity and size of instances (i.e. the amount of allocated resources)

The deployment plan will be expressed as a list of actions which need to be taken to fine-tune the computing environment. Those actions may concern:

- managing virtual machines in Clouds (starting, stopping etc.)
- moving data using binary data access tools
- starting/stopping an application in the HPC infrastructure

The deployment plan will be passed to the Cloud Execution Environment and the High Performance Execution Environment in order to effect the specified resource allocation.

As the deployment plan needs to be optimised, the Allocation Management Service will take into account several factors. The most significant of these are as follows:

- is it more efficient to transfer input data to the site where the atomic service is deployed or instantiate the service close to existing datastores
- workflow and atomic service resource demands
- volume and location of input and output data
- load of available resources
- cost of acquiring resources on private and public Cloud sites
- cost of transferring data between private and public Clouds (also between “availability zones” such as US and Europe)
- cost of using cheaper instances (whenever possible and sufficient; e.g. EC2 Spot Instances or S3 Reduced Redundancy Storage for some noncritical (temporary) data)
- public Cloud provider billing model (Amazon charges for a full hour – thus, five 10-minute tasks would cost 5 more times to run than an individual instance)
- security constrains (for instance: “sensitive data cannot be transferred to public Cloud infrastructure”)
- the possibility of reusing pre-deployed instances (sharing instances between workflows)

4.1.4 Data stored in Atmosphere Internal Registry

While the Registry will be prepared to accommodate a wide range of different metadata – since its internal mechanisms are based on the Semantic Integration (7) concept which delivers a high level of generality for domain-specific metadata solutions – eventually the following elements will be stored inside the Registry:

- Atomic Service configurations: a set of runtime parameters or documents containing such parameters as are required to deploy an Atomic Service template and set it up to serve a running and externally available Atomic Service Instance
- Metadata describing the properties of the Atomic Service – for instance whether the Atomic Service is stateless or stateful, what are its computational requirements etc.
- Available templates: the list of Atomic Service templates available for Atmosphere to be instantly deployed and used in applications when a need arises. While the virtual machine



- images of these templates are stored in the Cloud stack storage elements, the metadata describing and identifying them is inside the Registry
- Hosts and operating Atomic Service Instances: the list of available hosting machines which are able to accept new instances of Atomic Services to be deployed, as well as the list of all such instances currently deployed, running and available for application workflows
 - Datasets: the list of large binary data sets and storage resources required by Data Reliability and Integrity tools to monitor the availability of managed data resources (see Section 4.5.1)
 - (optional) Real-time measurements of host parameters: when deploying new Atomic Service Instances the AMS Manager may require historic data regarding performance and load of available hosts – if this is the case, the Registry will serve as a temporary buffer for recent measurements of the most vital parameters (such as CPU load, memory usage, available disk space etc.)

4.1.5 Provided interfaces

The following interfaces will be provided by Atmosphere to external actors.

4.1.5.1 AMS Manager

A RESTful interface for managing Atomic Service Instances will be provided by the Manager component. Two actions will be supported:

- Requesting the provisioning of specific resources (a fresh virtual machine for service installation, an Atomic Service Instance of a given type or an application running in the HPC infrastructure). All input parameters will be encoded in the JSON (JavaScript Object Notation) format.
- Informing AMS that a particular service is no longer needed and can be stopped.

4.1.5.2 Atmosphere Internal Registry

In general, there are two modes of interfacing the Atmosphere Internal Registry (as depicted in Figure 5). Standard interaction is handled by a remote RESTful API, providing a set of operations based on the HTTP protocol with signatures described in terms of:

- the URL endpoint to be used when invoking an operation
- the list of required and optional parameters which should (or might) be passed in the HTTP call to parameterise the output
- the structure of the expected output or the list of possible error messages

This API will be made available for external entities (mainly the AMS Manager and DRI Runtime components, but also for any other element of the VPH-Share environment which may need to interface the Registry) to store, retrieve and manage all the metadata stored in the Registry. The interface will also go beyond the basic (atomic) CRUD (Create, Read, Update and Delete) set of methods – custom operations will be added on a case-by-case basis when deemed useful for external components. Examples of such specific operations are:



- “list all running Atomic Services which follow a specific AS configuration”
- “for a given dataset, list all storage resources on which it is available”
- “list all hosts which do not currently contain any deployed ASI”

Providing such dedicated operations will facilitate straightforward development of Registry clients.

Apart from the RESTful programmer’s interface, the Registry will also expose another interface, dedicated to human users (see Figure 5 again). Any authorised user – typically an administrator (see Section 2 for a discussion of user roles) – can access this interface in order to modify, register or remove content from the entire domain. Additionally, some restricted access modes might also be included for all VPH-Share participants to view and browse the contents of the Registry. As some initial metadata has to be fed into the system by human users (for instance the prepared AS templates have to be registered by hand), the provisioning of such a Web interface will be prioritised when developing and deploying the first version of the Atmosphere Internal Registry. In order to maintain consistency throughout the Project, the Web interface will be embedded within the Master User Interface. Integration will be performed using the *mashup* methodology (8), with the Registry UI occupying an autonomous section of the interface, backed up by separate web server (which will also be responsible for serving the aforementioned RESTful API).

4.1.6 Dependencies

The Allocation Management Service has three external dependencies:

- The Cloud Execution Environment (Task 2.2) will host and provision ASIs on demand. It will also store data describing the status of the cloud infrastructure for the purposes of creating an optimal deployment plan. Finally, CEE will realise the deployment plan by starting/stopping Atomic Service Instances accordingly.
- Large Object Binary Federated Storage Access (Task 2.4) will be used to realise part of deployment plan concerning data management. This will include replicating/moving binary data to the required storage resources.
- The High-Performance Execution Environment will implement parts of the deployment related to provisioning Atomic Service Instances requiring HPC resources.

4.1.7 Control flow

This section explains how the functionality listed in Section 4.1.1 will be delivered. It focuses on explaining interactions between components, starting with those that are close to the end users (Master Interface), going through the AMS in the middle and ending with CEE. Creating a new Atomic Service Instance will be described in more detail. Other scenarios will also be explained, showing how they differ from the former.

The process by which application providers create a new Atomic Service Instance is described in Figure 6:



- The AMS Manager needs to store an up-to-date status of the infrastructure – therefore it queries the Cloud Execution Environment for current monitoring data.
- The monitoring component of CEE returns the available resources and the load of the infrastructure. This data is stored in the Internal Registry for further use.
- When the user opens the Master Interface and wishes to browse VM templates that are available for creating a new ASI, a request is sent to the Atmosphere Internal Registry.
- The reply contains a list of possible VM template choices. This information is presented to the end user by the Master Interface GUI.
- The application provider subsequently selects a template that will be spawned as a new ASI, resulting in a request to the AMS Manager to instantiate a new virtual machine from a specific template in the cloud infrastructure. The manager can also optionally ask AIR for the status of the infrastructure and receive data describing the available resources and their load.
- The AMS Manager invokes the Optimiser which analyses the current load of the infrastructure and prepares an optimal deployment plan (see Section 4.1.3), specifying that a new virtual machine needs to be spawned on a specific (private or public) cloud site.
- The Manager stores configuration data required by ASI to start a service in the Internal Registry. A deployment plan is sent to CEE cloud clients, which implement it by spawning a new virtual machine.
- The Cloud clients return the IP address of the virtual machine and the credentials needed for logging in to the Manager. As the virtual machine boots up, it may require some additional configuration in order to initialise its services. This configuration is obtained from the Internal Registry via a RESTful API.
- Once the virtual machine is running and configured its address and credentials are forwarded to the Master Interface and presented to the application provider to connect to the machine and install additional software (see Section 4.2.2).

Once the application provider has installed and configured the required software, the virtual machine can be saved as a VPH-Share Atomic Service. The sequence of operations is very similar to the one described above. The user opens the Master Interface and requests that their virtual machine be registered. This request is then delegated to the AMS which in turn contacts the CEE. The virtual machine is stopped and its image converted into a template which can later be used to spawn further Atomic Service Instances.

Requesting specific Atomic Service Instances follows a similar pattern. The Atomic Service Cloud Facade, as part of the Master Interface, contacts the AMS Manager to supervise the creation of a deployment plan. The Manager then queries AIR for the required metadata describing Atomic Service Instances, and for the status of the cloud infrastructure. It subsequently invokes the Optimiser that prepares an optimal deployment plan. On this basis the Manager can dispatch requests to CEE to start/stop ASIs in the cloud, HPEE (High Performance Execution Environment, see Section 4.3) to start applications and LOBCDER (see Section 4.4) to move data. As this is conceptually very similar to the ASI creation process depicted in Figure 6 we will omit a separate sequence diagram describing this case (to maintain conciseness).

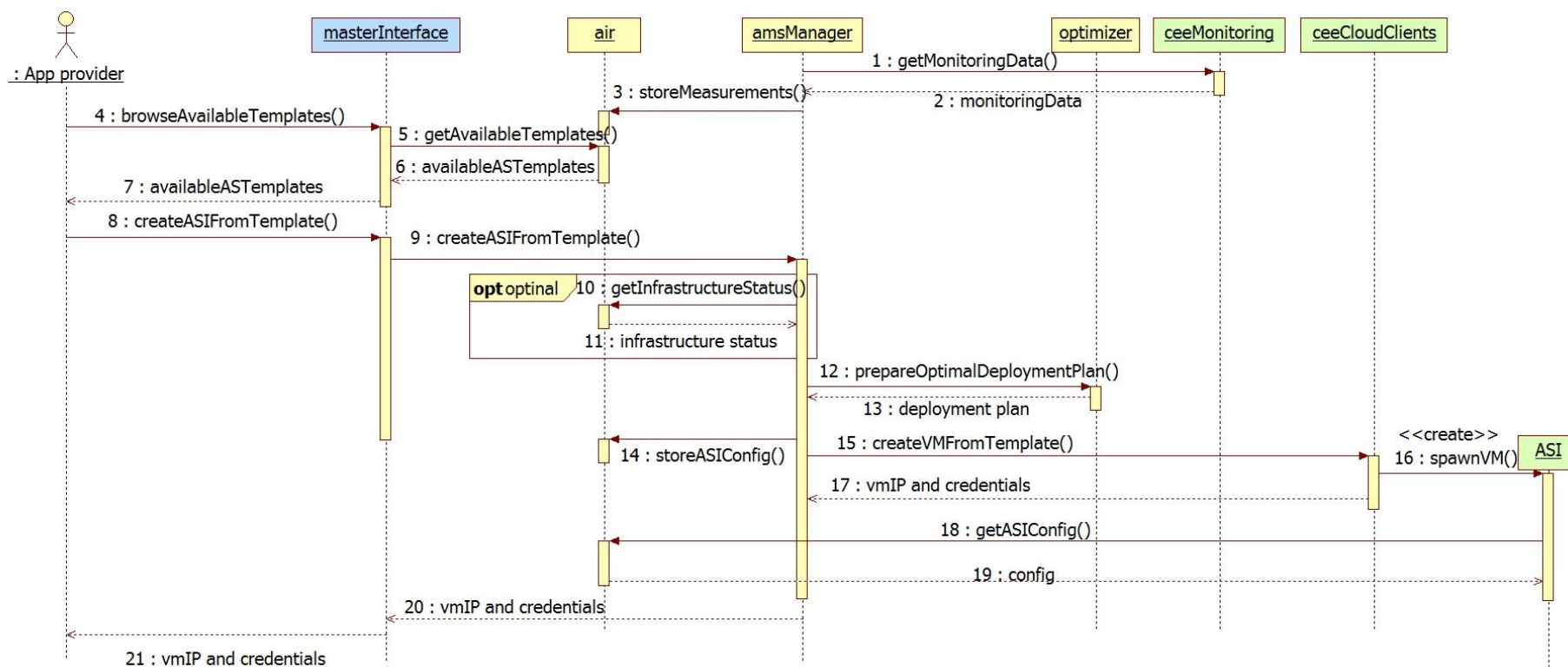


Figure 6: Creation of a new Atomic Service Instance by the application provider. All interactions between the end user, the Master Interface, AMS and the Cloud Execution Environment are illustrated in chronological order.



Using the AIR persistence service is quite straightforward. Atomic Service Instances (or the DRI Runtime) use a RESTful API to obtain or store data in its underlying database. The Master User Interface contacts the AIR Web interface to present users with requested data.

4.1.8 Candidate technologies

The AMS Manager and Optimiser components will be implemented in a general-purpose programming language, namely Java SE6. Proven open-source libraries and tools will be used to facilitate the development process. The Optimiser interacts only with the Manager so it is very reasonable to implement local communication between these components. They will be deployed as OSGi (9) bundles, facilitating dependency management and enabling us to easily switch optimisers implementing different optimisation policies. Bundles will be deployed in an Apache Karaf (10) container, conserving system memory and making maintenance easier. This approach ensures low communication overheads and allows these modules to be developed independently. If performance considerations force us to distribute these components, this can be easily achieved by deploying them in separate containers running on two or more servers and switching to remote communication. Apache Camel (11) will be used as the integration framework.

The development of the Atmosphere Internal Registry will be based exclusively on an open-source software stack. Below we present a list of candidates which are currently the technologies of choice for the implementation of the Registry. If, at some point during the course of the Project, new requirements emerge, the list of technologies may have to be extended.

The implementation of the Atmosphere Internal Registry will be based on the following tools:

- The persistence layer will be provided by MongoDB (12), a schemaless NoSQL database: this allows for flexible adaptation to a growing and rapidly changing metadata model
- The domain model and the domain-specific logic layer will be developed in Ruby due to its highly dynamic nature: this is especially important for the Semantic Integration methodology (7) we have chosen to adopt
- External interfaces and the web application will base on Sinatra (13) and Phusion Passenger (14) technologies – two stable solutions for this type of software

4.1.8.1 Methodology

The methodology of development assumes tight and rapid development cycles. Accordingly, the first prototype release will be issued relatively early on in the development process, in order to make the tool available to the community as soon as possible and to gather valuable feedback for future versions. These will be released in frequent “small delta” increments, ensuring faster response to user requests and lowering the detrimental impact of regression bugs.



4.1.8.2 Security

All components exposing publicly available RESTful interfaces will be secured with mechanisms provided by Task 2.6 (see Section 4.6).

4.2 Cloud application deployment and execution

The Allocation Management Service – itself a component of the WP2 framework, deployed on available computing resources by the VPH-Share developers – is tasked with creating an optimal deployment plan. Part of the plan specifies how Atomic Service Instances should be deployed in the Cloud environment. The Cloud Execution Environment forms part of the Data and Compute Cloud Platform which will be used to host instances and manage them in accordance with the deployment plan. Its main goal is to:

- Provide mechanisms for turning domain applications into Atomic Service Instances
- Hosting Atomic Service Instances in private and public Cloud infrastructures
- Provide a means and an API for managing ASIs

Many commercial providers offer mature Cloud services. Moreover, a wide range of open-source projects implement Cloud software stacks. However, as remarked in our state of the art analysis (2) none of the existing solutions provide all the functionality required by VPH-Share. Thus, Task 2.2 will not only deploy existing solutions but also develop custom modules that are necessary to fill the gap between the required functionality and the features provided by existing software systems.

4.2.1 Functionality

Describing the functionality of CEE requires us to determine who will use it. The classes of actors who will directly access different feature subsets provided by Task 2.2 are illustrated in Figure 7. Accordingly, actors of the subsystem can be classified as follows:

1. The Allocation Management Service, will access REST interfaces in order to:
 - a. Read the status of the Cloud infrastructure. This data will include standard load metrics for Atomic Service Instances (virtual machines hosting VPH-Share specific applications) and performance data. The former will consist of CPU usage, memory consumption, I/O operations and network transfers, where exposed by the applicable Cloud stack. The latter will comprise performance indicators such as the number of requests that can be served in a unit of time or the time required to process a single request;
 - b. Manage Atomic Service Instances. This will involve enacting a deployment plan that specifies actions such as creating new virtual machines from templates for application providers (see Section 4.1.7), saving a virtual machine as a new Atomic Service; starting Atomic Service Instances to provision the required functionality or stopping idle instances to lower costs. CEE must also be ready to provide a management API for private and public Clouds.
2. The Atomic Service Cloud Facade will access functionality provided by Atomic Service Instances. The Cloud Execution Environment will be responsible for hosting instances in



public and private Clouds. It will also be responsible for providing a well-known endpoint that will proxy and route requests to a dynamic pool of instances. This endpoint can be used optionally when direct access to a specific instance is not possible due to e.g. firewall restrictions.

3. The application providers will:
 - a. Use wrapping mechanisms enabling them to expose their applications as Atomic Services. CEE will facilitate publishing a REST or SOAP remote interface to remotely invoke applications deployed in the Cloud infrastructure. Additionally, it will ease the process of configuring security for applications;
 - b. Connect to virtual machines hosted in Cloud infrastructures in order to install and configure their applications. It is foreseen that console-based access over SSH and VNC connections will be available for application providers.

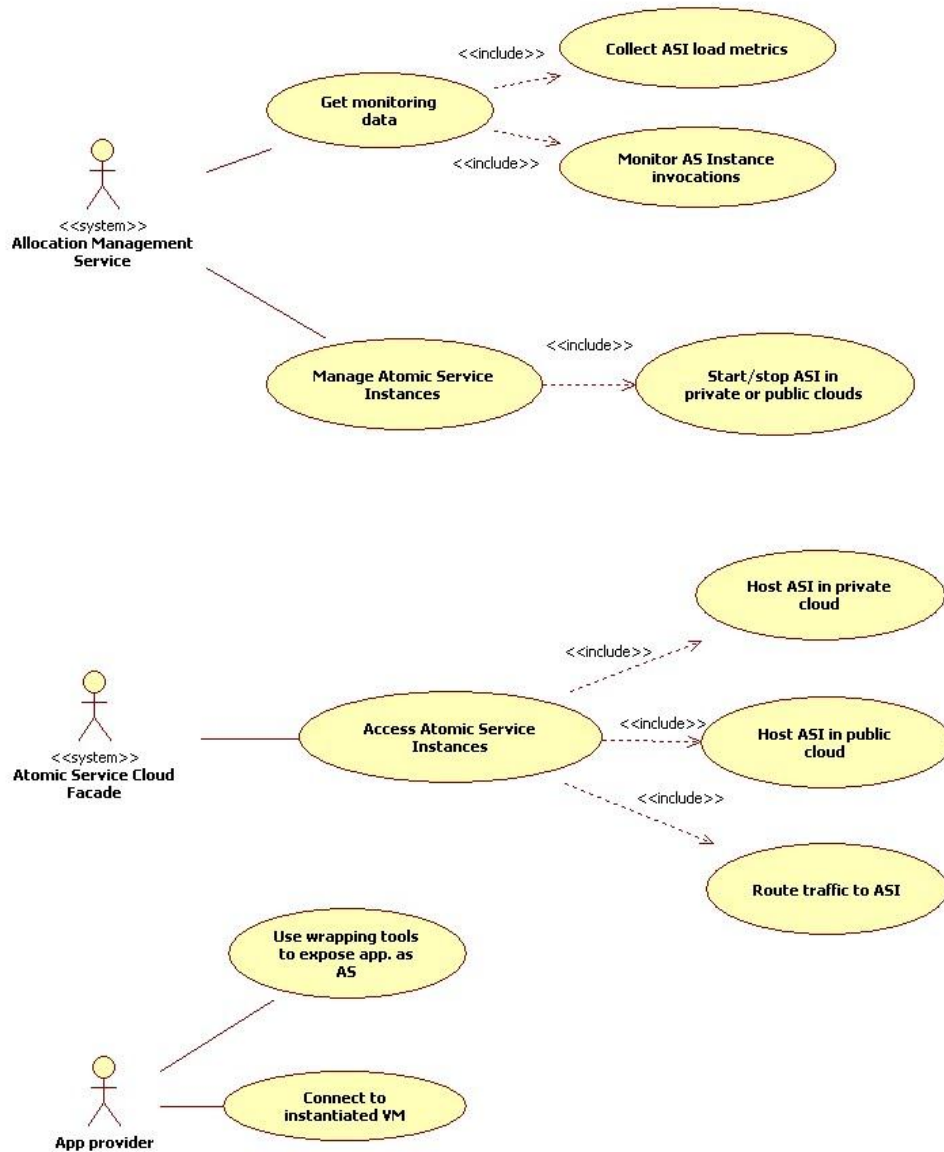


Figure 7: Actors of the Cloud Execution Environment, namely the Allocation Management Service, the Atomic Service Cloud Facade and the application provider. Features of the system that are not directly accessed by users, but are required to provide CEE functionality, are also presented.

4.2.2 Atomic Service Instance

The main building block of a VPH-Share workflow is the Atomic Service Instance. It is a virtual machine with preinstalled software, published as a SOAP or REST Web Service. An image of this VM (Atomic Service) needs to be stored in the VM repository and instantiated on demand when a workflow is started (and then shut down once the workflow finishes). We can distinguish two types of Atomic Services Instances: stateless instances (capable of being shared among workflows) and stateful instances, i.e. instances whose lifecycle consists of the following steps:

1. Configuring the application
2. Starting the application
3. Monitoring application execution status
4. Retrieving application results

The defining characteristic of stateful services is that at least one of the provided methods depends on other method(s). As a result, these services cannot be shared among workflows. Information on whether a service is stateless or stateful will be stored inside the Atmosphere Internal Registry (see Section 4.1.4).

In the scope of the Task 2.2 and Task 2.6 we plan to deliver libraries and tools that simplify the whole process of converting existing applications (in most cases command line applications) into Atomic Services. Section 3 presents the high level architecture of the atomic service. This architecture consists of three main layers:

- Security layer – a library (Apache module) integrated with security tools created in the scope of Task 2.6. The main responsibility of this layer is to ensure that every request that reaches lower layers is authenticated and authorised. This component will have only one implementation and will be generic for all Atomic Services.
- SOAP/REST Service layer – libraries that are able to expose the application as a SOAP or REST Service. Our aim is to avoid imposing limits on the number of libraries and programming languages that can be used here. As a result, the application developer responsible for wrapping applications into Atomic Services may choose the most suitable technology for each application.
- Wrapper layer – tools able to wrap existing command-line applications as libraries. The process of wrapping consists of several steps: at the beginning, the environment has to be configured in an appropriate way (e.g. the command-line application may require a configuration file); subsequently the application is executed and its results collected and forwarded to upper layers.

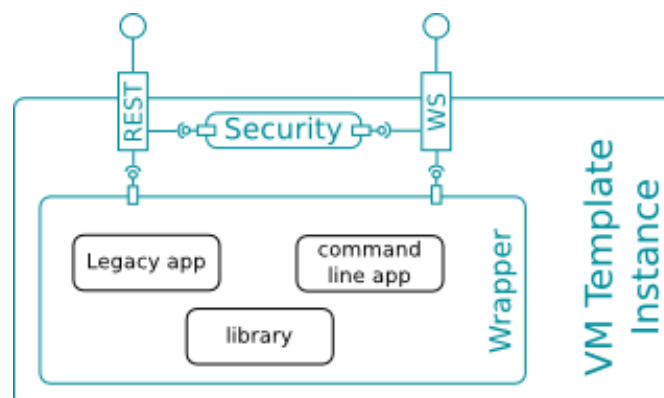


Figure 8: Structure of an Atomic Service Instance. The virtual machine (with a selected OS) hosts a VPH-Share application with a REST/SOAP interface exposed using wrapper mechanisms which involve a security module.

The process of wrapping existing applications into Atomic Services should be as simple as possible. To fulfil this requirement we plan to deliver preconfigured virtual machine images



that can be used as a starting point for creating new Atomic Services. These images will contain:

- A preinstalled operating system (e.g. Ubuntu, CentOS, Windows)
- Tools that allow Atmosphere to configure the installed software (e.g. security, monitoring layers) while booting up the virtual machine. We will have two types of configuration tools: tools delivered by cloud providers (e.g. generating a unique security identifier/key, enabling providers to log into the freshly spawned virtual machine) and tools created in the scope of WP2. The second group permits configuration of the Atomic Service Instance itself. This process consists of several steps: initially, the configuration for the specific Atomic Service Instance needs to be downloaded from the Atmosphere Internal Registry; subsequently, security attributes need to be applied in the security layer. This is followed by configuration of the monitoring system and, finally, a specific part of this configuration is applied to the wrapper and wrapped application itself.
- Installed security layer which will forward requests to the wrapper given appropriate credentials (a detailed description of this process can be found in the following part of this section)
- Sample command-line application (e.g. echo) wrapped as a SOAP or REST Service

4.2.3 Architecture

The Cloud Computing Environment will provide three distinctive types of features (see Section 4.2.1) and will therefore consist of several software components. An overview of its architecture is provided in Figure 9. Generally, CEE functionality can be divided into three categories:

- Standard solutions that will be installed, configured and maintained in order to provide an execution environment for Atomic Service Instances and build private Cloud infrastructures. These include:
 - Wrapping mechanisms (already described in Section 4.2.2)
 - Atomic Service Instance Proxy
 - Monitoring System
 - Private Cloud software stack
- Public Cloud providers
- Modules that will be developed within Task 2.2 that will control the aforementioned components in order to provide an efficient platform for hosting Atomic Service Instances:
 - Monitoring Controller
 - Atomic Service Instance Proxy Controller
 - Cloud Clients

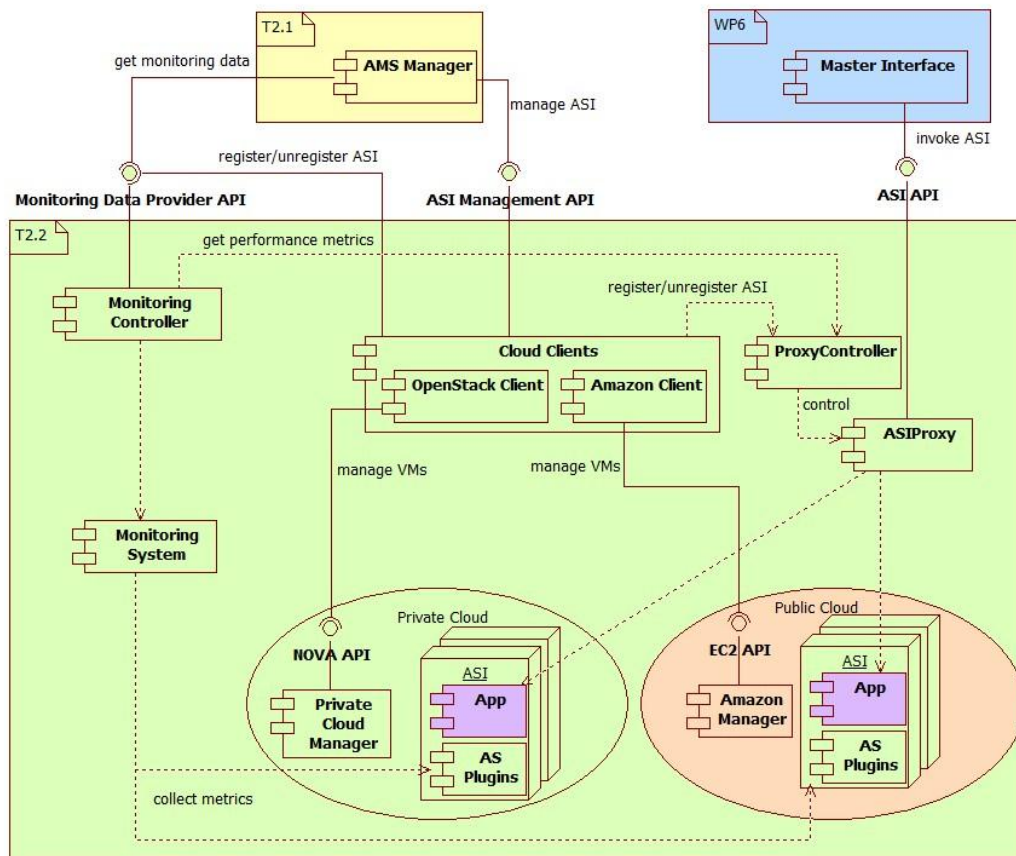


Figure 9: Cloud Execution Environment architecture overview. Components marked in light green are either developed within Task 2.2 or existing solutions that will be deployed and configured. Components that are marked in other colours are external to CEE but are significant to explain its architecture.

Wrapping mechanisms will facilitate exposing applications as Atomic Service Instances. These mechanisms have already been described in Section 4.2.2.

The Atomic Service Instance Proxy is a proxy that will enable transparent access to instances deployed in a dynamic pool of computing resources with IP addresses that are not known *a priori*. It will provide a proxy interface for instances under a well-known endpoint and route requests to an appropriate instance. Existing proxy servers will be deployed and configured. As standard proxy servers do not support dynamic pools of resources in an out-of-the-box fashion, an Atomic Service Instance Proxy Controller will be responsible for updating proxy configurations, reloading the server, and providing basic statistics on Atomic Service Instances on the basis of proxy server logs.

Similarly, the Monitoring System will be a standard solution for monitoring infrastructure and services, enhanced with a Monitoring Controller that will enable it to adapt to a dynamic environment. The Controller will parse monitoring logs and query the Proxy Controller for ASI performance data, and it will expose a single endpoint for exposing this data to AMS.



Cloud Clients will encapsulate client-side libraries that will handle interaction with managers of both public and private cloud infrastructures. The platform must be extensible with new clients for infrastructures that may emerge during the course of the VPH-Share project. Initially, cloud execution environment components of the Atmosphere platform will be used to manage two types of clouds – private (provided internally by Project partners) and public (provided by a selected commercial/scientific provider).

Use of both types of cloud systems is mandatory for several reasons. Private clouds are needed due to:

- the need to exercise full control over some critical data that is too sensitive to be processed in a public cloud environment
- cost issues for persistent instances: the cloud business model offers good cost effectiveness for relatively short-lived tasks by providing huge computational power without the need to invest in hardware; however, if we know that a given instance would be required for long periods of time, running it on a private infrastructure will likely prove cheaper.

At the same time there is also a need for access to public cloud infrastructures:

- to cover for on-demand traffic spikes by temporarily acquiring computational resources that do not exist in private clouds (or would require significant investment)
- to ensure limitless scalability of the proposed platform (to the extent that computing power and storage space can be acquired from public providers)
- to allow more manageable access to proprietary software such as MS Windows Server, whose license is provided as part of the service offered by Amazon, without the need to obtain separate vendor license agreements when provisioning services to third parties on custom infrastructure (such as SPLA, Service Provider License Agreement)

The internal architecture of public cloud infrastructures and commercial software stacks are typically not disclosed, with some notable exceptions (such as the use of open-source stacks by RackSpace). Such internal architecture details are beyond the scope of this document, as these are as varied as they are numerous and VPH-SHARE will only be a customer to these services. On the other hand, contributed hardware (provided by consortium partners) needs to be cloud-enabled. This section will describe how to accomplish this task.

We have decided to base our solution on an existing open-source cloud stack – OpenStack. Our choice was prompted by the following considerations (2):

- manageable and well documented deployment process
- rich features offered by the stack – including the ability to run instances, full network management including floating IP (ability to temporarily assign and reassign public IPs – similar to Amazon’s Elastic IP), Nova-Volumes allowing network-provisioned disk volumes to be mounted on instances (similar to Amazon EBS) etc.
- highly effective and manageable remote API and a large selection of implementing libraries

The overall architecture of an OpenStack-based private Cloud deployment is shown in Figure 10.

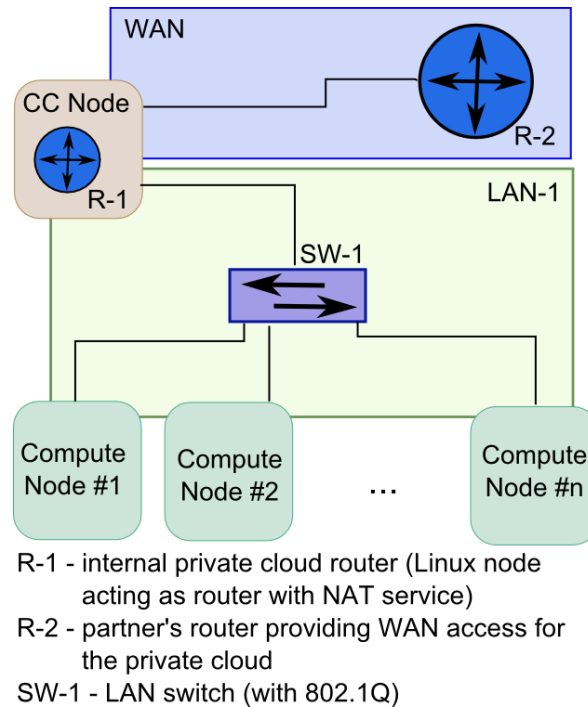


Figure 10: Architecture of a private Cloud deployment. Types of nodes (Cloud controller and compute) as well as network configuration are presented.

As shown in Figure 10 we plan to use two types of nodes: a single Cloud Controller Node (CC Node) as well as multiple identical Compute Nodes. The exact number of Compute Nodes depends on the partner and may be adjusted during the project to meet demands for resources.

The Cloud Controller will run all required Nova services except nova-compute (nova-api, nova-network, nova-scheduler and nova-volumes), the Glance Image Service and the required dependencies (MySQL server and RabbitMQ). It will act as an entry point into the cloud and provide general low-level management functionality (such as handling API requests and scheduling VMs to be run). Each Compute Node will run the nova-compute and nova-network services including the actual VMs via a standard virtualisation stack (KVM on Ubuntu with libvirt).

The cloud will employ two types of networks: an internal Local Area Network and an external Wide Area Network (connected to the Internet via a router – R-2 in the diagram). Only the Cloud Controller node will have direct access to both networks and, as such, it will act as gateway with NAT functionality (R-1) for the Compute Nodes. This will be achieved through standard Linux routing and filtering mechanisms, supporting outgoing connections (SNAT) as well as external IP addresses or TCP/UDP port forwarding for incoming connections to services running within the LAN (DNAT). Both networks will be Ethernet-based. The local network switch (SW-1) will support all OpenStack networking modes



enabling separation of nodes and VMs (cloud instances) network traffic (on the L2 level) with VLAN mechanisms (based on the 802.1Q standard).

4.2.4 Provided interfaces

Various components of the Cloud Execution Environment will expose a set of RESTful interfaces for the upper layers of the VPH-Share platform. For the AMS subsystem, there will be interfaces exposed by:

- the Cloud Client module, enabling management of ASIs hosted in public and private clouds
- the Monitoring Controller, exposing the status of infrastructure and ASI performance metrics

Every Atomic Service Instance will expose a Web Service interface enabling remote access to the application it hosts. This interface will be proxied by the Atomic Service Instance Proxy. In addition, SSH and VNC interfaces will be exposed to application providers, enabling them to install and configure applications.

All remote Web Service interfaces will be secured with an appropriate security mechanism provided by Task 2.6 (see Section 4.6 for details).

4.2.5 Dependencies

The Cloud Execution Environment does not invoke the functionality of any other subsystem of the Data and Compute Cloud Platform. It is the lowest layer of the Atmosphere system, although it does depend on certain existing solutions. These include:

- the monitoring system
- the atomic service instance proxy
- private Cloud software stacks
- commercial Cloud infrastructures

For a list specific tools that will be deployed please refer to Section 4.2.7.

4.2.6 Control flow

This section focuses on interactions between components required to support the use cases listed in Section 4.2.1. The simplest use case involves AMS obtaining the infrastructure status from the Monitoring Controller. In contrast, managing Atomic Service Instances is much more complex and will be discussed in more detail. Two specific scenarios can be distinguished in this context: creating a new Atomic Service and starting/stopping an Atomic Service Instance.

4.2.6.1 Creating a new Atomic Service

This use case is depicted in the UML sequence diagram in Figure 11. Numbers in parentheses indicate the relevant step in the diagram. It is worth noting that creation of instances may be



performed automatically, whenever the functionality of some Atomic Service is requested by WP6 tools (including workflow composition components) as well as manually, as demanded by system administrators (see Section 2 for a discussion of VPH-Share user roles).

- The process begins at the main entry point for the whole VPH-Share infrastructure, namely the Master Interface (web portal). The application provider logs into this portal and can then browse the list of available templates and choose the most appropriate VM template for a new Atomic Service.
- The user then requests creation of a new Atomic Service (1).
- The Master Interface responds by sending a request to the Allocation Management Service (AMS) (2), which contacts Cloud Client components in CEE to spawn new instance from a virtual machine template (3).
- An appropriate Cloud Manager is contacted (4) and a new virtual machine is spawned (5).
- IP address and credentials are returned by the Cloud Manager to Cloud Clients (6), and forwarded to the application provider through AMS and the Master Interface (7-9).
- The provider can now log into the virtual machine (using SSH, VNC or rdesktop) (10), install the application with all required tools (11), configure the application (12) and create start-up scripts (13). These scripts are responsible for configuring Atomic Service Instance software while booting up.
- The provider logs out (14) and saves the newly created Atomic Service (15). This request is propagated through AMS (16) to Cloud Clients (17).
- An appropriate Cloud Manager is contacted (18), the virtual machine is stopped (19) and converted to a template.
- An identifier of the new template is returned to the Cloud Client (20) and then to AMS (21) which registers the new Atomic Service (along with its configuration) in the Internal Registry.

4.2.6.2 *Invoking Atomic Service Instance*

Domain Scientists running workflows or executing a single service can use the Atomic Service Cloud Facade (part of the Master Interface) to execute running Atomic Service Instances. This process consists of several steps (see Figure 12):

- Initially, the request is sent from the Atomic Service Cloud Facade to the Atomic Service Instance Proxy (1). This request contains the name of the operation which has to be invoked, the list of required parameters and the security handler.
- The request is forwarded to the Security Facade (2) module, prepared by Task 2.6 and preinstalled on every Atomic Service Instance.
- The Security Facade communicates with the Security Framework (3 and 4) and checks if the request can be authenticated and authorised (5 and 6). If not, an exception is thrown (22 and 23), otherwise the request is forwarded to lower Atomic Service layers (7).
- The handle is also forwarded to lower layers to facilitate access to other VPH-Share secured services, e.g. databases.
- The wrapper prepares the application environment (8) and, if necessary, configures external services (9-14). Here the situation is similar: external components communicate with the security framework and check if the request contains appropriate credentials. If not, an exception is returned (15 and 16).



- Once configuration of the external components completes, the application can be executed (17) and the result returned to the Atomic Service Cloud Facade (18-21).

Separation of the security layer, which will be preinstalled on every Atomic Service virtual machine template, allows the developer to focus on the application functionality instead of security issues.

4.2.6.3 *Setting up an Atomic Service Instance*

The control flow for this use case is presented in Figure 13.

- The process begins with a request from the AMS to the Cloud Clients, asking them to deploy an Atomic Service Instance (1).
- Next, a request to start a virtual machine in the Cloud infrastructure is sent to an appropriate Cloud Manager (2) that instantiates a new VM (3). During this step the Cloud Manager passes the configuration ID which is later used to download Atomic Service Instance configuration from the Atmosphere Internal Registry.
- The IP address and credentials are returned to the Cloud Client module (4) and then to AMS (5).
- The Cloud Client component registers the instance with the Atomic Service Proxy Controller (6) which, in turn, updates the configuration of the Atomic Service Instance Proxy (7). This is required to make the ASI accessible via a proxy endpoint with a well-known address.
- Similarly, the Cloud Client component registers the ASI with the Monitoring Controller (8), updating the configuration of the Monitoring System (9).
- While booting, the ASI queries the Internal Registry for its configuration (10, 11) and then configures its own security (12) and application (13) settings.
- Finally, ASI opens a REST interface (14) to facilitate contact with external clients.

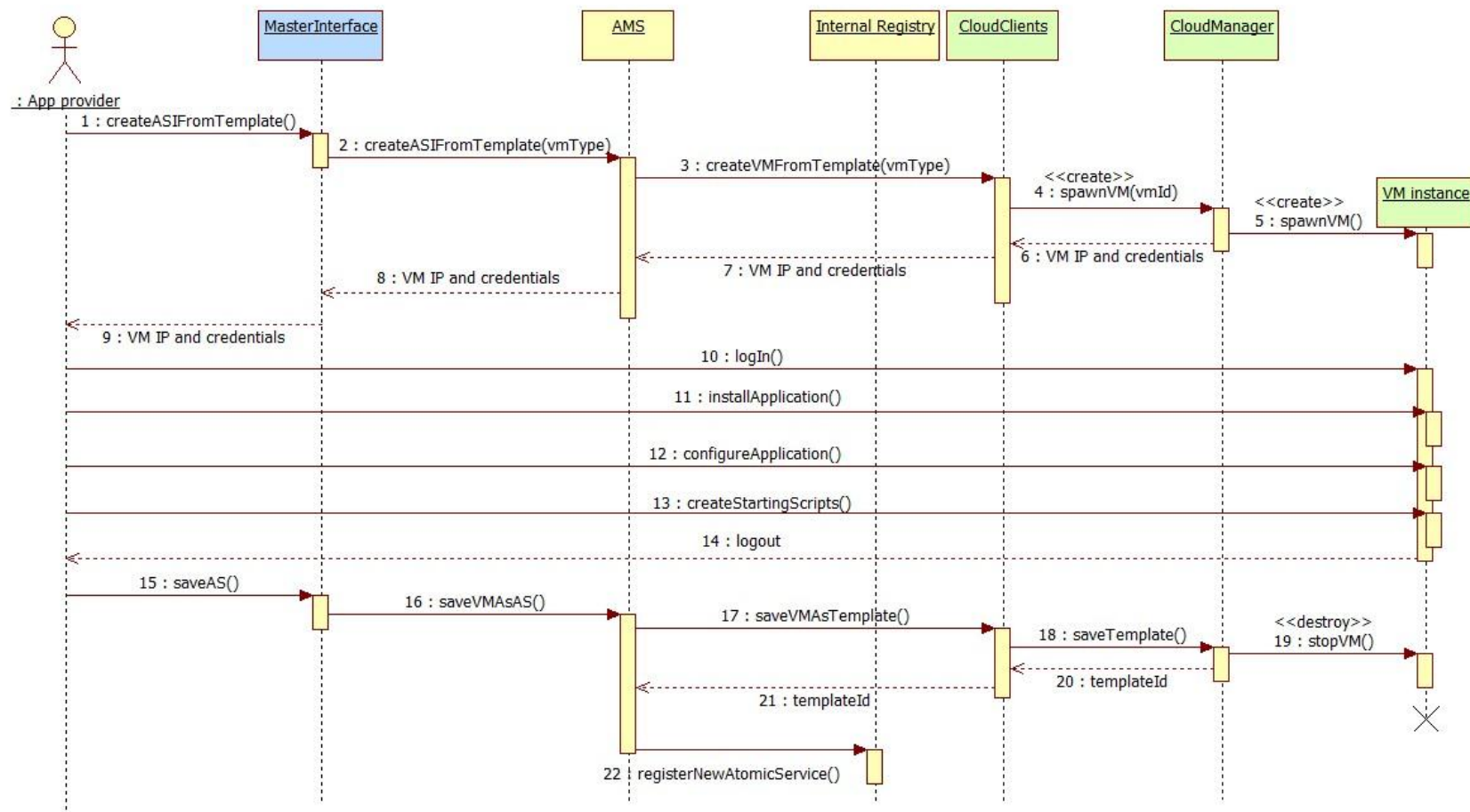


Figure 11: Creating a new Atomic Service. Interaction of the end user (application provider) with the infrastructure (Cloud Manager) is presented.

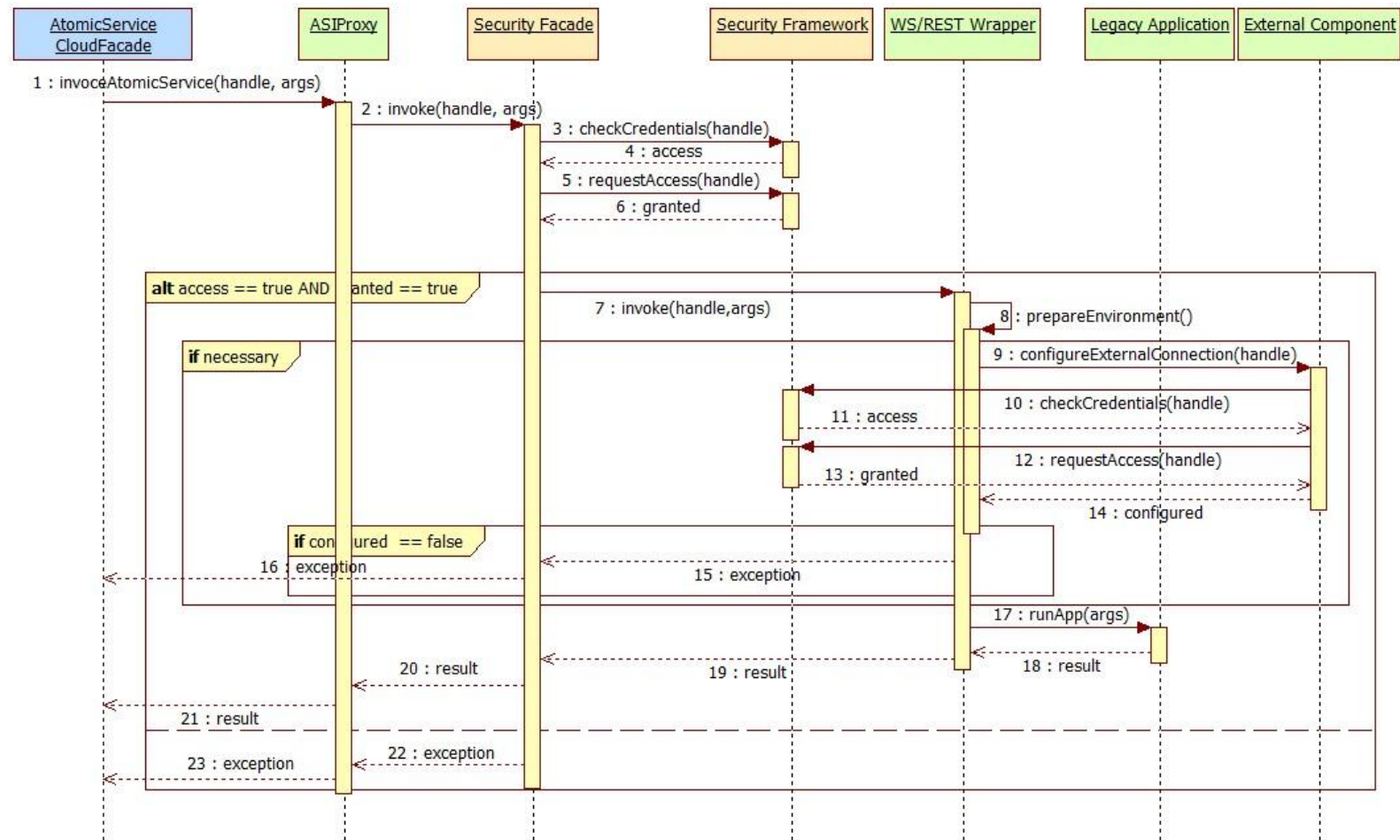


Figure 12: Invocation of an Atomic Service Instance.

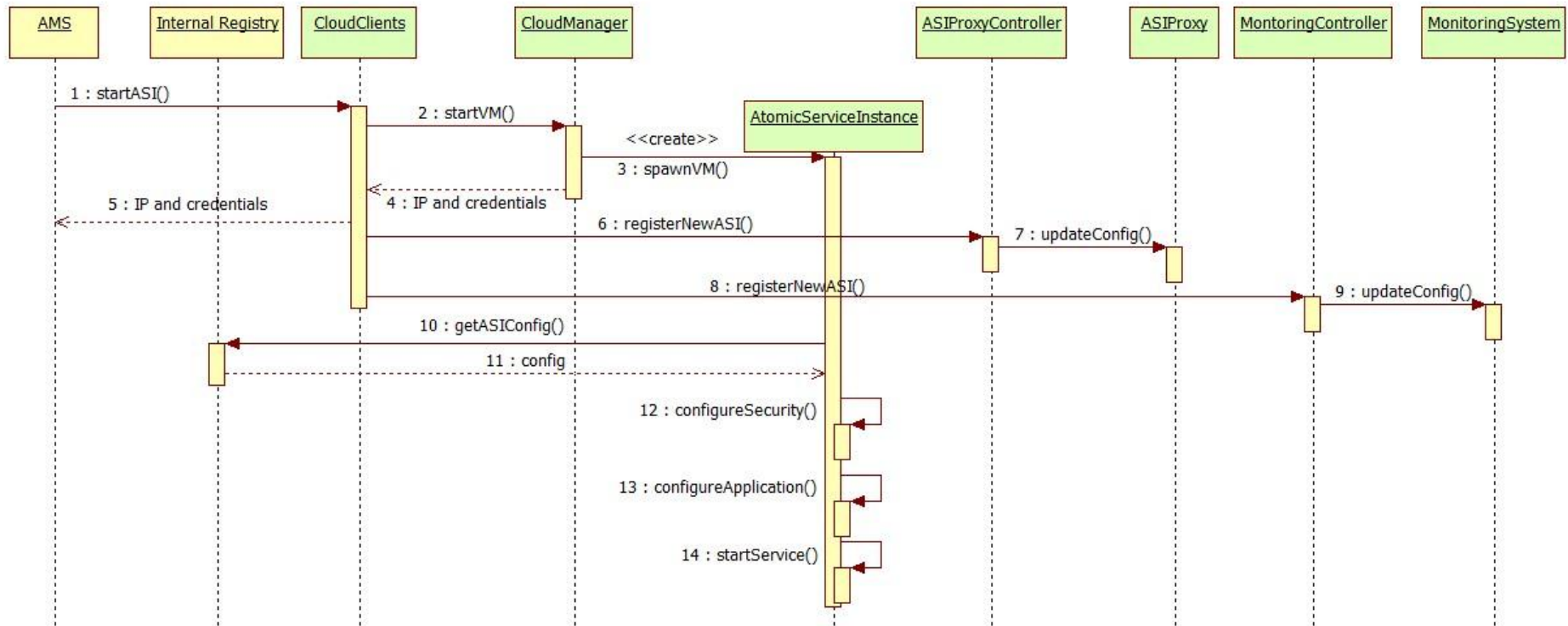


Figure 13: Control flow involved in deploying an Atomic Service Instance.



4.2.7 *Candidate technologies*

This section lists the technologies which will be utilised when implementing and deploying Task 4.2 components.

4.2.7.1 *Atomic Service Instance*

The architecture of the Atomic Service Instance can be split into four layers:

1. Operating system
2. Security layer
3. Web service layer
4. Wrapper and application

In the scope of the VPH-Share project we will deliver a set of virtual machine templates with preinstalled components. For Unix-like deployments we initially aim to deliver the latest stable Ubuntu release (15) (other distributions will be added as necessary). For Atomic Services which require Windows-based images we are going to use templates created by the Amazon Cloud provider. We cannot support Windows-based images on private Cloud platforms due to licensing issues.

In the security layer we plan to deliver a dedicated Apache module (16), which will work as a proxy for all messages sent to the Atomic Service Instance, and decide if a request should be forwarded to lower layers or not.

In the third and fourth layers our goal is to tailor the AS wrapping process to the requirements of a specific application rather than providing a fixed, generic solution. For example, if the application is created as a command-line tool then SoapLab2 (17) can be utilised, but if it is a library created e.g. in Python, then the appropriate Python library may be used to wrap it as a SOAP or REST service.

4.2.7.2 *Atomic Service Instance Proxy*

A lightweight and reliable web server called Nginx (18) will be deployed and configured. Its configuration can be easily updated and the server itself can be reloaded very quickly, which is an important benefit. It can also provide meaningful information in its logs.

4.2.7.3 *Monitoring*

Monitoring will serve a dual purpose – track the availability of all running instances and services, and collect data for AS deployment optimisation. Our monitoring tool of choice is Nagios (19), for the following reasons:

- it is mature and has a proven track record (over 12 years of active development)
- it is commonly applied in science and industry
- it is highly configurable
- it offers great flexibility and extendibility through probes, supporting standard and non-standard monitoring (generic and custom probes)
- it uses a lightweight daemon on the monitored nodes



We plan to deploy key Nagios components (server and GUI) on a dedicated VM, and use it to execute remote checks on ASIs using an NRPE mechanism. To allow this, we will need to predeploy the `nrpe_server` daemon and a set of required plugins on all AS Templates. However, due to the lightweight nature of those components, their presence will not affect the ASI's performance significantly.

We intend to collect standard statistical data such as CPU load, system physical memory/swap/hard drive usage and network utilisation. Of course, it would be equally possible to check additional parameters if needed (such as availability of services).

4.2.7.4 *Controllers*

To integrate monitoring and proxy subsystems with the rest of the infrastructure we plan to develop two dedicated tools exposing a RESTful API. These tools will permit updates of ASI lists in configurations of monitoring and proxy servers, while also exposing metrics gathered through Nagios and Nginx. Controllers should be lightweight and simple tools – thus, scripting language seems to be a reasonable choice. A promising implementation language seems to be Ruby, combined with the Sinatra REST framework (13) and deployed using the Phusion Passenger (14) web server module.

4.2.7.5 *Cloud Clients*

Both OpenStack and Amazon provide a RESTful remote API. There are also numerous libraries for accessing those (and other) Clouds. Following analysis of those libraries we have chosen a specific software package – JClouds (20). Our decision was prompted by the following JClouds features:

- Support for Clouds that we plan to use (OpenStack and AWS) as well as numerous other services (like GoGrid and Azure) in addition to private Cloud stacks (Eucalyptus, VCloud) allowing future extensions
- Technological compatibility with the rest of our platform (support for Java and OSGi)
- Active support and ongoing development efforts on the part of software vendors

The library will be deployed as an OSGi bundle in the Karaf container to provide smooth integration with the rest of Atmosphere.

4.2.7.6 *Cloud software stacks/providers*

OpenStack has been selected as the private Cloud software stack of choice by the VPH-Share consortium. The reasons have been explained in Section 4.2.3 (see also (2) for an in-depth description of the features offered by popular private Cloud stacks).

4.3 Access to high-performance computing environments

4.3.1 *Component description*

To provide a high-performance execution environment, Task 2.3 will develop and provide two components for VPH-Share work package 2: The Application Hosting Environment (21) (AHE) and the Audited Credential Delegation (22) (ACD) software. The PRACE (23) grid



infrastructure will be provided through VPH-NoE to ensure that the requirements of providing the grid infrastructure to the VPH-Share project as set out in Task 2.3 are met. AHE/ACD will be used to bridge the Cloud and grid infrastructure to ensure seamless transition for users or workflows who involve more significant computation resource requirements which the Cloud platform cannot provide. Furthermore, Task 2.3 will ensure that the VPH-Share software and security framework is integrated with both AHE/ACD as well as the grid infrastructure. This includes the GridSpace (24) workflow engine developed in ViroLab in FP6.

AHE is a lightweight service-oriented middleware suite which virtualises applications installed on the grid. AHE will expose the applications as RESTful web services, removing the need for the end user to interact with the complex grid middleware. An expert/admin user is required to set up the application on the grid and configure the AHE (only once) to ensure that application users can use the application.

The ACD component simplifies the security management of grid infrastructure. It provides a holistic virtual organisation (VO) tool that handles certificate setup and user management on a per-VO basis. ACD is able to hide the complexity of certificate usage allowing end users to access the system through a username/password combination, or through other security mechanisms such as shibboleth. A virtual organisation can use one certificate for multiple users by generating proxy certificates and map these proxy certificates to each user. This allows all actions by the users to be authenticated, authorised and audited. ACD will be modified to ensure that they are compatible with the VPH-Share XACML attribute-based authorisation mechanism as well as the general VPH-Share security framework.

To ensure that ACD/AHE performs as described in the description of work, we need to ensure that VPH-Share software is compatible with the grid infrastructure. To achieve this, applications which require grid resources will have to be identified and ensure that they are optimised and installable on the grid infrastructure. This is accomplished by the expert or administrator user. Furthermore, AHE is required to work with GridSpace developed in ViroLab. This will be achieved by way of a RESTful web service.

4.3.2 How HPC fits into the overall architecture of WP2 and VPH-Share

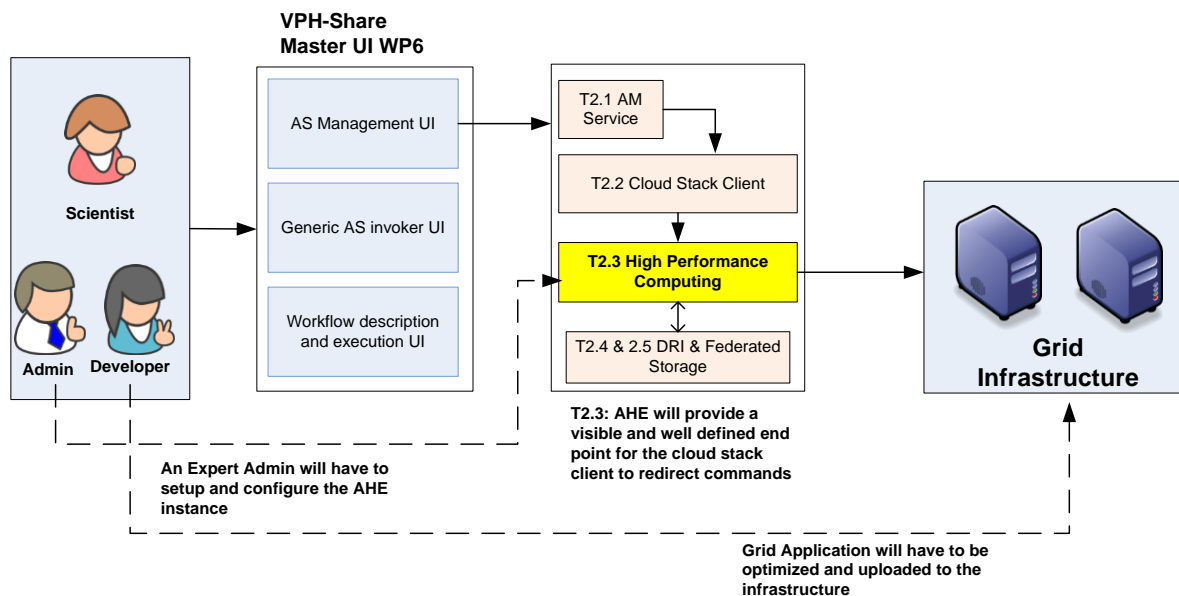


Figure 14: VPH-Share Overview. The main aim of Task 2.3 is to provide access to grid infrastructure and tools to the VPH-Share infrastructure.

In terms of the overall VPH-Share “infostructure”, as seen in Figure 14, the AHE/ACD components developed in Task 2.3 will be exposed to the end users through the master user interface developed in WP6. These components are the ACD security application and the AHE middleware. The master user interface will send commands to the Cloud job resource manager which will decide if operations will be redirected to the AHE/ACD and the underlying grid infrastructure. The main purpose of this is to provide the user with a seamless transition between the Cloud and grid infrastructure. Most API calls will be directed to the Cloud job resource manager. If resource demands exceed the Cloud capacity, the command will be redirected to the AHE for execution on the grid infrastructure. This approach hides the complexity of the underlying infrastructure, allowing the job resource manager to control which resources will be provided to the user. AHE/ACD can also be accessed directly, if required by the end user, using a RESTful web service.

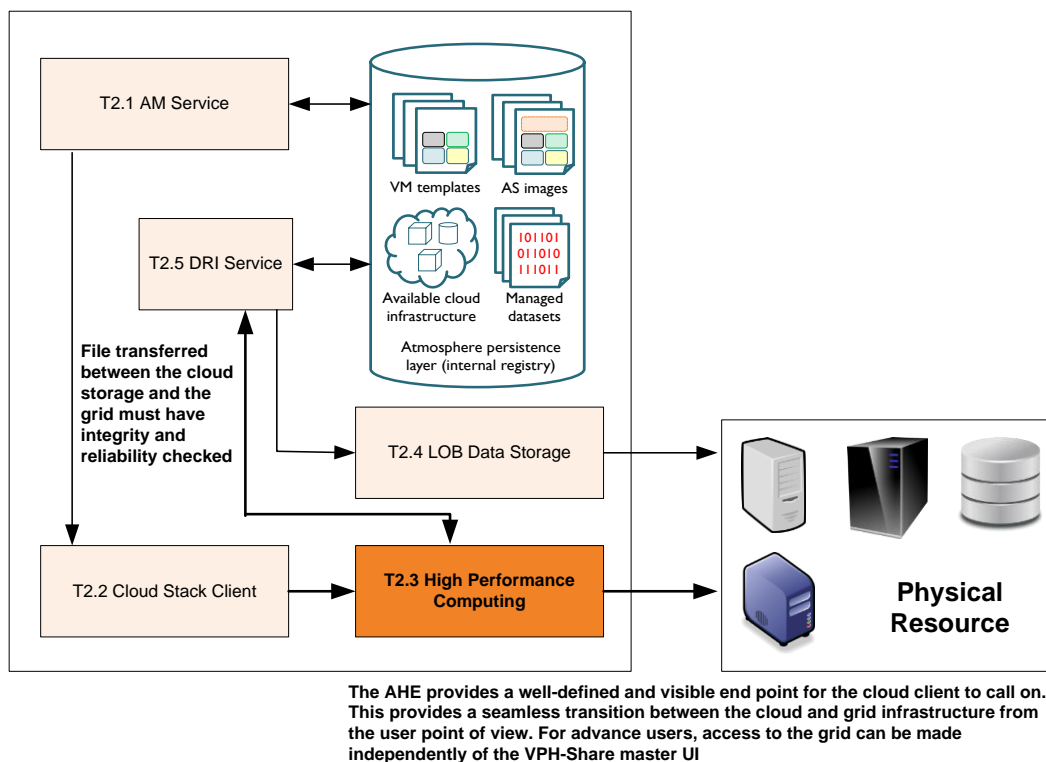


Figure 15: An overview of Task 2.3 within Work Package 2.

Within the WP2 as seen in Figure 15, the AHE/ACD must provide a visible and well-defined endpoint so that it can interact with the VPH-Share Cloud platform. AHE and ACD will be deployed as separate web applications on a J2EE-compliant application server. The AHE/ACD applications will integrate the VPH-Share security framework and utilise Cloud storage components. This will allow valid VPH-Share users to be authenticated and authorised by the ACD and allow AHE to retrieve data from the Cloud data storage and transfer it to the grid middleware, and copy results back to the storage infrastructure if required.

4.3.3 Detailed Design

4.3.3.1 Application Hosting Environment – requirements

As part of Task 2.3, AHE is expected to provide the following features:

- virtualise applications by exposing the grid infrastructure as a RESTful Web service
- provide a VPH-Share Cloud compatible RESTful API
- provide a well-defined and visible endpoint for AHE as well as a unique well-defined endpoint for each virtual application created (i.e. <http://vph-share.css.ucl.ac.uk/ah/as1/command>)
- be able to use the file storage system developed in VPH-Share
- integrate with the GridSpace 2 workflow tool. This is achieved by using the RESTful web service implemented in this project
- AHE will be redeveloped in Java to ensure better usability and reliability

4.3.3.2 AHE Components

The AHE core server consists of several components; these includes the AHE runtime module, the AHE engine module, the AHE API module, the AHE connector module, the AHE storage module, the AHE security module and extension points. A web client will be developed to provide web-based configuration capabilities for the AHE server. However, it is expected that most commands will be issued from the VPH-Share master interface.

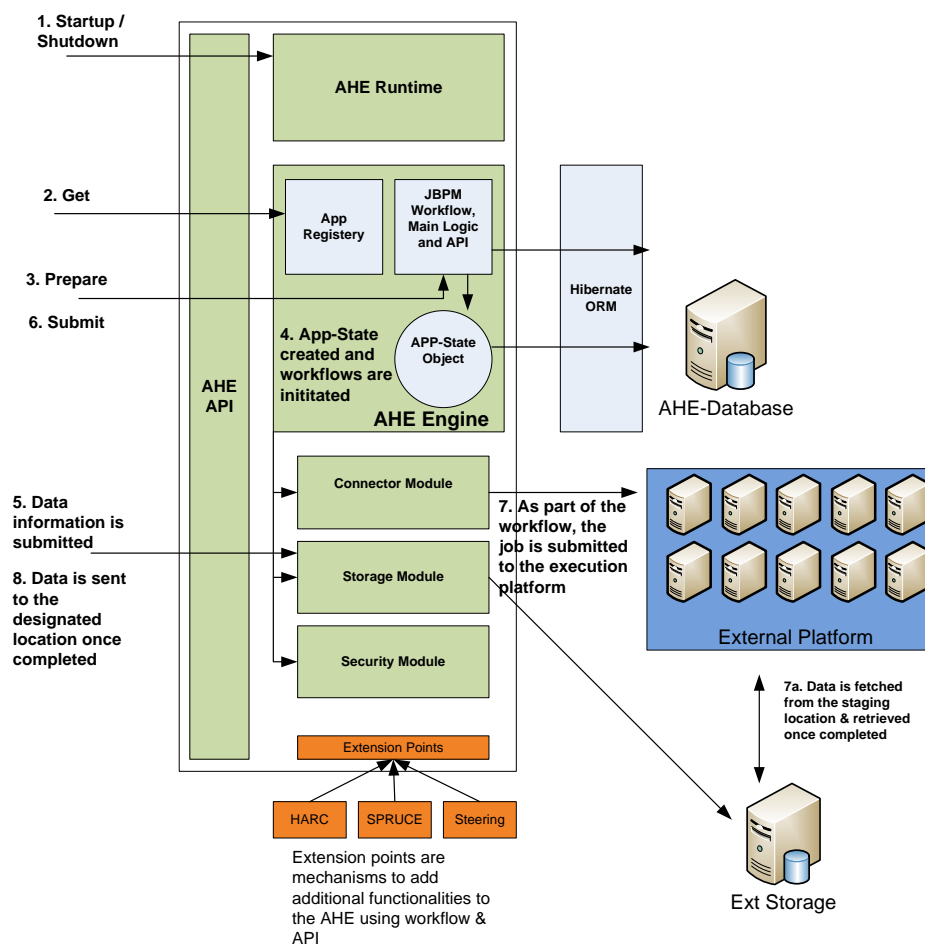


Figure 16: A typical AHE workflow

A typical AHE user case scenario can be seen in Figure 16.

1. The Runtime initialises all components, populates the internal data structures and ensures that the DB data is synced with the AHE data structures.
2. The user interfaces the app registry to see which applications are available.



3. A prepare command is issued, telling the AHE Engine to create a persistent APP-State (application state) Object which will keep track of the status and state of an executing application. This also initiates the AHE workflow process.
4. This APP-State Object is persistent and stored in a common database using an ORM (Object Relational Mapping) which decouples the type of database we can use between different installations.
 - a. The APP-State Object is associated with user/group and has a unique id.
 - b. Active APP-State data/objects are held in a registry and queried by the AHE engine, which determines when and how they can be run, and also when data can be checked in or retrieved.
 - c. Once completed, the APP-State object is set to inactive and removed from the active queue.
5. The file is staged. The server takes note of the location and transfer protocol and passes this information to the connector module so the job manager knows how to fetch it and where to put it back.
6. The Submit command is issued.
7. AHE workflow uses both the JBoss Java Business Process Management suite (JBPM) (25) and the Quartz scheduler (26). This allows complex workflows, supporting asynchronous tasks as well as multithread/concurrency support and human tasks to be modelled. The AHE engine deals with the security interface requirements when submitting the task to an external execution platform. It also polls the external platform (if it is configured to do so) and retrieves the data upon task completion. JBPM allows additional features to be implemented through the creation or modification of more complex workflows between different internal or external systems. JBPM is persistent, with all events logged. If the server crashes all the information and workflow state can be retrieved from the database and the application re-initialised.
8. Once completed, the data is retrieved and sent to a scratch disk (temporary file storage) or redirected to an external source, allowing the user to fetch the result.

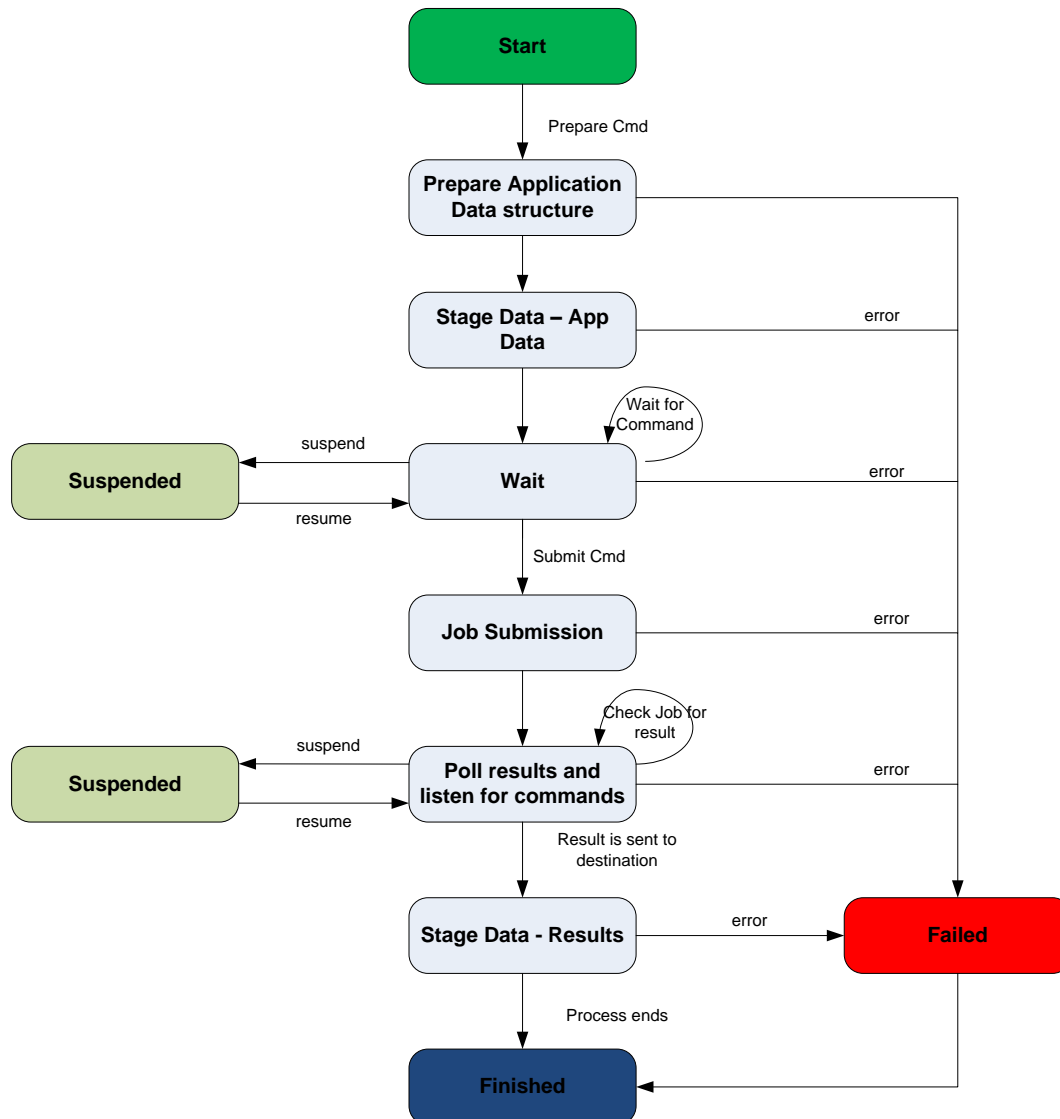


Figure 17: The AHE Job lifecycle state diagram.

The AHE job lifecycle can be described as seen in Figure 17. The basic job submission process proceeds through a number of different stages. The process starts when a Prepare commands is received by AHE. This first stage includes preparing data structures and setting extra job submission information, including where to stage the initial and final data. Once this is completed, a job can be submitted to the execution platform when the submit command is issued. Once the job has been submitted to an external execution platform, AHE goes into a polling state, checking regularly for the completion of the job. When the job completes, output data can be retrieved, the requestor notified (using automatic e-mail or other notification tools) and the job submission process comes to an end. This workflow is modelled and executed using the JBoss jISEBPM workflow library and the states can be mapped to the OGSA-BES (27) specification.

The following sections will describe the key features, classes and external packages used by each AHE module.

4.3.3.3 AHE Runtime

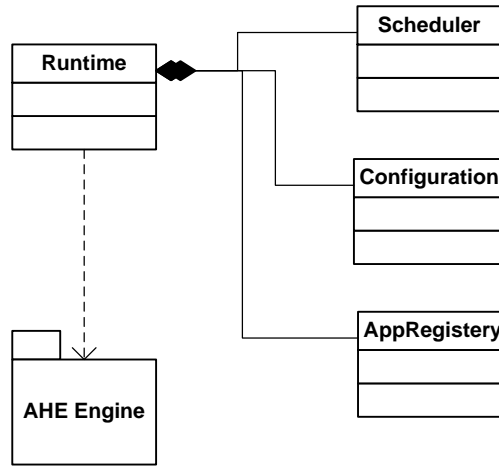


Figure 18: AHE Runtime module UML diagram

The runtime module is responsible for starting up/shutting down the server, initializing all components, ensuring that the proper user configuration has been applied, maintaining the internal data structures and core components and making sure that internal data structures, such as job the scheduler and the application registry, are in sync with the database as seen in the UML diagram in Figure 18. This module provides the following features:

- initialise all data structures and components
- provide an API for all internal data structures and databases, including AHE configuration, app registry and job scheduler
- ensure that all internal data structures are synchronised with the persistent data sources

4.3.3.4 AHE Engine

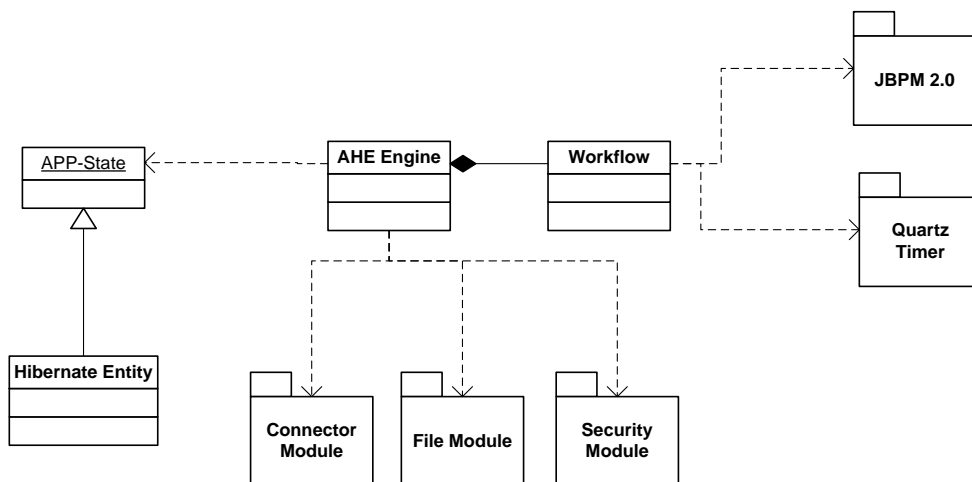


Figure 19: AHE Engine module UML diagram

The AHE Engine contains all the logic and essential functions which implement the core features related to providing virtualisation through web services, as seen in the UML diagram in Figure 19. These include maintaining and running the JBPM workflow engine and workflows for each application instance. It also provides an API for creating an APP-State Object which represents an instance of a virtualised application. The APP-State Object is fed through the JBPM workflow which describes how the data and application is processed.

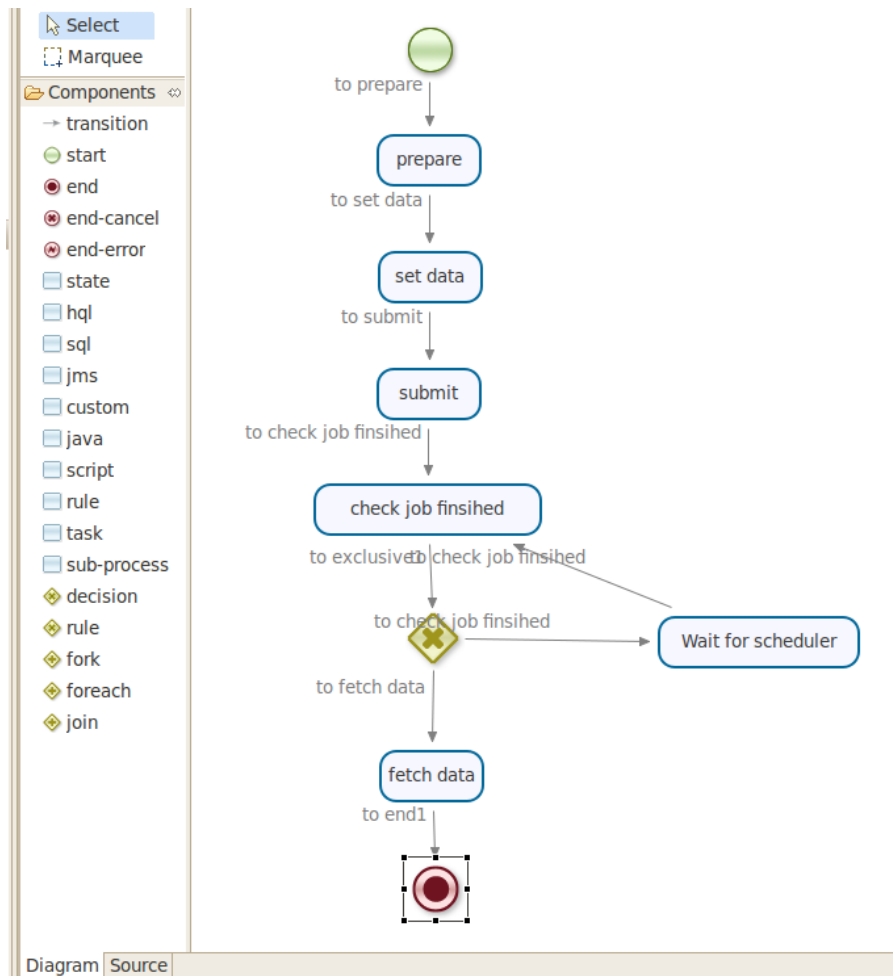


Figure 20: A Simple JBPM workflow document example created using the Eclipse JBPM editor. JBPM supports complex processes which include human interaction, event handling as well as rules.

A JBPM workflow is described using the Business Process Modelling Notation 2.0 (28) (BPMN) specification, as seen in Figure 20. It calls on specific Java classes, scripts or Drool rules (29) to perform certain functions. JBPM supports complex processes, involving human interaction, Drool framework rules and event handling. It also provides a set of powerful process management tools. This allows new complex workflows and features to be quickly introduced in AHE.

- provides an API for AHE logic and essential functions
- maintains the JBPM workflow engine
- maintains JBPM workflow documents which specify how jobs can be executed and what routines to call

- creates and handle multithreaded concurrent workflows
- maintains the Job scheduler and Quartz timer to check task status and launch new jobs
- generates an APP-State object which describes an instance of a virtualised application
- ensures the persistence of APP-State objects with Hibernate

4.3.3.5 AHE Connector Module

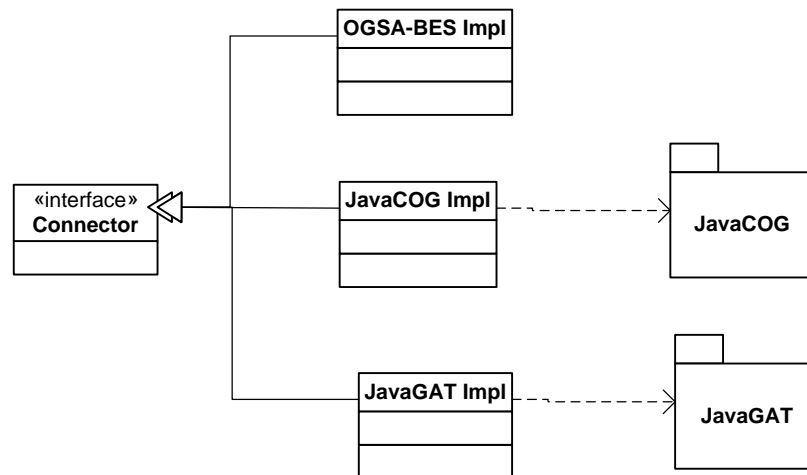


Figure 21: AHE Connector Module UML diagram

The connector modules provide a set of classes or external jars that will allow AHE to connect to external execution platforms. The connector module provides a generic Java interface (using adapter pattern or manual dependency injection) where adapters for external job managers will have to be written, as seen in the UML diagram in Figure 21. This Java interface will be used by AHE-Core to call and execute these functions, ensuring a loose-coupling relationship between AHE and external libraries. The AHE connector module will support Globus, GridSAM, OGSA-BES and other job managers using the JavaCOG and JavaGAT library, in addition to internally developed libraries. It fulfills the following functions:

- provides an interface and implementation classes for external job manager submission
- provides implementation for an OGSA-BES-based manager
- uses JavaCOG and JavaGAT to access external platforms such as Globus, GridSAM etc.

4.3.3.6 AHE API Module

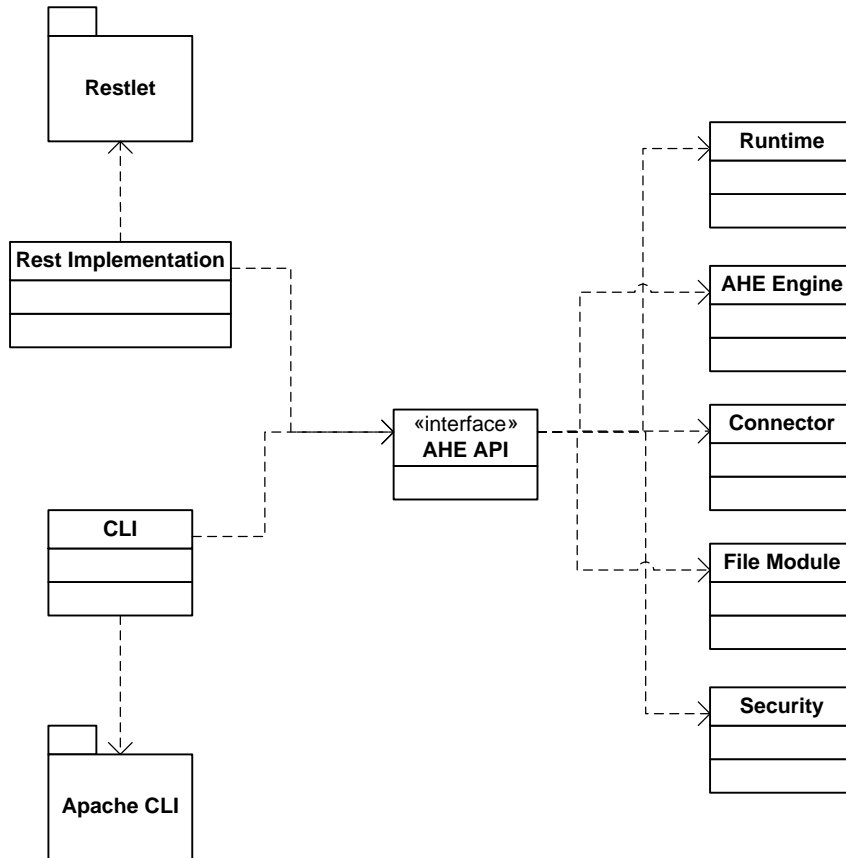


Figure 22: AHE API module UML diagram

This module contains all the code which provides APIs for external access, including the RESTful component required to access the AHE-Core Server as well as command-line access (Apache Common CLI) and the general AHE Java API (facade pattern). It provides a mid-to-high level API, allowing AHE features to be extended more easily, as seen in the UML diagram in Figure 22. The API module:

- provides a mid-to-high level RESTful Java API for the AHE-Core Server
- enables command-line access (Apache CLI)

4.3.3.7 Authentication Module

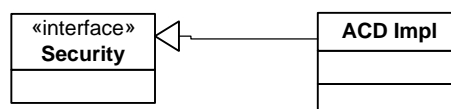


Figure 23: AHE security module UML diagram

This module provides an abstract interface enabling classes to implement access to the ACD security module, as seen in the UML diagram in Figure 23. The interface will provide a set of

basic functions to check the identity of the user and establish the actions which they are allowed to perform:

- 🌐 user access and control API
- 🌐 ACD interface

4.3.3.8 AHE Storage Module

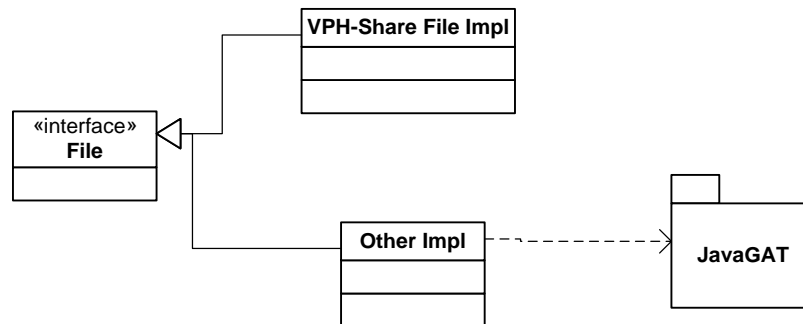


Figure 24: AHE storage module UML diagram

Ideally, files will be transferred directly from user-specified locations to the grid infrastructure. Once a job is completed, a command will be issued to the grid infrastructure telling it to transfer the file back to a location specified by the user. The general class structure of the storage module can be seen in the UML diagram in Figure 24. The VPH-Share implementation will contain code which specifies how to communicate with external file transport mechanisms. JavaGAT will be used to provide basic file transfer capabilities such as WebDAV (30) and gridFTP (31). The storage module provides an interface and API for file management functions, i.e. determining where a given file is located and where to transfer it.

4.3.3.9 AHE Extension Points

New features can be added to the AHE server by creating or modifying JBPM workflows and extending the AHE Engine API. In this way, extensions such as HARC (32), SPRUCE (33) or RealityGrid Steering (34) can be introduced. Note that all authentication needs to be carried out through the authentication module mentioned in Section 4.3.3.7.

4.3.4 Audited Credential Delegation Security Component

4.3.4.1 Requirements

The Audited Credential Delegation (ACD) application will be responsible for handling security in Task 2.3. ACD will incorporate the VPH-Share authentication mechanism; providing a mapping for user authorisation using attributes to determine which actions a given user may perform on the HPC. It should be noted that grid infrastructure security is handled and determined by the infrastructure provider and ACD/AHE security policy will have to abide by those conditions regarding the installation and operation of applications as well as transferring data into and out of the grid infrastructure.

ACD authentication and authorisation mechanisms are loosely coupled, ensuring added flexibility in the VPH-Share project where ACD authentication can be controlled or provided by the VPH-Share security framework. To use an external authentication mechanism, Shibboleth security will be implemented in ACD. A mapping between VPH-Share users will have to be created within ACD, which will map VPH-Share attributes to ACD user roles.

The ACD is expected to:

- provide a holistic virtual organisation security solution, allowing tighter control of user action as well as identity management
- handle Grid Security Certificate management as well as generation and management of temporary proxy certificates
- handle user authentication and authorisation
- log all users actions within each Virtual Organisation

4.3.5 Components

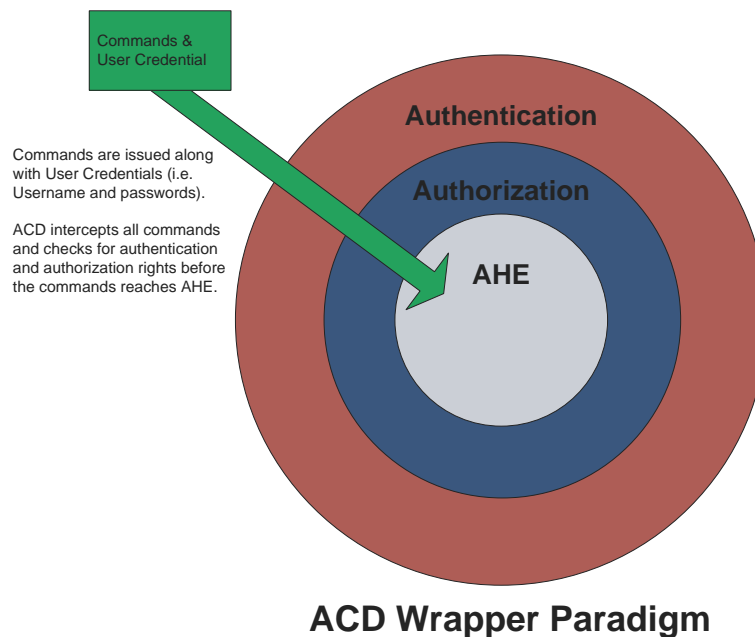


Figure 25: ACD wrapper paradigm

ACD is based on the concept of wrappers, as seen in Figure 25. A wrapper is a connector between a component and the outside world. Each task performed by a user is intercepted by each layer of the security wrapper to establish the identity of the requestor, check whether they are authorised to perform the given task and record the result. ACD is designed as a Web Service, providing an interface compliant with the Web Service Description Language, SOAP, WS-Policy and WS-Security.

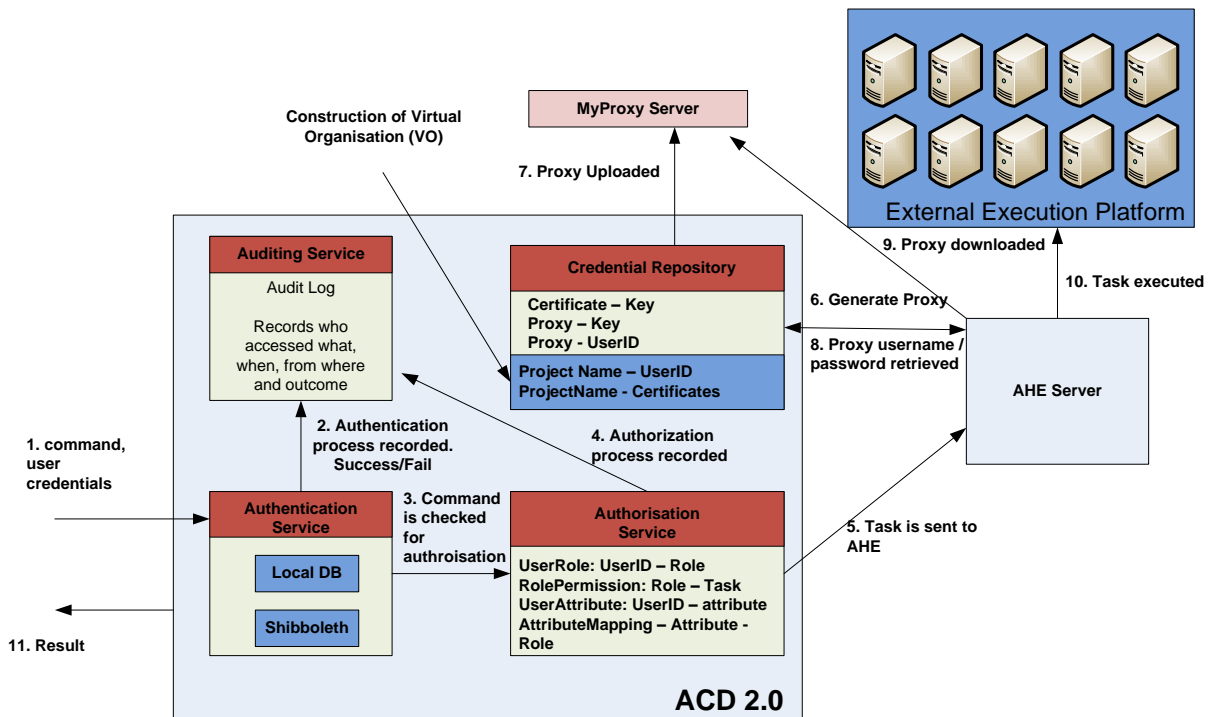


Figure 26: A typical ACD workflow. All commands are intercepted by ACD and checked before being sent to AHE.

As seen in Figure 26, a typical ACD workflow can be described as follows:

1. The User/VPH-Share submits a command with credentials (token, username/password pair etc.)
2. An authentication attempt is recorded by an auditing service.
3. ACD checks whether the command is authorised given the attributes mapped to the user.
4. The authorisation check request is logged, along with its result.
5. If authorisation succeeds, the command is passed to AHE.
6. ACD picks the certificate associated with the VO and checks if the user is assigned to this VO. If so, a proxy certificate is assigned.
7. The proxy certificate uploaded to a MyProxy server.
8. ACD sends a randomly generated username/password pair (required to access MyProxy) to AHE.
9. AHE downloads the session proxy using the supplied username and password.
10. The command or task is sent to the grid along with the proxy certificate.
11. The user retrieves results.

4.3.5.1 Authentication Service

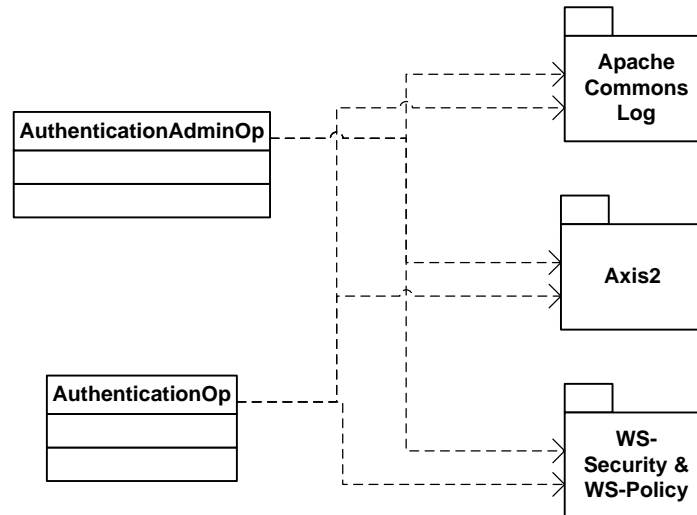


Figure 27: ACD Authentication Service UML diagram

One of the main objectives is to ensure that the end user does not need to directly handle the certificate. The current implementation supports a username-password combination. By default, to be authenticated, a user has to provide a username and password using the OWASP (Open Web Application Security Project) best security practices including strong passwords with length and symbol constraints.

Under the VPH-Share security framework, user authentication can be provided externally. The Shibboleth security mechanism will be used to authenticate the user between both systems. For this to work, mappings between the VPH-Share users/attributes and ACD roles will have to be created and maintained within ACD.

The authentication service consists of two classes, **AuthenticationAdminOp** which provides secure administrative authentication functions and **AuthenticationOp** which provides a function for authenticating normal users. These classes use the Apache common log, axis2 for web services and WS-Security and WS-policy libraries, as seen in the UML diagram in Figure 27.

4.3.5.2 Authorisation Service

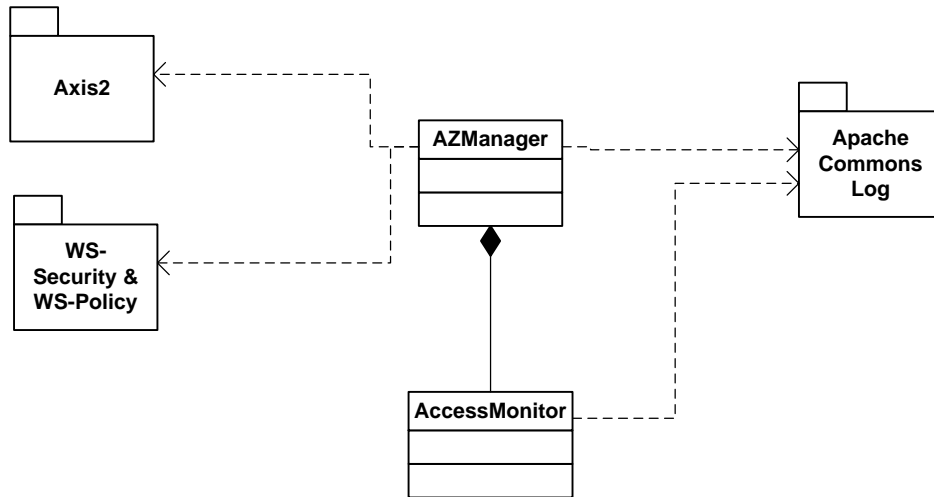


Figure 28: ACD Authorisation Service UML diagram

This component controls all actions performed in the VO. It applies a parameterised role-based access control model which assigns permissions to roles. The VO policy designer associates each user in the VO with the role that best describes their functions. The tasks and permissions assigned to the role are drawn from VO features. It should be noted that allowing the user to perform a certain task ultimately lies with the grid resource provider who has the final authority. In the context of AHE, two VPH-Share user roles are relevant here: admin and scientist.

The VPH-Share security framework is expected to adopt an attribute-based model. A VPH-Share attribute-to-ACD role mapping will be implemented, enabling VO policy designers to associate users with AHE roles either as an admin or a scientist. The **AZManager** class is responsible for checking the user's authorisation using the **AccessMonitor** class, as seen in the UML diagram in Figure 28.

4.3.5.3 Credential Repository

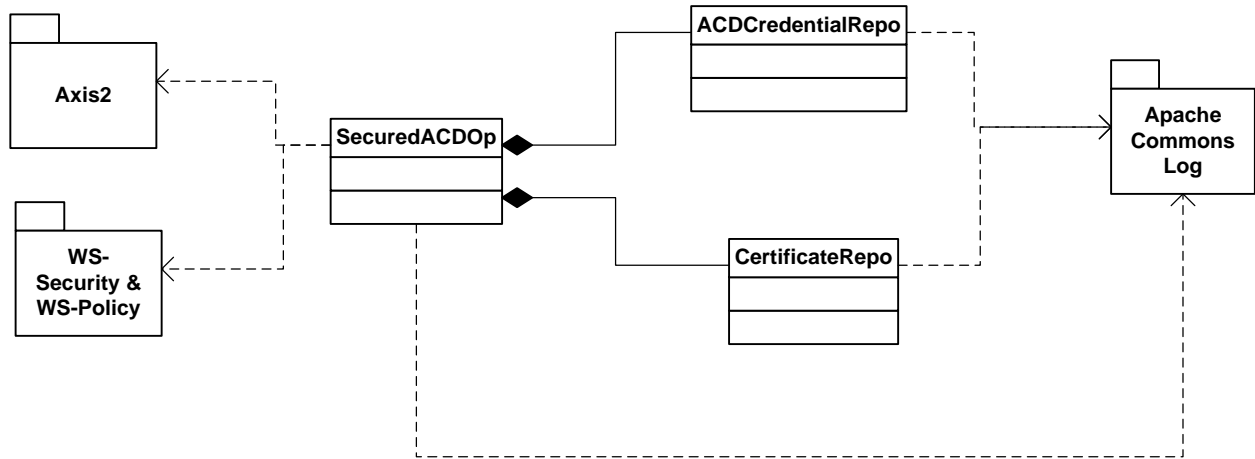


Figure 29: ACD Credential Repository Service UML diagram

This component stores the security certificates belonging to specific virtual organisations and the corresponding private keys required to communicate with the grid. This component enables the creation and management of VO membership, allowing members to access the grid resource. This component maintains a list of issued proxy certificate and their corresponding private keys and maintains the association between each user and proxy issued. From the grid resource point of view, all commands are seen as arriving from the VO and not from individual users. However, the VO is able to distinguish which command is issued by which user through the proxy-user mapping.

The **SecuredACDop** class provides the key features as described above. It utilises **ACDCredentialRepo** and **CertificateRepo** classes which handle credentials and certificate information respectively, as seen in the UML diagram in Figure 29.

4.3.6 Interfaces

AHE and ACD will provide the following basic interfaces using a visible, well-defined and Cloud-compliant REST format in the form of:

`http://example.ac.uk/ahe/as1/AHE-SPECIFIC-COMMAND-STRUCTURE.`

In general, most AHE REST command will be in the following format:

`http://example.ac.uk/ahe/as1/USERID/AHE-COMMAND?para=value&...`

The specific interfaces of the presented platform are described in the following subsections.

4.3.6.1 AHE interface

 `status` – this command can be used to check a job or AHE server status



- `getData` – set where and how to get data (such as job results)
- `putData` – set where and how to upload data for processing
- `prepare` – prepare the execution of an AHE job
- `monitor` – monitor job execution
- `getRegistryEPR` – get a reference to the application registry
- `start` – commence job execution
- `terminate` – terminate job execution
- `setProperty` – set a property of the AHE server
- `getProperty` – get a property of the AHE server

4.3.6.2 *SecureACDOps interface*

- `removeProject` – remove a project
- `getCertificateDetails` – get the details of a project certificate
- `generateProxies` – generate a project proxy
- `updateP12ProjectCert` – update project certificate
- `getUserProjects` – get list of user projects
- `assignP12CertToProject` – assign user to project
- `createNewProject` – create a new project
- `viewCurrentProjects` – download a list of ongoing projects
- `removeUserProjects` – remove user from project

4.3.6.3 *AzmanWS Interface*

- `createNewTask` – create a new task
- `getRoleAssignment` – get user roles
- `viewAllRoles` – view all available roles
- `createNewRole` – create a new role
- `viewRolePermissions` – view all role permissions
- `viewAllTasks` – view all tasks
- `assignUserToRole` – assign user to role
- `revokeUserRole` – revoke user role

4.3.6.4 *AccessMonitorWS interface*

- `accessMonitor` – obtain access to monitoring operations

4.3.6.5 *SecureAuthenticationAdminOps interface*

- `removeUser` – remove a user account
- `changePassword` – change user password
- `createUser` – create a user account
- `updateUserDetails` – set description of specific user
- `getUserDetails` – download description of specific user
- `resetPassword` – reset user password



4.3.6.6 *AuthenticationOP interface*

🌐 login – log into AHE

4.3.6.7 *Other interface dependencies*

AHE and ACD will also require the following interfaces from the VPH-Share security mechanism, as well as the Cloud data storage component (pseudo commands are listed):

- 🌐 **Get data** – fetch user data from VPH-Share Cloud storage
- 🌐 **Put data** – upload user data to VPH-Share Cloud storage
- 🌐 **Authenticate user** – authenticate a given user
- 🌐 **Get user credentials** – get user security attributes

4.3.7 *AHE/ACD implementation technologies*

AHE and ACD will use the following protocols, libraries and frameworks:

1. Java SE and Java EE (35)
2. Hibernate ORM (36)
3. Quartz Scheduler (26)
4. JBoss jBPM (25)
5. JavaGAT (Grid Toolkit) (37)
6. JavaCOG Toolkit / JGlobus
7. Tomcat Server (38) / Eclipse Jetty Server (39)
8. File protocols: webDAV, gridftp, SFTP, NFS
9. Restlet library to provide RESTful web services (40)
10. Apache Common CLI (41)
11. Apache Log (42)
12. Axis2 (43)
13. WS-Security and WS-Policy (44)

4.4 **Access to large binary data in the cloud**

4.4.1 *Introduction*

As scientific data tends to become larger every day, storing and managing it becomes an important issue. In the case of data-intensive applications, although storage resources may be abundant, access to large binary data may prove to be a big drawback. Thus, to facilitate access and process large data pieces, transparent and efficient storage mechanisms must be in place. Cloud storage promises to virtualised storage on demand. However, data is not likely to be stored in just one storage framework or with a single vendor.

The Large Object Cloud Data storageE fedeRation (LOBCDER) aims to ensure 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 namespace. This unified view provides benefits such as:

- 🌐 distribution of data to ensure accessibility and reduce latency



- dynamic addition and removal of storage resources
- provisioning virtually limitless storage capacity
- improved collaboration through data sharing
- simplifying the process of developing scientific applications

In order for the unified storage space to be usable for application developers, end users and administrators, LOBCDER provides an intuitive interface that all actors can use, in the form of a webDAV server. Additionally it is desirable to provide one standard common interface for all the actors, a feature that reduces the design complexity as well as maintenance overhead. Figure 30 shows a conceptual view of how different actors interact with LOBCDER.

By providing a standard frontend, LOBCDER is decoupled from specific client implementations. This enables application developers to freely choose APIs and client implementations that will allow them to mount and use the available storage space. There are several clients that enable mounting webDAV servers. Some of them include davfs2 (45) and wdfs (46) for Linux platforms, while for Windows there is a built-in client and a third-party client called NetDrive (47). Additionally there is a multitude of APIs developers may use, such as Jackrabbit (48) for Java, Onion (49) for C++, PerlDAV (50) for Perl and so on.

As mentioned above, the frontend interface must be intuitive and allow existing client implementations and APIs to mount the available storage space. This greatly benefits end users, as they do not have to navigate through complicated applications in order to use LOBCDER. Finally, administrators also benefit from a standard frontend, as they have at their disposal a common management layer able to govern loosely coupled storage resources.

Federating and aggregating multiple storage resources can provide some benefits, such as the ones mentioned previously. This however, calls for an operations component able to implement a wide spectrum of policies, as well as handle requests coming from the frontend. An example of such a policy is to replicate entire data resources over the available storage resources to increase accessibility. Another example is to divide individual data resources into blocks, and to spread them over the available storage resources. This approach decreases access latency and avoids vendor “lock in” problems in the case of public clouds. The simplest storage policy is a one-to-one policy, where a data resource is located in only one storage resource. With this policy, the location of data is under the owner’s control.

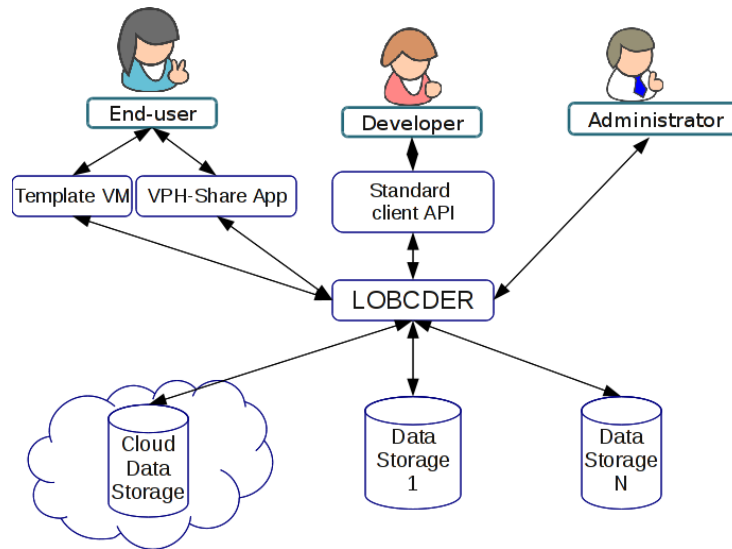


Figure 30: LOBCDER interactions with application developers, end-users, and administrators

Besides storage policies, the operations component takes into consideration user access policies. These policies promote collaboration and sharing, while at the same time ensuring that only authorised users and software obtain access to specific data resources.

To bind all the available storage together, a versatile and extendable component will be put in place. Since cloud implantations vary, and in some cases different storage frameworks may have to be federated and aggregated, the use of an adaptive and extensible component is apparent.

4.4.2 System Overview

From the description provided above, the LOBCDER service is divided into three main components: the frontend interface, the operations backend and the access backend. Figure 31 shows the overall design of LOBCDER.

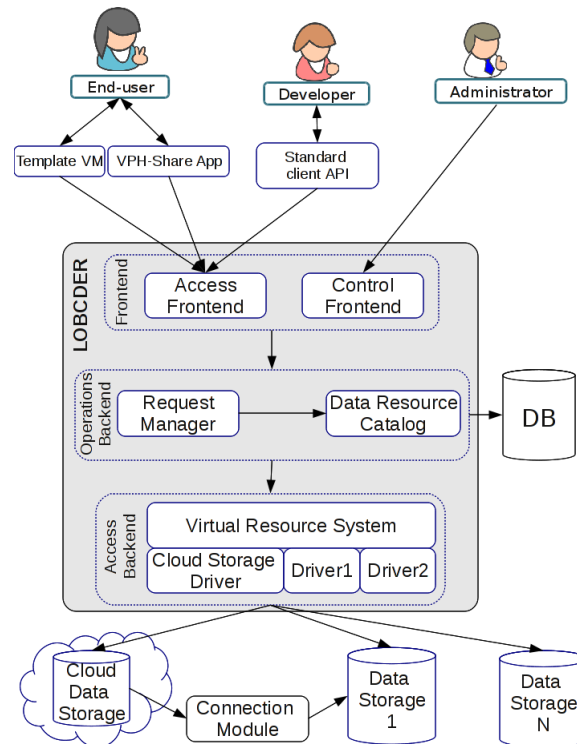


Figure 31: LOBCDER architecture

4.4.3 Frontend

The frontend's purpose is to provide access and control through standardised, user-friendly interfaces. This removes the need for developing and maintaining specialised clients. The frontend is made up by two major components: control and access frontend.

The control frontend provides administrators with the means to manage the behavior of LOBCDER as well as its underlying storage resources. More specifically, the control frontend provides the methods to add and remove storage resources with their corresponding credentials and sets the redundancy policy.

The access frontend provides applications and users with a standardised access interface. This standardised interface allows clients to mount the available storage space and present it in a single namespace. This enables end users and developers to transparently use storage resources that would otherwise be complicated to manage and use. Figure 32 depicts the interaction of end users and developers with LOBCDER's access frontend. In addition, the access frontend provides the following methods:

1. **Copy:** creates a duplicate of a source dataset (identified by its Logical Data Resource Identifier or LDRI), at a destination data resource
2. **Delete:** removes a data resource
3. **Lock/Unlock:** locks and unlocks access to a data resource while its owner is modifying it and making it available again
4. **Make Collection:** creates a new collection or bucket at the location of the specified LDRI
5. **Move:** moves a data resource to the location specified by an LDRI
6. **Get Properties:** retrieves properties for a data resource such as mime type, and length

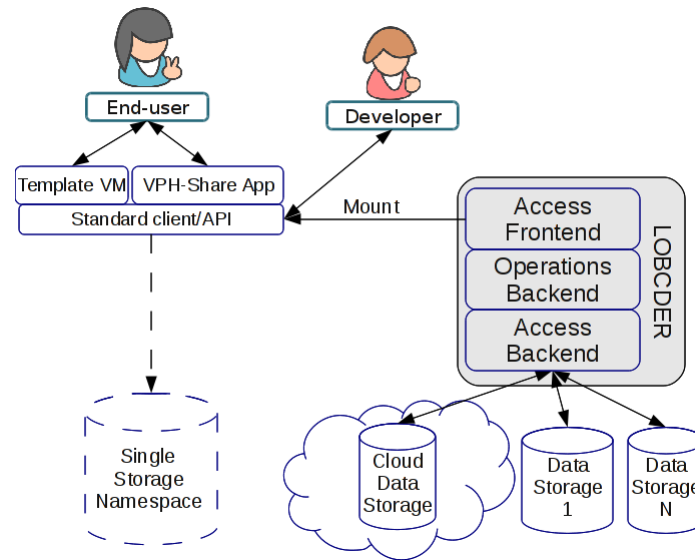


Figure 32: End users and developers can use standard clients and APIs to mount LOBCDER

4.4.4 Operations Backend

The purpose of this middle layer is to map logical instances to physical ones. It takes requests from the frontend and maps them to existing storage resources and associated credentials.

The request manager's role is to accept requests from the access frontend, forward LDRI to the Data Resource Catalogue, get the corresponding entry, and, if the user has the necessary permissions to access a specific data resource, obtain its physical location. If the request manager receives a request to allocate new data, it tries to do so in the physical space available to the requestor. Subsequently, it creates an entry in the Data Resource Catalogue.

There is a special LDRI where the data is mapped directly to physical resources bypassing the LOBCDER intelligence for smart storage allocation and replication. If a user stores the data in a given LDRI, data will be transferred directly to the physical storage resource associated with this LDRI (here LDRI is a structure in the Data Resource Catalogue, similar to a directory in a regular file system). For example, if an LDRI (a directory) has the name "fs2.das3.science.uva.nl" then all of its children will allocate storage on the specified resource and LOBCDER will not try to automatically replicate data from there. This might be important for sensitive data which must reside on a specific resource and never leave this resource.

Additionally, the request manager communicates with the access backend, passing requests for physical resources. Apart from receiving requests from the access backend, the request manager also accepts requests from the control frontend. If the request is to add or remove storage resources, the manager connects with the database to make the necessary changes.

The Data Resource Catalogue's role is to resolve LDRI to physical access locations and provide metadata about stored objects, as well as user permissions for data resources. This is performed with the use of a storage resources database in the WP2 persistence layer (the

Atmosphere Internal Registry, described in Section 4.1.4). This database holds data resource entries that bind physical with logical data.

4.4.5 Access Backend

This component provides the necessary abstraction for uniformly accessing physical storage resources. Its main building block is a Virtual Resource System (VRS), able to abstract any physical resource system, thus providing a uniform API to the components above it. Moreover, VRS, with its plugin framework, allows for implementation of additional drivers, making LOBCDER extensible.

4.4.6 Data persistence

For LOBCDER to federate storage resources it requires a description of these resources. This is provided by the StorageResource entry with the following data members (Figure 33):

- 🌐 **URI:** A resource identifier of the following type: `schema://location`
- 🌐 **Credential:** The necessary credential for accessing that resource
- 🌐 **Permissions:** A set of VPH users eligible to access this resource
- 🌐 **Properties:** An open set of properties that describe that resource, such as capacity, bandwidth, etc.

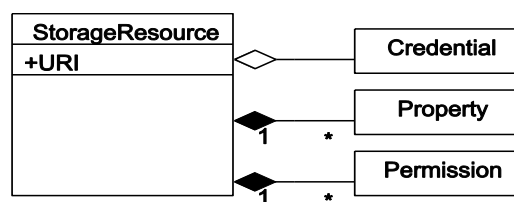


Figure 33: StorageResource structure

For a description of data resources, LOBCDER uses a DataResource entry. This entry binds logical and physical data objects, and has the following data members (Figure 34):

- 🌐 **UID:** a unique id of the data resource
- 🌐 **LDRI:** a logical data resource identifier, for example `/path/name`
- 🌐 **PDRI:** an URI to the physical resource where the actual data is stored, including all replicas. PDRI include a reference to associated StorageResource responsible for data allocation
- 🌐 **Owner:** VPH user Id of the data resource owner
- 🌐 **Permissions:** Access policy for this DataResource entry. May include a full list of VPH Users eligible to access the resource
- 🌐 **Metadata:** a metadata derived from physical resources, such as length. May include additional metadata associated with content of the file
- 🌐 **Children:** if DataResource is a directory then contains list of other DataResources located inside

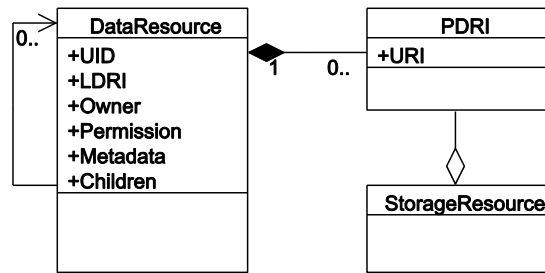


Figure 34: DataResource structure

LOBCDER provides an API to attach plugins which are invoked on operations coming from Access Frontend. A plugin has ability to update metadata associated with DataResource. A plugin attached to “close” operation may analyse content of the DataResource and update Metadata entry. An example of such plugin is a subsystem which extracts metadata from DICOM (Digital Imaging and Communications in Medicine) files and updates Metadata with properties of the image file, like resolution, etc. These properties are saved in DataResource and later can be used as parameters for the queries.

Both of the descriptions mentioned here are stored in the WP2 persistence layer to allow access to the available metadata.

4.4.7 Connection Module

To enable fast data transfers between storage resources, LOBCDER employs high performing transfer protocols such as GridFTP, and UDT. Together with these transport protocols, the right storage policies optimise data transfer rates, by taking advantage of parallel streams and third-party transfers. Moreover to avoid centralisation bottlenecks, connection modules are deployed next to or near the data.

4.4.8 Scaling

In order to cope with a large amount of requests and provide faster data transfers with minimum latency, the LOBCDER service will be deployed in multiple resources as shown in Figure 35. This enables requests for data resources to be served by LOBCDER instances closer to the requested data. This means, that all LOBCDER instances share access to the data persistence layer described in Section 4.4.6, with the addition that each LOBCDER instance will have to calculate the “distance” from a client request, and redirect it if another peer is situated “closer” to the client.

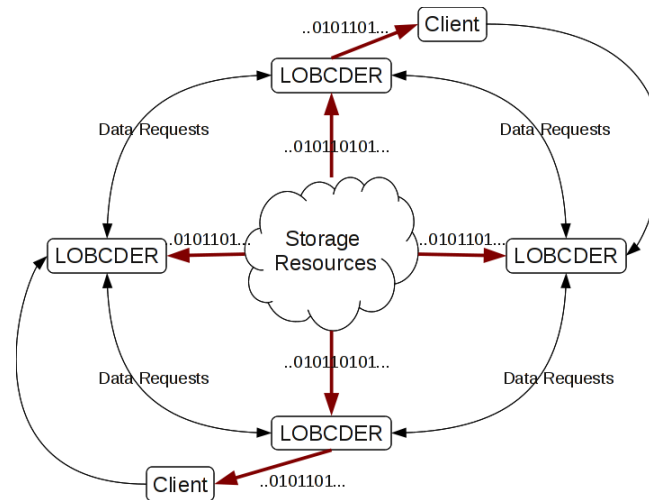


Figure 35: Distributed LOBCDER architecture

4.4.9 Usage scenarios

If an administrator wishes to register a new storage resource with the use of the control frontend, he/she would have to provide the following information:

- 🌐 the resource's URI, in the form of `schema://location`
- 🌐 the credential for accessing this storage resource
- 🌐 list of users who can use this resource
- 🌐 properties describing that resource (i.e. the quota)

This request is forwarded to the request manager, which in turn creates a new **StorageResource** entry in the database.

Alternatively if an administrator wishes to remove a resource, he/she needs to provide its URI. The request will be forwarded to the request manager, where the entry will be removed from the database, as long as it's not holding registered data. If, however, this storage resource holds data, the request manager will have to check if that data are replicated in some other storage resource. If not, the control frontend offers a choice to migrate the data to some other storage resource.

The two other actors, namely the end user, and the application developer, perform the same actions on the LOBCDER service. These actions, as described in Section 4.4.1, are:

- 🌐 **Copy:** in order to perform a copy of a data resource, the request manager creates a new **DataResource** entry, and, depending on the storage policy, the access backend creates a new physical copy of the data resource
- 🌐 **Delete:** when requesting to perform a delete, the request manager will have to get all the physical copies of the data resource, and then remove the relevant **DataResource** entry
- 🌐 **Lock/Unlock:** when locking a data resource (i.e. binary file), the request manager, blocks any other requests for that specific data resource. Equally, when unlocking the request manager allows requests for that data resource



- **Make Collection:** since a collection is a logical concept containing other data resources, this request only creates a new **DataResource** entry
- **Move:** when moving a data resource the request manager only renames the **DataResource**'s LDRI
- **Get Properties:** depending on the property requested, the request manager will return the relevant metadata found on the **DataResource**'s metadata

4.4.10 Implementation Technologies

LOBCDER is developed in Java as a web application, able to be deployed on application servers such as GlashFish or Tomcat. These application servers also provide mechanisms for load balancing and increasing the availability of the applications they hold. Moreover, the frontend uses standard technologies that enable client application to mount the storage space federated by LOBCDER. In addition these application servers can provide good web frontend solutions for the access and control frontends.

The access backend uses the JClouds API, which is an OCCI (Open Cloud Computing Interface) implementation to access OCCI compatible cloud storage services. To connect to the available databases LOBCDER can use RESTful APIs, as well as other persistence frameworks.

4.5 Data reliability and integrity

Data reliability and integrity (DRI) is needed to ensure sound 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. As remarked in (2), today's Cloud delivery models do not offer means for the Cloud user to perform such auditing tasks in a certified and trustworthy manner. 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

In order to facilitate these goals, we begin by introducing the concept of a *managed dataset*, which represents a specific data item (a file or a collection thereof) which is known to the VPH-Share services and for which specific information can be located in the VPH-Share metadata registry. Managed datasets can be registered with VPH-Share upon being uploaded to the Cloud infrastructure – subsequently the infrastructure should be able to monitor and track their availability in an automatic manner. Thus, the functionality provided by Task 2.5 is intimately tied to the data access layer provided by Task 2.4.

4.5.1 Structure of a Managed Dataset

As Work Package 2 concerns itself with access to binary data, the mechanisms developed in Task 2.5 are specifically tied to the requirements and properties of file-based storage. This is

in line with the Project’s Description of Work (1) and with the features of data integration technologies derived from Task 2.4, with which Task 2.5 will be closely integrated (see Section 4.4 for details). Managing structured storage, such as relational databases, populated with semantically-rich data (e.g. metadata extracted from DICOM files) falls within the scope of Work Package 3 and is subject to separate verification mechanisms (although database servers themselves, along with the data they store, can be registered and managed by Atmosphere as Atomic Services – in fact, this is the preferred way of exposing relational data repositories to users of VPH-* collaboration. Please refer to (51) for details.).

At its core, a managed dataset consists of a selection of files which are uploaded to the VPH-Share data storage infrastructure and which can be tagged for automatic management with the use of an appropriate interface (see below). This concept is schematically presented in Figure 36.

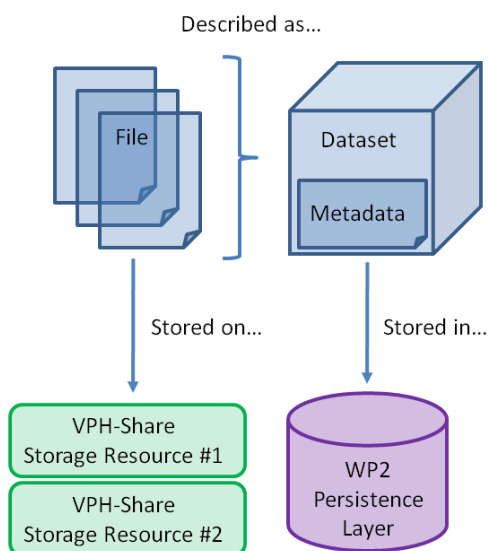


Figure 36: Schematic representation of a VPH-Share Managed Dataset

Each managed dataset may consist of an arbitrary number of files (in the simplest case, just 1 file), to which a selection of metadata is appended and stored in the WP2 persistence layer. Managed datasets map directly to data resource objects described in Section 4.4.6, where each dataset consists of one or more DataResource entries. Thus, the description of managed datasets will necessarily include all the metadata associated with data resources, in addition to some DRI-specific metadata. Initially, the DRI metadata extensions are expected to consist of the following elements (on a per-file basis):

- 🌐 **owner** (reference to VPH-Share user ID)
- 🌐 **method of generation** (uploaded manually, generated by a VPH-Share Atomic Service Instance or registered externally)
- 🌐 **date of registration**
- 🌐 **checksum** (this will be calculated by means of a cryptographic hash function whenever a resource is registered. The value of this function will be stored in the Atmosphere Internal Registry and used to validate the integrity and availability of each file)



- list of **storage resources** (see Section 4.4.6) to which the file is currently deployed (internal reference to WP2 persistence layer data storage schema)
- **access log** (as reported by Atmosphere)

While this schema is expected to cover all the requirements addressed in the Project's Description of Work and in Deliverable D2.1, we foresee that additional metadata can become necessary at later stages in the Project's lifecycle. To this end, the metadata schema will be extensible without the need to clear the contents of the WP2 internal registry. Any updates to the managed dataset schema will be reported and elaborated upon in subsequent WP2 deliverables (i.e. platform prototype descriptions).

4.5.2 Tagging datasets

Before automatic verification of managed datasets can take place, it is first necessary to tag specific data as subject for management. It is foreseen that the DRI component will involve a user interface extension (portlet-based) to enable authorised administrators (possessing a specific role) to tag specific datasets for automatic management (please refer to Section 2 for a description of VPH-Share administrator responsibilities). This interface will display the existing data storage resources and allow creation of new managed datasets consisting of selected files. For each new dataset tagged by administrators, a selection of metadata will need to be provided. Most of the attributes listed in the previous section can be supplied automatically by the system – at a minimal level of involvement, the administrator will merely be asked to provide a unique name for the newly created set. As an extension of the user interface, the administrators will also be able to modify and/or purge selected datasets.

In addition to UI-based tagging, the DRI component will also provide API-level access for the same purpose, whenever a VPH-Share application (or workflow) needs to tag specific data as a managed dataset. This option will be provided in the context of useful data generated by applications as opposed to data registered manually by end users. See Section 4.5.4 for further details.

4.5.3 The DRI Runtime service

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. It is expected to operate autonomously (without having to be invoked by administrators) although it will respect the policies defined as part of the managed dataset metadata descriptions and stored in the WP2 persistence layer.

The DRI Runtime will be implemented as 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).

In line with the Atomic Service specification described in Section 3, the DRI Runtime will be deployed into a virtual machine and then registered with Atmosphere mechanisms for automatic management. It is envisioned as a persistent Atomic Service (i.e. once running, it

would not be subject to automatic recall, except as requested by system administrators). At the core of the service is an application that periodically polls the WP2 internal registry for lists of managed datasets and then proceeds to verify the following:

- the availability of each dataset at locations read from the internal registry
- the integrity of each dataset (checksum-based validation)

This mechanism is depicted in Figure 37.

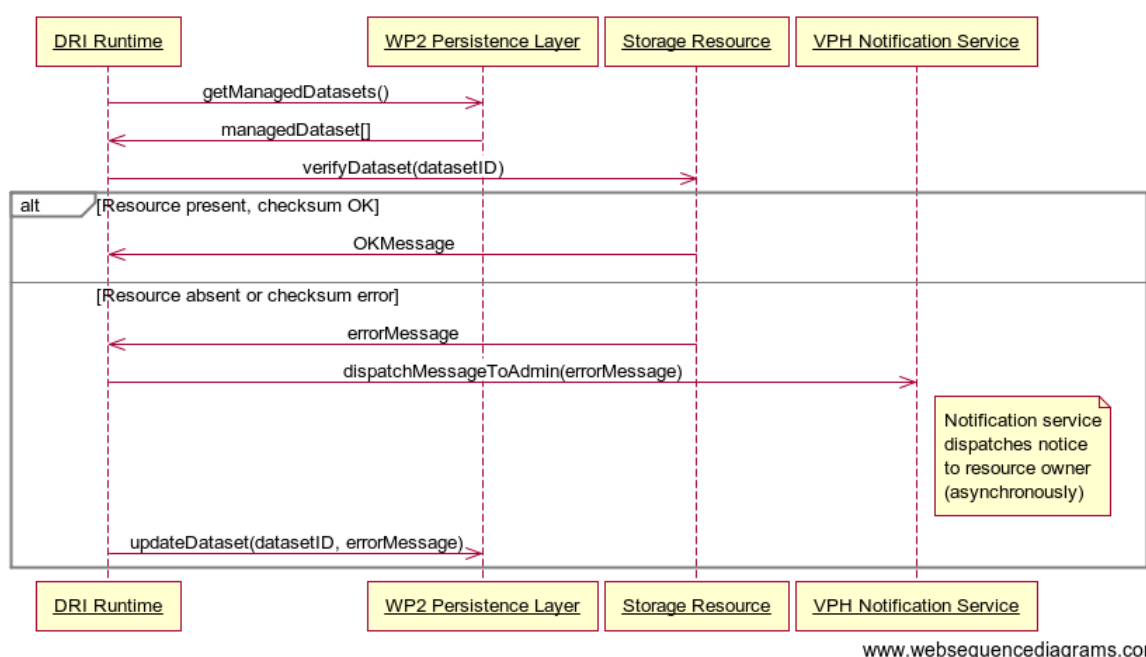


Figure 37: Autonomous operation of the DRI Runtime.

The DRI Runtime will contact individual storage resources (registered for use in the VPH-Share project) and validate the integrity and availability of the data stored on these resources. Should errors occur, the DRI Runtime will invoke a notification service (see Section 4.6 for details) to issue a warning message to subscribed system administrators (typically, the user defined as the dataset’s owner).

In addition to controlling the availability and integrity of data stored in the Cloud infrastructure, the DRI component will also be able to schedule data replication with Task 2.4 tools, when requested by the administrator or by the Atmosphere application execution components. Again, the same request can be issued manually (via the appropriate Master UI extension) or by invoking an API method of the DRI Runtime. DRI will recognise data storage resources registered in the WP2 internal registry (managed by Task 2.4 tools) and provide the ability to replicate a dataset to one or more storage resources. This feature may be exploited by Atmosphere to ensure that computations occur in close proximity to data, which is an important prerequisite of maintaining competitive performance of the platform as a whole.



4.5.4 DRI Interfaces

As hinted upon in the preceding sections, DRI will provide end-user interfaces in the form of a Master UI portlet, as well as an API implemented by the Runtime service, where DRI features may be invoked directly by other VPH-Share infrastructure components.

The specifics of the DRI portlet will be elaborated upon in the corresponding WP6 deliverable; here we intend to focus on the API, which will provide access to the low-level functionality of DRI and enable it to be configured.

As DRI exposes a stateless Web Service, all configuration parameters are stored in the Atmosphere Internal Registry. Whenever a configuration change request is invoked, the DRI will automatically update its monitoring policies stored in AIR. In light of this, the DRI API should support the following operations:

- 🌐 `getStorageResources()`: `storageResourceID[]` – returns a list of currently registered data storage resource identifiers
- 🌐 `getStorageResourceData(storageResourceID)`: `storageResourceDescription` – returns information on a specific storage resource, as stored in the Atmosphere Internal Registry. **storageResourceDescription** is an XML document describing the structure of each storage resource
- 🌐 `getManagedDatasetInfo(datasetID)`: `datasetDescription` – requests information on a specific managed dataset stored in the Atmosphere Internal Registry. Returns an XML document specifying the structure of the managed dataset
- 🌐 `registerManagedDataset(datasetDescription)`: `datasetID` – tags a new dataset for management with Task 2.5. **datasetDescription** is an XML document describing the structure of the dataset. This information will be automatically parsed by DRI and stored in the Atmosphere Internal Registry. Returns a unique identifier of the new dataset
- 🌐 `alterManagedDataset(datasetID, datasetDescription)`: `void` – changes the dataset specification stored in the Atmosphere Internal Registry. This action should be used to add or remove files from a managed dataset
- 🌐 `removeManagedDataset(datasetID)`: `void` – excludes the specified resources from automatic management. This does not delete the resources; it merely stops the DRI runtime from polling them
- 🌐 `assignDatasetToResource(datasetID, storageResourceID)`: `void` – requests DRI to monitor the availability of a specific managed dataset on a specific storage resource. If this dataset is not yet present on the requested storage resource, it will be automatically replicated there with the use of Task 2.4 tools
- 🌐 `unassignDatasetFromResource(datasetID, storageResourceID)`: `void` – requests DRI to stop monitoring the availability of a specific managed dataset on a specific storage resource. This action does not delete the dataset on the given storage resource. If the dataset is not present on the selected storage resource, this action has no effect
- 🌐 `getChecksumsForDataset(datasetID)`: `checksum[]` – produces a set of cryptographic hashes for each component of the managed dataset. This information



- should not normally be of interest to external users (it is used internally by DRI to validate the availability and integrity of datasets)
- `validateDataset(datasetID)`: output – performs asynchronous validation of the specific dataset and produces a document which lists any problems encountered with the dataset's availability on the storage resources to which it had been assigned
 - `setManagementPolicy(managementPolicy)`: void – changes monitoring parameters. **managementPolicy** is an XML document specifying the frequency and type of availability checks performed on managed datasets
 - `getManagementPolicy(datasetID)`: `managementPolicy` – retrieves an XML description of the management policy applicable to a specific dataset

Each invocation will need to be augmented by a security token which can be intercepted and parsed by the security Policy Enforcement Point residing on the virtual machine on which the DRI Runtime operates. This issue is out of scope of this section but will be covered in more detail in Section 4.6.

4.5.5 Implementation Technologies

The DRI Runtime will be developed as a standalone application set up on a virtual machine obtained from Atmosphere management services (Task 2.1) and registered as a persistent Atomic Service. Following launch (which will occur automatically whenever an instance of the Runtime is deployed), the application will enter a cycle (of configurable duration) where at fixed time intervals it will contact the WP2 registry and subsequently validate managed datasets with the use of Task 2.4 access tools.

The application will be developed in Ruby/JRuby (which can import native Java archives and invoke them as if they were Ruby objects). A component of the Runtime will be deployed to an application server (we are likely to use Sinatra for this purpose (13)) and expose a RESTful Web Service API, which will provide the backend the functionality of the DRI component. The service will be secured with authenticity tokens and will contact the Task 2.6 Policy Decision Point (see Section 4.6 for details) to guard against unauthorised access. Console access to the virtual machine on which DRI components reside will be limited to WP2 developers and/or system administrators.

The Atmosphere Internal Registry will be the persistence layer used to store DRI metadata and descriptions of managed datasets. A discussion of the technologies used in its implementation can be found in Section 4.1.

User interface extensions will be implemented using a Portlet-2.0 compliant interface, in Java; however the decision on which specific portal platform to adopt for the VPH-Share Master Interface rests with WP6. The issue will be addressed in D6.2, which is due by Month 9 of the Project. Regardless, it is expected that the portlet solution developed in Task 2.5 will remain compatible with the Master Interface by way of compliance with the JSR-286 standard (52).



4.6 Security for Cloud applications

Our commitment is to provide an integrated security for the VPH-Share results, ensuring that their use by consumers is done in an authorised manner by trusted users.

Some general concepts in the services and security domain should be explained before specifying the security for VPH-Share cloud applications.

- **Service:** our solution is limited to endpoints with public interfaces. Security is applied at the interaction/transport level, not at the application level. Thus, even if this section often refers to the VPH-Share endpoints as “Services” or “Atomic Services”, it should be understood as: any software readable resource, registered and known to the VPH-Share platform by means of a prescribed specification.
 - **Service specification:** it represents the service by defining (logically) its requirements, policies, and any behavioural aspect that might be relevant to the platform and the consumers. The abstract specifications of services should include any security requirement, any trust relationships that must be established between it and other parts of the platform (or an external entity), and the security properties that must be made available to it for its execution/mediation needs. Collecting such security requirements from the architectural design and the atomic services specifications will be the next task upon completion of this deliverable.
 - **Service descriptor:** this artefact should be seen as a document-based instantiation of the behavioural aspects described in the service specification. Specifically, service descriptors contain the information needed to interact with services. Conditions and properties that are relevant to security will be registered and maintained in these descriptors, so that they can be consumed by other services.
 - **Service contract:** although providing tools to establish and monitor SLAs between application providers and end users is out of scope of the project, all the artefacts, functional metadata, requirements and policies that might condition access to a service at the transport or application level will be considered an agreement from the security solution’s point of view. Any security property that is used to filter access to a resource is part of the service contract. The fulfilment of expectations will be validated by compliance checking and auditing tools.
 - **Environment:** this broad term refers to the surroundings of a service. This may include people, devices, software and conditions that influence or interact with the service. The system administrator configures the platform and thus defines the environment for the security conditions to be enforced.
- **Actors:** three roles have been defined for VPH-Share (see Section 2 for details), and for each of them security mechanisms must behave in a different way. Their needs and expectations are covered in different use cases and tools. These roles are not mutually exclusive. In this section, we extend the previous definition by providing some aspects of each role that are relevant to security.
 - **Application provider:** this actor can configure services to make certain capabilities available to its consumers. This includes defining constraints that will be applied to filter consumer requests to the service. In VPH-Share, application providers are participants of the project.



- ❏ **Domain scientist:** this actor interacts with the platform in order to realise a capability. The platform may or may not fulfil the scientist's needs, depending on the access restrictions defined for it or its domain. In order to interact with the platform and mediate/invoke services, scientists must be users of the BioMedTown portal.
- ❏ **System administrator:** this actor can define the security environment in which domain scientists and application providers interact with the platform tools and services. This includes defining patterns by which trust is established between consumers and providers, as well as defining conditions that might grant users access to further capabilities. In VPH-Share, they are participants of the project.
- 🌐 **Security:** our commitment is to provide a specialised security framework, configurable and customisable, that can be integrated into the VPH-Share solutions. The activities needed to provide this security, and how they are parameterised, have caused additions to the VPH-Share architecture. This parameterisation is based on some variables defined at atomic service level, that later are reused and extended for specific instances of the atomic service.

4.6.1 Architectural Design

This section provides a high-level architecture description of the security solution for protecting Atomic Services within the VPH-Share platform. It has three components:

- 🌐 the Security Management framework, which encapsulates the management functionalities in an OSGi container, and is accessible to System Administrators and Application Developers through the Master Interface
- 🌐 the Security Proxy, actually a pair of proxies (to intercept incoming and outgoing traffic from the Virtual Appliance), that will filter the message requests sent to the Atomic Service, and protect its communication
- 🌐 the Security Agent, which provides the Security Proxy with a set of tools so that it can filter the messages by authenticating the Requester and enforcing access control conditions

There is an instance of the Security Manager in the WP2 Deployment, to handle all the Security Proxies/Agents that would be deployed, one per Atomic Service instance. The Security Manager will interact with the actors through the Master Interface, and will also be the application layer that operates with the Permanence layer designed for the results of this Work Package.

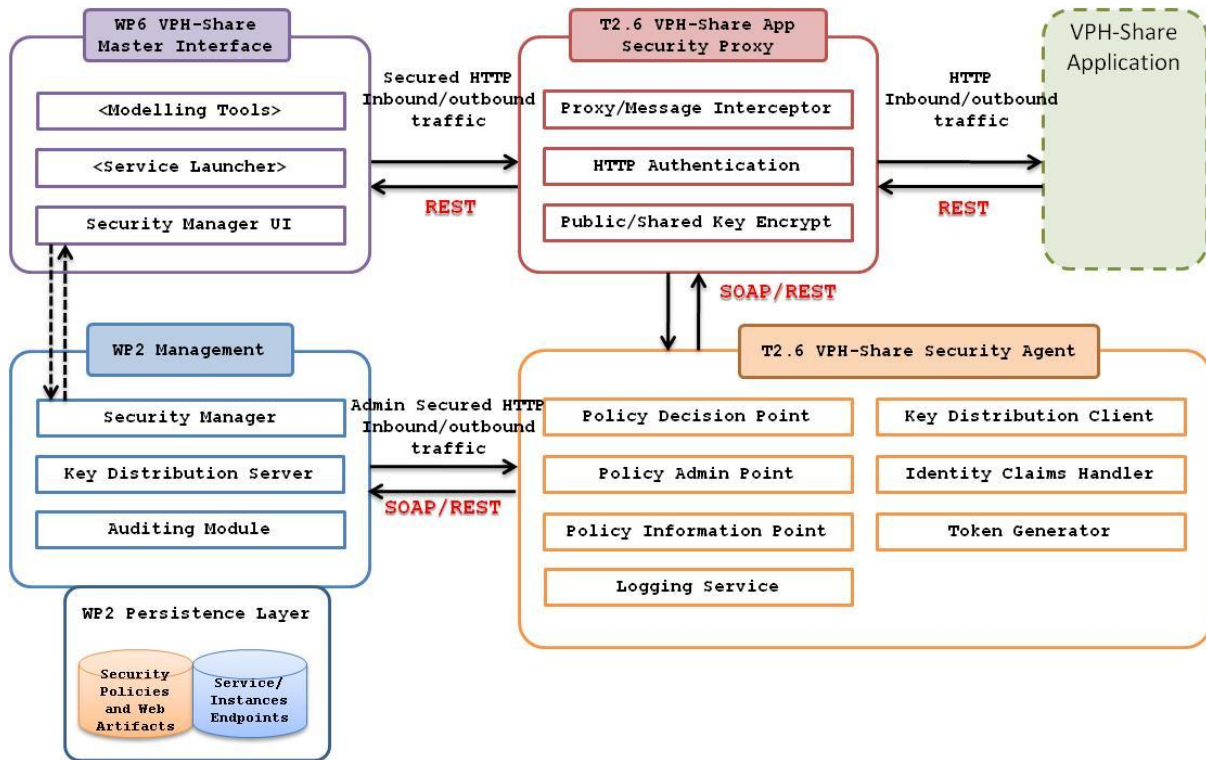


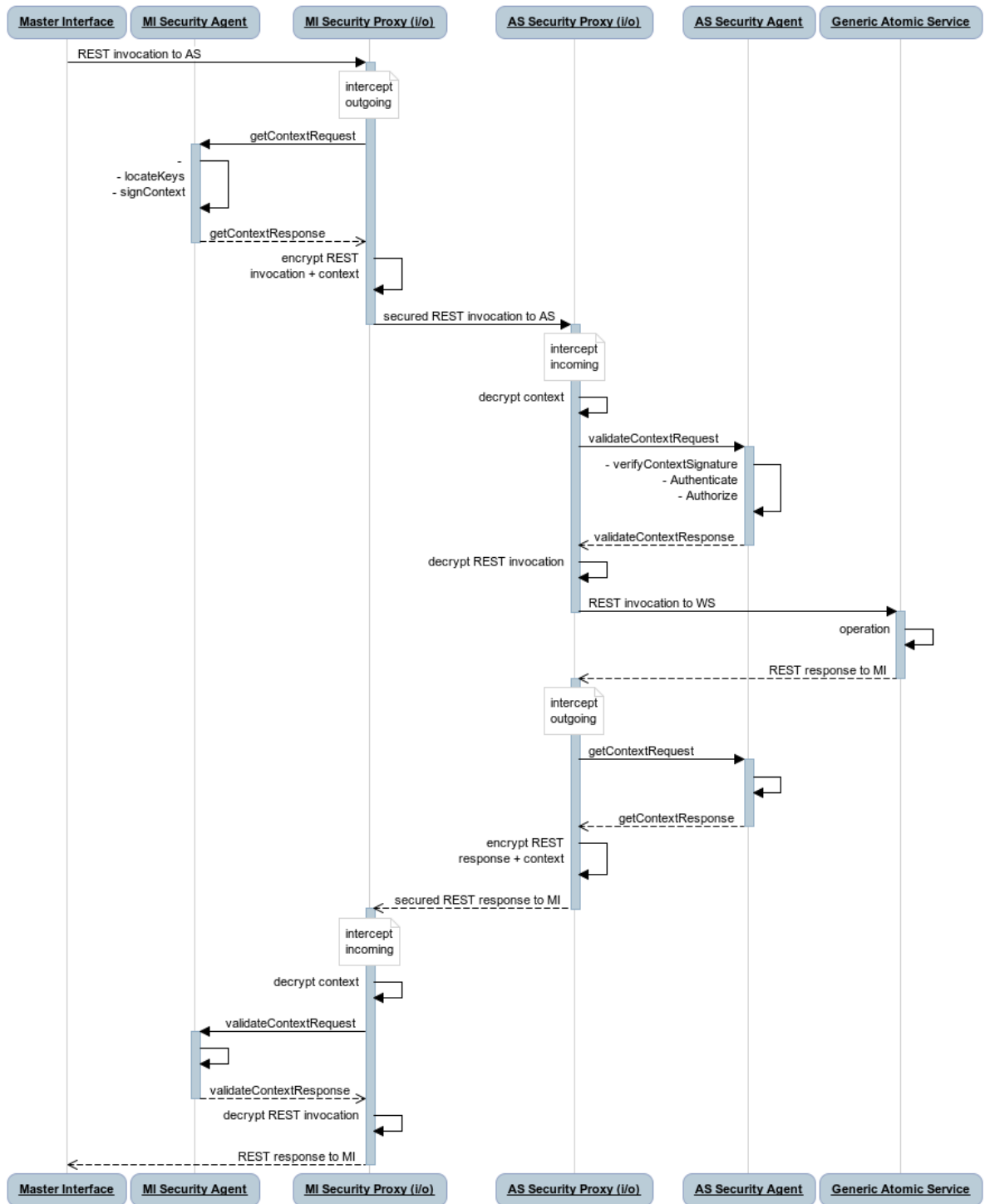
Figure 38: Overall architecture of the VPH-Share Security Components

The Security Proxies and the Security Agents, on the other hand, will operate in each Virtual Appliance (or VPH-Share endpoint, including the Master Interface). We provide them with administrative APIs so that the Security Manager can configure them using a special “channel”, beside the consumer’s requests and service responses.

4.6.1.1 Security Proxy

Proxies are deployed and configured in the Virtual Appliance. They are Apache Server modules that intercept incoming and outgoing traffic, and they can be configured to invoke ‘filtering’ routines that can extract a message, modify it and insert it back in the traffic. We will route the ‘filtering’ routines to the Security Agent, and based on the header of the message, will authenticate and authorise the service invocation.

Moreover, other Apache modules will allow us to encrypt and decrypt the message, given some patterns. At this time, the community has not produced an Apache module that can be configured to perform encrypting/decrypting according to several patterns. The available modules only follow one specific pattern. We will therefore extend the Apache module so that it is able to interpret which pattern it has to use, and where to fetch the needed artefacts.



www.websequencediagrams.com

Figure 39: Security Proxies - Encrypting/Decrypting the AS requests and responses

Implementing an interceptor for the outgoing traffic can be tricky when compared to implementing a listener for the incoming traffic, and at this point we are assuming different implementations for Windows and Linux, as the Virtual Appliances can use both OS.

Security proxies will not interact directly with the service; they will filter its traffic according to their configuration. The Proxies of the instances will have in their internal registry the same valid endpoints list as in the moment they were instantiated. They will invoke the Security Agent whenever a message signed by a validated VPH-Share entity is directed to an unknown VPH-Share endpoint. The Security Agent will then update the Proxy's local registry with all the known valid endpoints of the platform. The internal registry of the proxies is not updated regularly, only if it needs to, as the most likely scenario is that the atomic service is always invoked by the same user/workflow. After the update, the proxy checks again the list, and if it still does not find the endpoint, it will consider it invalid and fail the request/response.

4.6.1.2 Security Agent

This component encapsulates all the needed tools to secure the end-to-end conversation between services. It is actually the interface of the whole identity management and authorisation processes. The proxy handles the service request to the agent whenever HTTP authentication and decrypting (both performed by the server infrastructure) have been done.

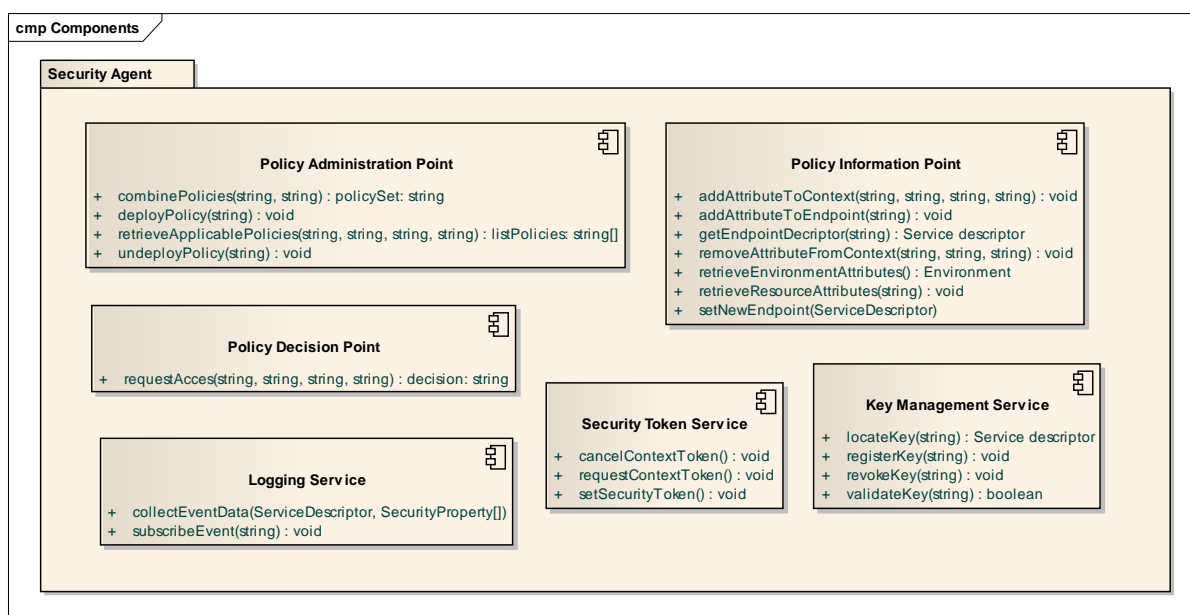


Figure 40: Security Agent – Components Diagram and interfaces

Its features are available to consumers who have signed in the platform. For this, we will rely on WP6 to provide us data required to establish a User Session.

The abstract concept of a session can be explained as the association of an initiator with a stream of communication. A session may represent a user's connection to a server, or a set of related transactions in a connection-less environment (as in the case of using a cookie to maintain the persistence of transactions between a browser client and a web server).

In this section we describe the main operations of the components of the Security Agent.



The activities of the VPH-Share backend are affected by the Security procedures. We present also some of these sequences, as use cases for the security tools.

4.6.1.2.1 Policy Decision Point

Method Name	requestAccess	
Method Definition/Description	This method issues and authorisation decision by matching the attributes of the context of the access request with the attributes of the conditions expressed in the policies	
Method input attributes		
<i>Name</i>	<i>Type</i>	<i>Description</i>
<i>Context</i>	Set of Attribute identifiers and their values	Pairs of attribute-value, expressed using the XACML context schema.
Method output attributes		
<i>Name</i>	<i>Type</i>	<i>Description</i>
<i>Decision</i>	String	Might be Allow or Deny
Dependency with other components	This method is invoked internally by the Security Agent	

4.6.1.2.2 Policy Administration Point

Method Name	retrieveApplicablePolicies	
Method Definition/Description	This method performs a lookup on the policies registry of the Security Agent, and selects those who match a specified security context. These policies can then be combined and deployed (activated) in the PDP.	
Method input attributes		
<i>Name</i>	<i>Type</i>	<i>Description</i>
<i>Context</i>	Set of Attribute identifiers and their values	Pairs of attribute-value, expressed using the XACML context schema.
Method output attributes		
<i>Name</i>	<i>Type</i>	<i>Description</i>
<i>Policies</i>	A list of Policy IDs	Applicable policies



Dependency with other components	This method is invoked internally by the Security Agent	
Method Name	combinePolicies	
Method Definition/Description	This method generates a PolicySet (a combined ordered set of policies) according to a Combining Algorithm. We implement a combining algorithm that can be described as “the most specific wins”, that is: the policy that specifies more constraints on the request overrides those that are more permissive/generic.	
Method input attributes		
<i>Name</i>	<i>Type</i>	<i>Description</i>
<i>Context</i>	Set of Attribute identifiers and their values	If a context is specified, a retrieveApplicablePolicies operation is performed to fetch the policies that must be combined
<i>Policies</i>	A list of Policy IDs	Policies that must be combined
Method output attributes		
<i>Name</i>	<i>Type</i>	<i>Description</i>
<i>PolicySet</i>	PolicySet	A PolicySet which contains all the input policies, meant to be implemented in an ordered way.
Dependency with other components	This method is invoked internally by the Security Agent	

4.6.1.2.3 Key Management Service

Method Name	locateKey	
Method Definition/Description	Whenever an atomic service must be invoked, we must know its public key to correctly establish trust and secure messaging, by invoking this method.	
Method input attributes		
<i>Name</i>	<i>Type</i>	<i>Description</i>
<i>ServiceURI</i>	URI	The URI of the service that is going to be invoked



<i>ServiceURL</i>	URL	Alternatively, this method can be invoked with unique information that is needed to communicate with the service. This includes network layer (IPv4 or IPv6) addresses and transport port numbers.
Method output attributes		
<i>Name</i>	<i>Type</i>	<i>Description</i>
<i>PKey</i>	Public Key	Digital Credential that must be used to encrypt messages to the service
Dependency with other components	This method is invoked internally by the Security Agent	
Method Name	registerKey	
Method Definition/Description	This method must be invoked to register the public key. This method will issue a message to the Security Manager (that will store the key in the permanence layer, to be available to other VPH-Share services). This must be done whenever a new keypair (private key / public key) is generated for the service	
Method input attributes		
<i>Name</i>	<i>Type</i>	<i>Description</i>
<i>Pkey</i>	Public key	The new public key of the service
Method output attributes		
<i>Name</i>	<i>Type</i>	<i>Description</i>
<i>Result</i>	String	
Dependency with other components	This method is invoked internally by the Security Agent	
Method Name	revokeKey	
Method Definition/Description	This method must be invoked to cancel the validity of a key. For a key to be revoked, a new keypair (private key / public key) must be available (registered) already for the service	



Method input attributes		
<i>Name</i>	<i>Type</i>	<i>Description</i>
<i>Pkey</i>	Public key	The public key of the service that will be invalidated
Method output attributes		
<i>Name</i>	<i>Type</i>	<i>Description</i>
<i>Result</i>	String	
Dependency with other components	This method is invoked internally by the Security Agent	
Method Name	validateKey	
Method Definition/Description	This method performs a formal validation of the key, and checks if the key has expired or been revoked. All VPH-Share keys will be issued by a Certificate Authority (CA). This method will also check that the key is issued by a trusted CA.	
Method input attributes		
<i>Name</i>	<i>Type</i>	<i>Description</i>
<i>Pkey</i>	Public key	The public key to be verified
<i>Usage</i>	String	Can be Signature Key or Encryption Key (depending if it is used to sign a message or to encrypt the payload). This will affect the validation of the trusted CA, which will be performed for signing keys.
<i>ExpirationDate</i>	Datetime	Optional. If this parameter is informed, the expiration validation will be done taking this date instead of the server's current date. Its use is to verify if the key is going to expire soon, so that a new key should be generated preventively.



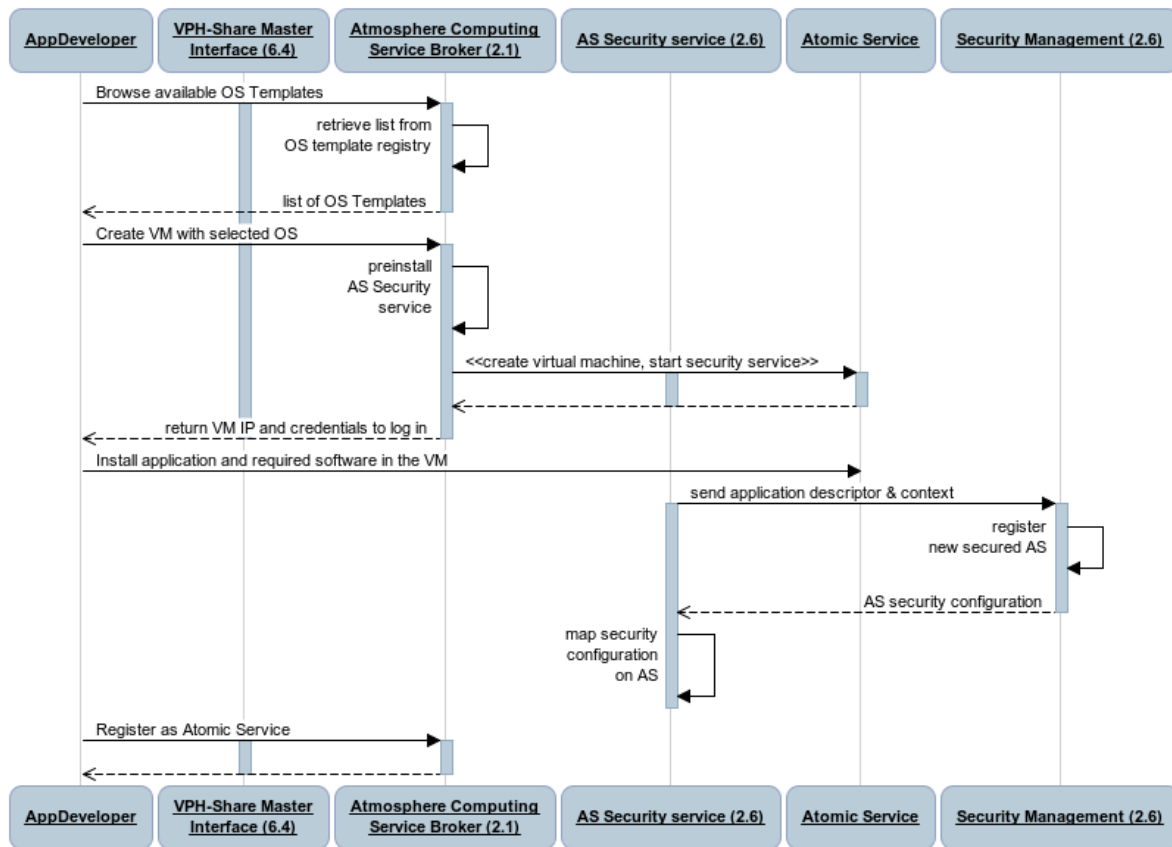
Method output attributes		
<i>Name</i>	<i>Type</i>	<i>Description</i>
<i>Result</i>	String	
Dependency with other components	This method is invoked internally by the Security Agent	

4.6.1.2.4 Security Token Service

Method Name	requestContextToken	
Method Definition/Description	This method requests a security token from a given subject's context.	
Method input attributes		
<i>Name</i>	<i>Type</i>	<i>Description</i>
<i>Context</i>	Set of Attribute identifiers and their values	Attribute-value pairs of the security context of the service's request (user's claims)
Method output attributes		
<i>Name</i>	<i>Type</i>	<i>Description</i>
<i>Token</i>	Token with the user's attributes and their values	A SAML/X.509 token with the attribute-value pairs of the Subject.
Dependency with other components	This method is invoked internally by the Security Agent	

4.6.2 Creation and Instantiation of Virtual Appliances

The Security Agent will already be deployed in the OS template, but not properly configured nor started. Thus, the first action to take will be to send the security configuration to the virtual machine, together with a script that will use it to start the services and provide them with the necessary artefacts. The Security Management services will compose the security configuration based on the requirements expressed in the Master Interface by the application provider. If the application provider does not have any specific constraint to add, a default configuration (defined and maintained by the system administrator) will be deployed instead.



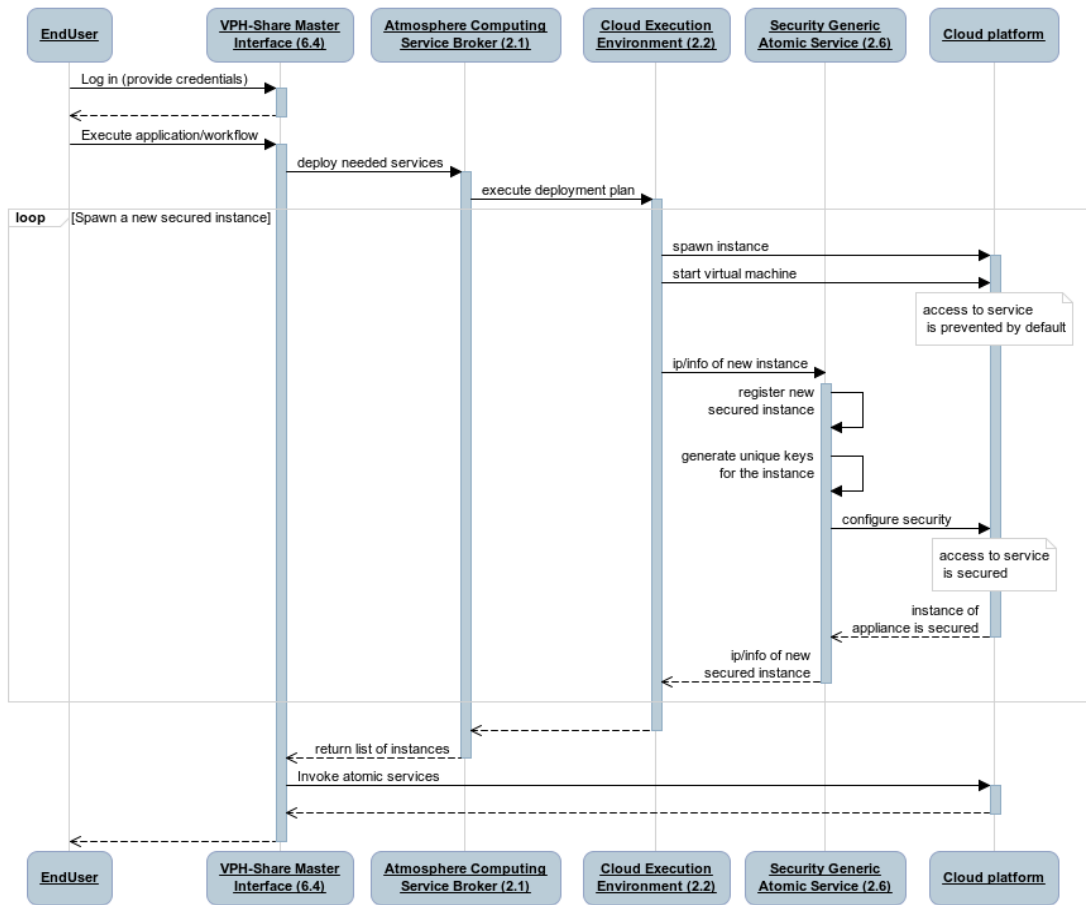
www.websequencediagrams.com

Figure 41: Securing Atomic Services

Once the application provider has finished installing and configuring the Atomic Service, the Security Management is handed with the necessary service descriptors to compose a security profile for the Atomic Service.

The Atomic Service in this Virtual Appliance will NOT be accessible to consumers; its ports will be protected. This Virtual Appliance is meant to be an image that can be replicated to create usable instances, but not usable itself.

The Security configuration of the Virtual Appliance is used for its instances, only taking into account some environmental changes (namely, the IP address at which the instance is exposed).



www.websequencediagrams.com

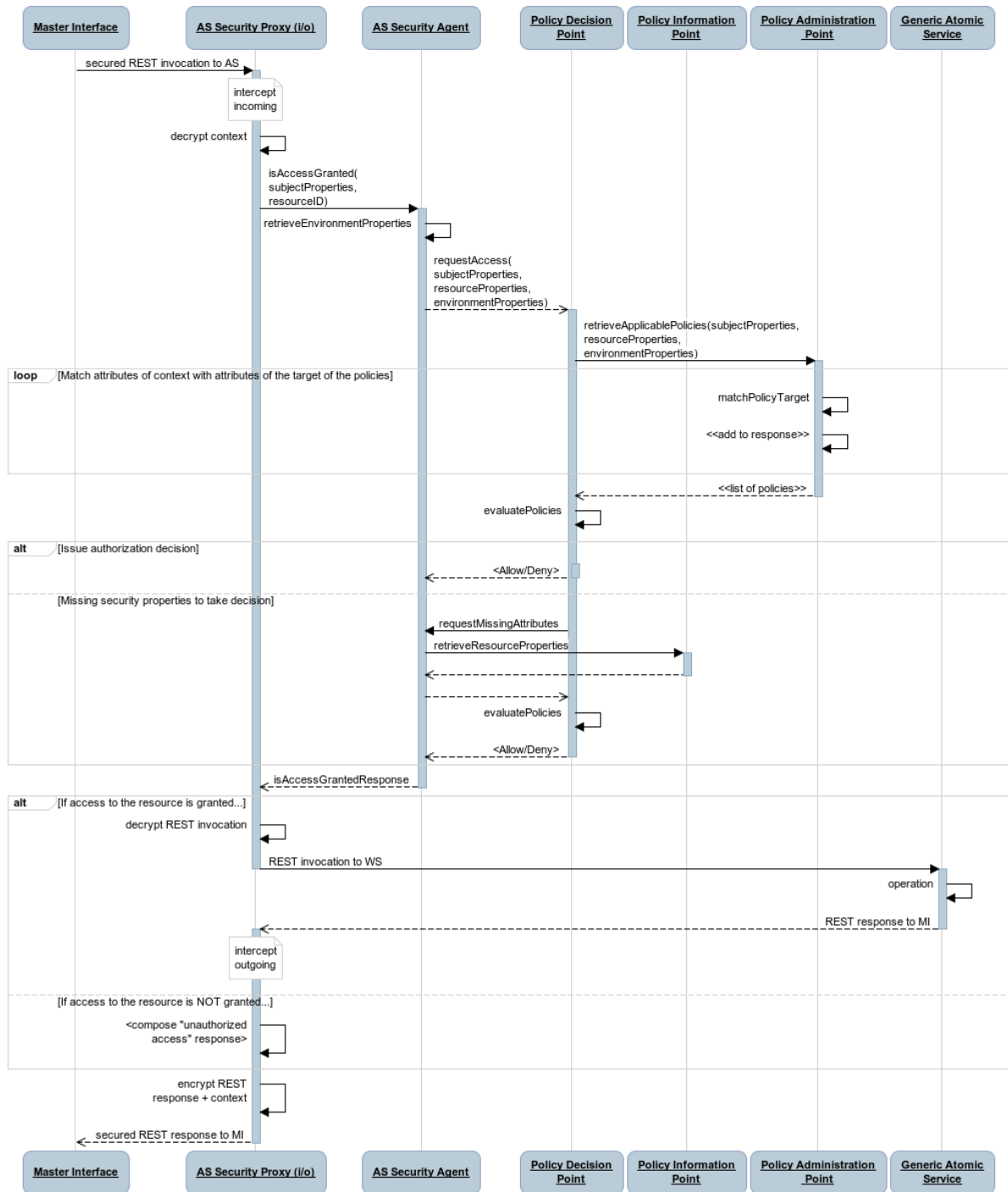
Figure 42: Spawning a Secure Atomic Service Instance

Each instance is provided with a pair of keys: private key (kept securely in the Virtual Appliance) and public key (which is kept in the persistence layer, and made available to any other VPH-Share service by the Security Management component).

4.6.3 Invocation of Atomic Service

The mechanism for easier and automated trust establishment leverages the use of the Atomic Services to external entities, allowing us to define a framework with the needed security and trust aspects for a reliable operation.

For the authorisation of requests, we will follow the model proposed by the XACML TC (Technical Committee). We will adapt this model to cover the several use cases in which Context properties and Policies must be distributed and combined in the Security Agents.



www.websequencediagrams.com

Figure 43: Security Agent – Authorizing a request to an Atomic Service

4.6.4 Policy Administration Point

The main functionality of the Policy Administration Point is to provide the Policy Decision Point (PDP) with the policies that are applicable to the context of the request that has to be evaluated.



The Policy Administration Point (PAP) is the component that will manage the XACML Objects, identify them and store them. Conditions and constraints must be applied to a Context that is being currently recognised by the PAP. That is to say that we will only accept policies (or policy updates) that correspond to situations that can be handled by the Security Agent to which the policy has been distributed.

Once a Policy is accepted and identified, it still has to be translated to a corresponding XACML PolicySet. This implies deciding how the new Policy will be combined with the already available policies.

The following case describes the relationship between the Authorisation Handler, the Policy Administration Point, the Policy Information Point and the Policy Decision Point. The request/response protocol used by these components to communicate one with the other are all based on XACML messages.

When the Security Proxy intercepts a request for a protected service, it sends it to the Security Agent, which rewrites it in XACML terms, as an Authorisation request. In order to know which policies apply to the specific context of the Request, it sends a Query to the PAP, which carries the Target of the request. With this Target, the PAP is able to locate the applicable policies, which are indexed by Target. But in order to carry on this location, it needs to know all the values of the attributes related to the Request Context, as policies may be stored using semantics that are not the same as the semantics used in the Request. Actually this is the key feature of our solution; it allows different expressions of the same access control conditions. So, the PAP queries the Policy Information Point to retrieve all the attributes related to the entities of the Target, and their values.

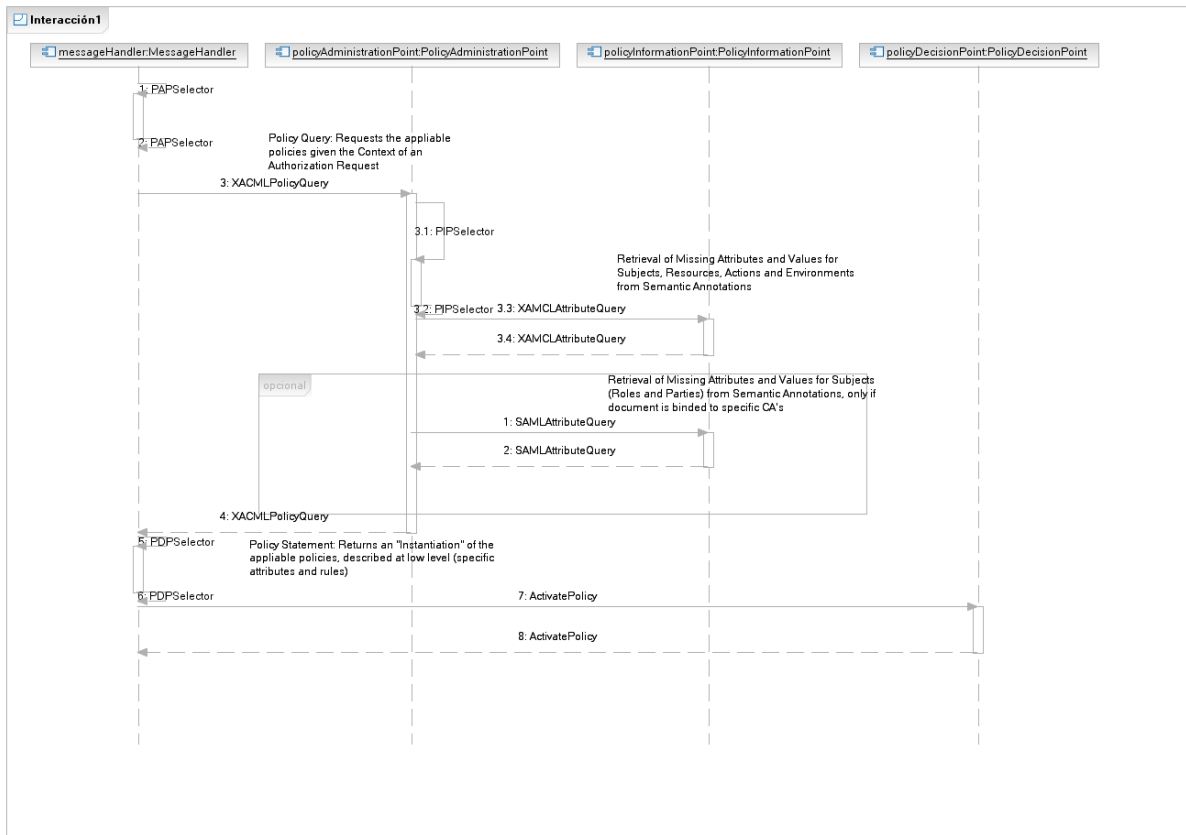


Figure 44: Activation of PDP policies.

With these attributes, the PAP is able to rewrite the Request Context into a Context which is expressed in the same terms as the policies. It then delivers the rewritten Context and the relevant policies to the Authorisation Handler, which then has everything it needs to send an AuthDecisionRequest to the PDP.

4.6.5 Policy Information Point

In XACML, attributes describing resource, subject, action and environment are basically labels with no semantics. Ontologies can be used to give formal semantics to attributes, which can be exploited in order to lower the complexity and ease the maintenance of XACML policies.

This decoupling between the definition of attributes and their relationships (in ontologies), and access control rules (in XACML policies) also allows an easier integration of external attribute authorities and mediation between attributes issued by different authorities. If users can present attributes coming from two different attribute authorities, the respective ontologies can be imported and a mapping established with the attributes used in XACML policies. If then one of these authorities changes something in the attribute assertions it issues, or a third attribute authority is added, this needs to be reflected only in the ontologies, without the need to modify the XACML policies. In turn, XACML policies can be written taking into account only the concepts (in terms of attributes) that are relevant for their purposes, without the need to handle unnecessary attribute complexity.



For example, if the attribute authority a1 can issue the “a1:Developer” attribute, while the authority a2 can issue the “a2:Programmer” and “a2:SeniorProgrammer” attributes, but all of them, for the purposes of the access control policies, can be considered equivalent to a single one (which we indicate as “:Developer”), the following can be added to the ontology, and then just the “:Developer” attribute be used in the XACML policy:

```
a1:Developer rdfs:subClassOf :Developer.  
a2:Programmer rdfs:subClassOf :Developer.  
a2:SeniorProgrammer rdfs:subClassOf :Developer.
```

It is important to note that, in a certain sense, the attribute ontologies become part of the policies. In fact, even if they are not strictly speaking part of the XACML policies, they can determine the result of policy evaluation. For example, if an XACML policy allows access to some resource to subjects with the “:Developer” attribute, but in the ontology we say that

```
:Employee rdfs:subClassOf :Developer.
```

then subjects with the “:Employee” attribute will be granted access (which is probably not the desired behaviour), because the PIP (Policy Information Point) will return the additional “:Developer” attribute for anyone with the “:Employee” attribute. For this reason, it is important to consider the issue of where the ontologies used by the PIP are stored, and of who can modify them. From the XACML point of view, this likely means to involve the PAP (Policy Administration Point).

4.6.6 Auditing and Logging

The Security Audit module and the Security Logging module are based on XDAS (Distributed Audit Services), which is an open standard for a system to audit an entire security domain (not just a single authentication domain) for any security-relevant event, specifically access control issues when managing protected resources.

XDAS provides a generic event taxonomy that makes it easy for both application providers and system administrators to understand the meaning of a given generic event. Filters and triggers to specific actions can be defined for these generic events.

We use an implementation of the full XDAS specification, OpenXDAS (53). In this section, we are not describing how this implementation is performing its functionalities internally, as we will be using its open libraries without getting into the audit and logging details. Instead, we will focus on which are its uses within VPH-Share.

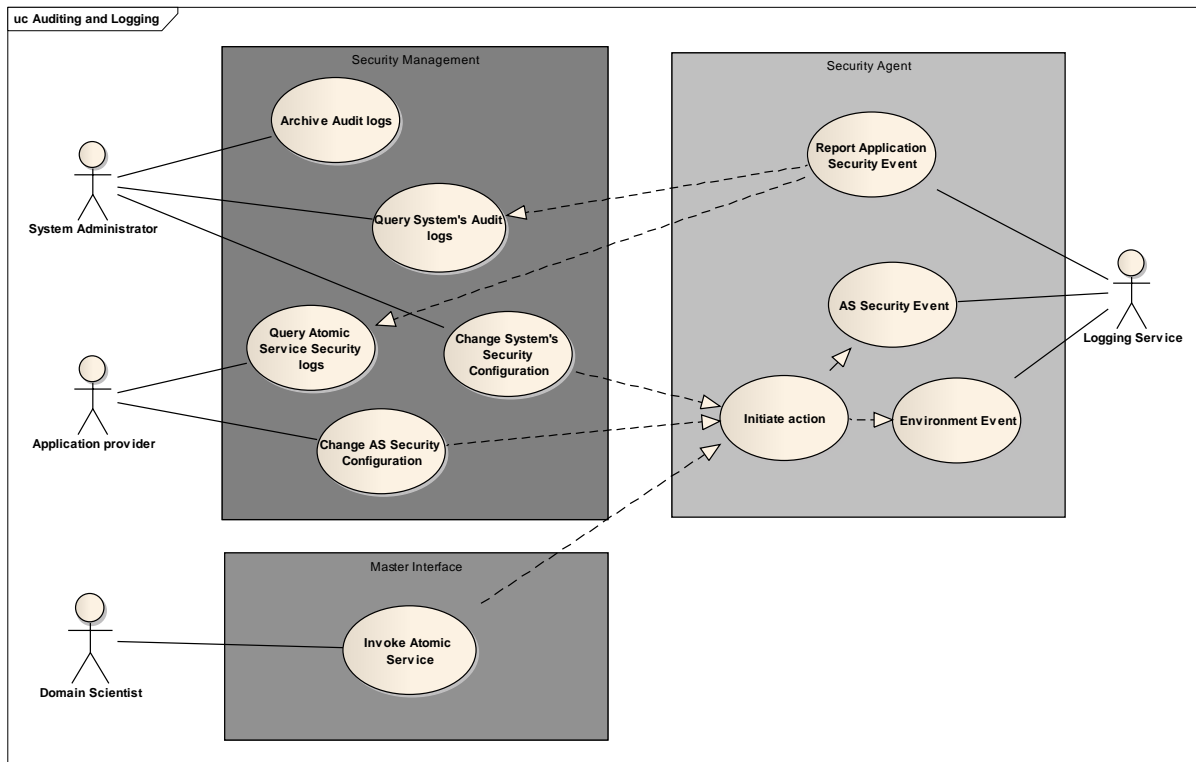


Figure 45: Security Audit – Use case for both the Management and the Agent’s event logging

The XDAS provides a set of primitives only, which are used by audit applications. Our implementation supports the following actions via APIs:

- 🌐 submission of audit events of an application
- 🌐 importing audit events from a running audit service
- 🌐 providing pre-selection criteria for both the event submission and event import to invoking applications
- 🌐 defining audit event filters that provide basic forms of event reactions such as log, alarm or custom action, which can be then performed by an audit analysis application

5 IMPLEMENTATION METHODOLOGY

The pace of implementation of WP2 tools is determined by the deadlines for successive WP2 deliverables, as expressed in the Technical Annex. Moreover, it is important to respect the deadlines for publication of Project results and delivery of internal components to collaborating Work Packages (and to end users) for testing and exploitation. Given these assumptions it seems natural to institute an iterative development cycle, with releases corresponding to WP2 milestones and smaller “agile” interim releases approximately every 2-3 months. This development methodology would result in frequent prototype releases, which can be tested both by representatives of WP2 tasks and by user groups, thereby generating further input for subsequent phases of implementation and deployment (possibly including fresh requirements). It will also lend itself to enhanced cooperation with external WPs (with which WP2 shares common interfaces) and with affiliated projects, such as



P-Medicine – in this way, we should always be able to showcase the results of our current work, rather than relying on the “last available” prototype release, which can be many months old (54).

According to the Technical Annex, the following important milestones are applicable to WP2:

- MS1: VPH-Share Infostructure Services – Architectural Design completed (M12)
- MS3: VPH-Share Service Bundle 1 Alpha Release – 1st Prototype of Cloud (WP2) and Data (WP3) Services (M12)
- MS7: VPH-Share Service Bundle Beta Release – Cloud (WP2), Data (WP3), Semantic (WP4), User Access (WP6) (M24)
- MS11: VPH-Share Service Bundle Candidate Release – Cloud (WP2), Data (WP3), Semantic (WP4), Access (WP6) (M36)
- MS12: Full integration of four flagship workflows with VPH infostructure (Candidate Release (M42)
- MS13: Comprehensive collection of data sources accessible through Candidate Release (M42)
- MS15: VPH-Share Service Bundle Maintenance Release – Cloud (WP2), Data (WP3), Semantic (WP4), Access (WP6) (M48)
- MS16 VPH-Share White Paper Released (M48)

As the first prototype of WP2 tools will need to be produced by Month 12, in time for the initial Project review, we intend to focus on a selection of core components required to implement and present a sample use case involving the use of the VPH-Share infrastructure. This goal will drive the implementation and development effort in the second half of Year 1 of the Project.

We envision the following components as being ready for deployment in the first prototype release of WP2 tools:

- Atmosphere resource management layer: rudimentary manipulation and registration of Atomic Services
- Atmosphere Internal Registry: initial data schema (conforming to the provisions established in this deliverable) and populated with sample data
- A selection of sample Atomic Services, registered and managed via Atmosphere, capable of being deployed on private Cloud resources belonging to the participating institutions
- A test-bed setup where such services may be instantiated and accessed
- Rudimentary end-user and application programmer’s interfaces for the atomic services listed above

To ensure continued integration of WP2 tools, CYFRONET will host a code repository to which participating institutions will be required to commit their component code. This will enable us to conduct automated testing (with developer-provided test cases) as well as schedule releases of the entire WP2 software package, with all components discussed in Section 4. The results of the application development and testing process will be reported upon in corresponding WP2 deliverables (starting with D2.3). In addition, any procedural



issues encountered during development will be communicated to Project Management and reported upon in WP1 reports.

The following table briefly summarises the technologies which will be utilised while developing and deploying elements of the WP2 platform:

Technical task	Technologies used
Task 2.1: Cloud Resource Allocation Management	Dedicated WS providing an API and GUI extensions for Cloud resource management. Java OSGi bundles deployed in a Karaf container as a service host. Atmosphere Internal Registry (AIR) as a common persistence layer for all of WP2 (NoSQL storage with service overlay)
Task 2.2: Cloud Application Deployment and Execution	On-demand deployment of services through OpenStack/commercial Cloud managers integrated with Atmosphere. Monitoring facilities and a service proxy for intranets. GUI extensions for system administrators (see Section 2) and API extensions for workflow management tools (WP6).
Task 2.3: Access to High-Performance Computing Infrastructures	Application Hosting Environment facilitating access to computing grids, with a Web Service overlay and API extensions for workflow management tools (WP6).
Task 2.4: Access to Large Binary Data in the Cloud	Service-based management of Cloud storage, compatible with popular distributed storage mechanisms. Ability to mount federated storage as element of the local file-systems on VPH-Share Atomic Service Instances. GUI extensions for browsing and management.
Task 2.5: Data Reliability and Integrity	Automatic monitoring of designated datasets, in terms of their availability and integrity; ability to set replication/verification policies (stored in AIR). Service backend (deployed and managed as a generic VPH-Share Atomic Service) with API and GUI extensions.
Task 2.6: Security	Attribute-based security policy with authentication mechanisms derived from VPH community portals. Security proxies on each VPH-Share Atomic Service Instance enabling interception of calls and user authorisation with the use of an external Policy Decision Point.

6 CONCLUSIONS

This document constitutes a starting point for the development and implementation work on WP2 components as well as on interfaces between WP2 and other Work Packages of the Project. Having agreed upon the architecture of WP2, its role and place in the Project architecture, as well as on the specific features and interconnections between WP2 modules we may now commence with development of an initial prototype. To this end, Work Package



2 has secured collaboration with relevant representatives of other Work Packages, especially in the following fields:

- provisioning of storage and computing resources for development and testing purposes (cross-WP effort)
- access to application components and datasets belonging to (or generated by) the VPH-Share flagship workflows (WP5)
- discussions regarding the preferred technologies for user interface development and integration (WP6)
- interfaces between internal WP2 components and the VPH-Share data management infrastructure (WP3)

As of the date of publication of this deliverables, WP2 representatives were actively involved in collaboration with other Work Packages (including end users of the Project's application workflows) and we foresee that this collaboration will continue throughout the development lifecycle.

The upcoming six-month period of the project (Months 7-12) will be devoted to implementing initial versions of WP2 tools (as remarked above) and developing an integrated prototype cloud platform, which is expected to be produced in time for the first yearly review of VPH-Share. Following the delivery and assessment of this prototype, we intend to meet with Project users and collaboratively decide upon the set of features, which will become part of the second prototype due by Project Month 24 (Milestone MS7). This collaboration, as well as its outcome, will be reported upon in the upcoming WP2 deliverables and in deliverables issued by other WPs (where relevant).

7 REFERENCES

1. VPH-Share Project Consortium. VPH-Share Annex I - Description of Work..
2. VPH-Share Project Consortium. D2.1: Analysis of the State of the Art and Work Package Definition..
3. Richardson T, Stafford-Fraser Q, Wood KR, Hopper A. Virtual Network Computing. IEEE Internet Computing. 1998 Jan-Feb: p. 33-38.
4. Fourer R, Gay DM, Kerningham BW. AMPL: A Modeling Language for Mathematical Programming, 2nd ed.: Duxbury Press; 2002.
5. Spellucci P. A SQP Method for General Nonlinear Programs using Only Equality Constrained Subproblems. 1993: p. 448.



6. CHOCO Solver. [Online].; 2011. Available from: <http://www.emn.fr/z-info/choco-solver/>.
7. 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.
8. Merrill D. Mashups: the new breed of Web app. [Online].; 2009. Available from: <http://www.ibm.com/developerworks/xml/library/x-mashups/index.html>.
9. OSGi Alliance homepage. [Online]. Available from: <http://www.osgi.org>.
10. Apache Karaf OSGi container. [Online].; 2011. Available from: <http://www.osgi.org>.
11. Apache Camel Integration Framework. [Online].; 2011. Available from: <http://camel.apache.org/>.
12. MongoDB NoSQL Document Store. [Online]. Available from: <http://www.mongodb.org>.
13. Blake Mizerany. Sinatra Ruby DSL. [Online]. Available from: <http://www.sinatrarb.com/>.
14. Phusion Passenger deployment module. [Online]. Available from: <http://www.modrails.com>.
15. Ubuntu homepage. [Online].; 2011. Available from: <http://www.ubuntu.com>.
16. TAS Foundation. Apache HTTP Server Project. [Online].; 2011. Available from: <http://httpd.apache.org>.
17. SoapLab2 homepage. [Online].; 2011. Available from: <http://soaplab.sf.net>.
18. Nginx wiki. [Online].; 2011. Available from: <http://wiki.nginx.org/>.
19. Nagios homepage. [Online].; 2011. Available from: <http://www.nagios.org/>.
20. JClouds. [Online].; 2011. Available from: <http://code.google.com/p/jclouds/>.



21. Zasada SJ, Coveney PJ. Virtualizing access to scientific applications with the Application Hosting Environment. *Computer Physics Communications*. 2009: p. 2513-2525.
22. Haidar AN. Audited Credential Delegation: A Usable Security Solution for the Virtual Physiological Human Toolkit. *Interface Focus*. 2011.
23. PRACE consortium. The Partnership for Advanced Computing in Europe (PRACE). [Online]. Available from: www.prace-project.eu.
24. Ciepiela E. Exploratory Programming in the Virtual Laboratory. In *Proceedings of the International Multiconference on Computer Science and Information Technology*; 2010. p. 621-628.
25. JBoss Community. jBPM. [Online].; 2011. Available from: <http://www.jboss.org/jbpm>.
26. The Quartz Scheduler. [Online].; 2011. Available from: <http://www.quartz-scheduler.org>.
27. OGSi consortium. The Open Grid Services Architecture Basic Execution Service. [Online].; 2008. Available from: www.ogf.org/documents/GFD.108.pdf.
28. BPMN Information Home. [Online].; 2011. Available from: <http://www.bpmn.org>.
29. JBoss Community. Drools. [Online].; 2011. Available from: <http://www.jboss.org/drools>.
30. WebDAV Resources. [Online].; 2011. Available from: <http://webdav.org>.
31. GridFTP documentation. [Online].; 2011. Available from: <http://globus.org/toolkit/docs/3.2/gridftp/>.
32. MacLaren J, McKeown M. HARC: A Highly-Available Robust Co-scheduler. In *Proc. of the 5th UK e-Schicne All Hands Meeting*; 2006.
33. Beckman P. SPRUCE: A System for Supporting Urgent High-Performance Computing. In *Grid-Based Problem Solving Environments (IFIP 2.5 Conference Proceedings)*.
34. Pickles SM. A Practical Toolkit for Computational Steering. *Phil. Trans. R. Soc. A*. 2005: p. 1843-1853.



35. Java. [Online].; 2011. Available from: <http://www.java.com>.
36. JBoss Community. Hibernate. [Online].; 2011. Available from: <http://www.hibernate.org>.
37. Van Nieuwpoort RV, Kielmann T, Bal HE. User-Friendly and Reliable Grid Computing Based on Imperfect Middleware. In ACM/IEEE conference on Supercomputing; 2007.
38. Apache Tomcat. [Online].; 2011. Available from: <http://tomcat.apache.org>.
39. Eclipse Jetty. [Online].; 2011. Available from: <http://www.eclipse.org/jetty>.
40. Restlet - RESTful web framework for Java. [Online].; 2011. Available from: <http://www.restlet.org>.
41. Common CLI. [Online].; 2011. Available from: <http://commons.apache.org/cli>.
42. Apache log4j. [Online].; 2011. Available from: <http://logging.apache.org/log4j/1.2/>.
43. Apache Axis2. [Online].; 2011. Available from: <http://axis.apache.org/axis2/java/core/>.
44. Oasis Web Service Security. [Online].; 2011. Available from: <http://www.oasis-open.org/committees/wss/>.
45. DAVFS2. [Online].; 2009. Available from: <http://savannah.nongnu.org/projects/davfs2>.
46. WDFS - WebDAV filesystem for FUSE. [Online].; 2007. Available from: <http://noedler.de/projekte/wdfs/>.
47. NetDrive. [Online].; 2011. Available from: <http://www.netdrive.net/>.
48. Apache Jackrabbit. [Online].; 2011. Available from: <http://jackrabbit.apache.org/>.
49. Onion WebDAV C++ library. [Online].; 2009. Available from: <http://sourceforge.net/projects/libonion/>.
50. PerlDAV. [Online].; 2011. Available from: <http://www.webdav.org/perldav>.



51. VPH-Share Project Consortium. D3.2.1: Data Management Platform Software Release v1..
52. Java Community Process. JSR-286 Portlet Specification. [Online]. Available from: <http://jcp.org/aboutJava/communityprocess/final/jsr286/index.html>.
53. Distributed Audit Services (XDAS) Project. Update of the Preliminary Specification. [Online]. Available from: <https://www.opengroup.org/projects/security/xdas/>.
54. Braude EJ. Software Engineering: an Object-Oriented Perspective New York: John Wiley & Sons; 2000.
55. Microsoft Inc. SPLA Program Benefits. [Online].; 2011. Available from: <http://www.microsoft.com/hosting/en/us/licensing/splabenefits.aspx>.
56. Rackspace OpenStack. [Online].; 2011. Available from: <http://www.rackspace.com/cloud/openstack/>.
57. Calcote J. OpenXDAS User's and Developer's Manual. ; 2009.

8 LIST OF KEY WORDS/ABBREVIATIONS

ACD	Audited Credential Delegation
AIR	Atmosphere Internal Registry
AMS	Allocation Management Service
API	Application Programmer's Interface
ASI	Atomic Service Interface
ASIP	Atomic Service Instance Proxy
AWS	Amazon Web Services
BLOB	Binary Large Object
CCU	Cloud Compute Unit
CEE	Cloud Execution Environment



CLI	Command-Line Interface
DHCP	Dynamic Host Configuration Protocol
DNAS	Dynamic Network Authentication System
DNAT	Dynamic Network Address Translation
DRAC	Dell Remote Access Card
DRI	Document Repository Interface
EBS	Elastic Block Storage
EC2	Elastic Compute Cloud
FASP	Fast And Secure Protocol
FC	Fibre Channel
FLASK	Flux Advanced Security Kernel
GFS2	Global File System 2
GSI	Globus Security Infrastructure
HAIL	High Availability and Integrity Layer
HPC	High-Performance Computing
HTTP	HyperText Transfer/Transport Protocol
IaaS	Infrastructure as a Service
iLO	Integrated Lights-Out
ILOM	Sun Integrated Lights Out Manager
IP	Internet Protocol
IPMI	Intelligent Platform Management Interface
iSCSI	Internet Small Computer Systems Interface
JBPM	Java Business Process Management suite
KVM	Kernel-based Virtual Machine
LAN	Local Area Network



LDRI	Logical Data Resource Identifier
LOBCDER	Large OBject Cloud Data storage fedERation
LXC	Linux Containers
MAC	Media Access Control
NAS	Network Attached Storage
NAT	Network Address Translation
NFS	Network File System
OASIS	Organisation for the Advancement of Structured Information Standards
OCCI	Open Cloud Computing Interface
OGF	Open Grid Forum
PaaS	Platform as a Service
PDP	Policy Decision Point
PDU	Power Distribution Unit
POR	Proof Of Retrievability
PXE	Preboot Execution Enviroment
QoS	Quality of Service
RAID	Redundant Array of Independent Disks
RDBMS	Relational Database Management System
REST	Representational State Transfer
RHEL	Red Hat Enterprise Linux
RMM	Remote Management Module
RSA	Remote Supervisor Adapter
RTT	Real Time Transfer
S3	Simple Storage Service
SaaS	Software as a Service



SAN	Storage Area Network
SDK	Software Development Kit
SLA	Service Level Agreement
SNAT	Static Network Address Translation
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SRB	Storage Resource Broker
SSH	Secure Shell
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	Universal Datagram Protocol
UDT	UDP-Based Data Transfer
UML	Unified Modelling Language
VHD	Virtual Hard Disk
VM	Virtual Machine
VMM	Virtual Machine Monitor (Hypervisor)
VO	Virtual Organisation
WAN	Wide Area Network
VENUS	Verification for Untrusted Storage
VNC	Virtual Network Computing
VRS	Virtual Resource System
WSRF	Web Services Resource Framework
XACML	eXtensible Access Control Markup Language