

Deliverable D3.3

Session Manager, runtime system and data layer: installation, integration and usage; description of interfaces to WP2, WP4 and WP5 – report and demonstration.

Project Start:	01-03-2006
Project Duration:	36 Months
Priority area	2.4.11
Contract No.:	INFSO-IST-027446
Website:	http://www.virolab.org

Due-Date:	31-08-2007
Delivery:	12-09-2007
Lead Partner:	CYFRONET
Coordinator	UvA, Prof. Dr P.M.A. Sloot
Dissemination Level:	Public
Status:	Final
Approved:	Quality Board, Steering Committee
Version:	1.1

Log of Document

Version	Date	Changes Summary	Authors
0.1	16/08/2007	Initial draft version	Tomasz Gubala
0.2	20/08/2007	Some sections filled in (conclusions pending)	Piotr Nowakowski
0.3	21/08/2007	Input to 2.1.4 section	Tomasz Bartyński
0.4	21/08/2007	Input to 4.2 section	Marek Kasztelnik
0.5	21/08/2007	Input to section 2.1.6	Joanna Kocot
0.6	22/08/2007	Input to sections 1 and 2.1.1, figs labels, references added	Tomasz Gubala, Eryk Ciepiela
0.7	22/08/2007	Section 2.1.5 extended and references added; summary section added.	Piotr Nowakowski
0.8	22/08/2007	DAS section added	Matthias Assel
0.9	23/08/2007	Input to sections 2.3.2 and 2.3.3	Michał Pelczar
0.10	23/08/2007	Adding geno2drs demo	Tomasz Gubala
0.11	27/08/2007	Structure changes, figure fix, some minor fixes	Tomasz Gubala, Marian Bubak
0.12	27/08/2007	Overview and runtime sections changed	Tomasz Gubala
0.13	27/08/2007	Executive summary overhauled; other minor corrections	Piotr Nowakowski
0.14	29/08/2007	QUaTRO demo added, some minor corrections	Tomasz Gubala, Bartosz Balis
0.15	30/08/2007	Corrections	Włodzimierz Funika, Tomasz Gubala
0.16	30/08/2007	Minor corrections in section 4.2	Marek Kasztelnik
0.17	30/08/2007	Executive summary updated	Piotr Nowakowski
0.19	02/09/2007	Further additions, corrections	Joanna Kocot, Eryk Ciepiela
0.20	03/09/2007	Experiment script example and description added	Tomasz Gubala
0.21	04/09/2007	Minor corrections in sections 5.2 and 5.3	Michał Pelczar
0.22	04/09/2007	Added data mining with Weka experiment	Maciej Malawski
0.23	04/09/2007	Added PROToS content	Kuba Wach
0.24	05/09/2007	Changes to DAS section and overall reviewing	Matthias Assel, Aenne Löhden
0.25	06/09/2007	PROToS executive summary added. Other minor changes.	Bartosz Balis
0.26	07/09/2007	Refinement of section 3.4	Tomasz Bartyński
0.27	07/09/2007	Added QUaTRO section	Kuba Wach
0.28	07/09/2007	MLA demo added, corrections	Tomasz Gubala, Alfredo Tirado
0.29	07/09/2007	Updated Weka experiment	Maciej Malawski
0.30	08/09/2007	Minor corrections in sections 3.2.4 and 6, adding of section 6.2.3	Michał Pelczar
0.31	08/09/2007	Minor corrections in section 4.2	Marek Kasztelnik

Version	Date	Changes Summary	Authors
0.32	09/09/2007	Minor changes	Eryk Ciepiela
0.33	10/09/2007	Added experiment planning demo and some other changes	Tomasz Gubala, Marian Bubak
0.34	10/09/2007	DAC description revised; other minor changes	Piotr Nowakowski
1.0	11/09/2007	Proofreading	Piotr Nowakowski
1.1	12/09/2007	Changed manuals list, few acronyms added	Tomasz Gubala, Marian Bubak

Table of Contents

1. EXECUTIVE SUMMARY.....	8
2. OVERVIEW OF THE FIRST VIRTUAL LABORATORY PROTOTYPE.....	10
3. DEMONSTRATION OF THE VIRTUAL LABORATORY PROTOTYPE.....	15
3.1. VIROLOGICAL ANALYSIS OF HIV VIRUS GENOTYPE.....	15
3.1.1. <i>Description of Experiment</i>	15
3.1.2. <i>User Group</i>	16
3.1.3. <i>Execution Inside the Virtual Laboratory</i>	16
3.2. QUERYING OVER PROVENANCE AND DATA.....	21
3.2.1. <i>Description</i>	21
3.2.2. <i>Intended user group</i>	22
3.2.3. <i>Requirements</i>	22
3.2.4. <i>Execution Inside the Virtual Laboratory</i>	22
3.3. ACQUIRING DRUG RESISTANCE INFORMATION REGARDING HIV VIRUS.....	25
3.3.1. <i>Description</i>	25
3.3.2. <i>User Group</i>	25
3.3.3. <i>Requirements</i>	26
3.3.4. <i>Execution Process</i>	26
3.4. DEMONSTRATION OF PLANNING OF AN EXPERIMENT.....	29
3.4.1. <i>Description</i>	29
3.4.2. <i>User Group</i>	29
3.4.3. <i>Setting up the work environment</i>	29
3.4.4. <i>Planning the data acquisition part</i>	31
3.4.5. <i>Sharing experiment with other developers</i>	32
3.4.6. <i>Planning the computation access part</i>	34
3.4.7. <i>Releasing the experiment plan for users</i>	36
3.5. DATA MINING FOR A CLASSIFICATION PATTERN.....	38
3.5.1. <i>Description</i>	38
3.5.2. <i>Intended user group</i>	39
3.5.3. <i>Technical Perspective</i>	39
3.5.4. <i>Requirements</i>	40
3.5.5. <i>Detailed code explanation</i>	40
3.5.6. <i>Running the experiment</i>	41
4. RUNTIME SYSTEM.....	44
4.1. GRIDSPACE ENGINE.....	45
4.1.1. <i>Implementation Description</i>	48
4.1.2. <i>Current Functionality</i>	50
4.1.3. <i>Planned Functionality</i>	51
4.2. GRID RESOURCES REGISTRY.....	51
4.2.1. <i>Implementation Description</i>	51
4.2.2. <i>Current Functionality</i>	53
4.2.3. <i>Planned Functionality</i>	53
4.2.4. <i>Deviations from the Design Document</i>	54
4.3. DOMAIN ONTOLOGY STORE.....	54
4.3.1. <i>Implementation Description</i>	55
4.3.2. <i>Current Functionality</i>	56
4.3.3. <i>Planned Functionality</i>	56
4.4. GRID OPERATION INVOKER.....	57
4.4.1. <i>Implementation Description</i>	57
4.4.2. <i>Current Functionality</i>	58
4.4.3. <i>Planned Functionality</i>	58
4.5. GENERIC DATA ACCESS CLIENT.....	58
4.5.1. <i>Implementation Description</i>	59
4.5.2. <i>Current Functionality</i>	60

4.5.3. <i>Planned Functionality</i>	60
4.5.4. <i>Deviations from the Design Document</i>	60
4.6. GRID APPLICATION OPTIMIZER.....	60
4.6.1. <i>Implementation Description</i>	61
4.6.2. <i>Current Functionality</i>	62
4.6.3. <i>Planned Functionality</i>	62
4.6.4. <i>Deviations from the Design Document</i>	63
5. DATA VIRTUALIZATION AND ACCESS.....	64
5.1. DATA ACCESS AND HANDLING.....	64
5.1.1. <i>Implementation Description</i>	65
5.1.2. <i>Current Functionality</i>	65
5.1.3. <i>Planned Functionality</i>	66
5.2. DATA RESOURCE DISCOVERY.....	66
5.2.1. <i>Implementation Description</i>	66
5.2.2. <i>Current Functionality</i>	66
5.2.3. <i>Planned Functionality</i>	66
5.3. SECURITY HANDLING (AUTHENTICATION, AUTHORIZATION AND CRYPTOGRAPHY).....	66
5.3.1. <i>Implementation Description</i>	67
5.3.2. <i>Current Functionality</i>	67
5.3.3. <i>Planned Functionality</i>	67
5.3.4. <i>Deviations from the Design Document</i>	67
5.4. NOTIFICATION, MESSAGING, MONITORING.....	68
5.4.1. <i>Implementation Description</i>	68
5.4.2. <i>Current Functionality</i>	68
5.4.3. <i>Planned Functionality</i>	68
5.5. DATA STORAGE AND LABORATORY DATABASE.....	68
5.5.1. <i>Implementation Description</i>	68
5.5.2. <i>Current Functionality</i>	69
5.5.3. <i>Planned Functionality</i>	69
6. PROVENANCE TRACKING SYSTEM – PROTOS.....	70
6.1. PROToS CORE.....	70
6.1.1. <i>Implementation Description</i>	70
6.1.2. <i>Current Functionality</i>	72
6.1.3. <i>Planned Functionality</i>	72
6.2. EVENT GENERATION TOOL.....	73
6.2.1. <i>Implementation Description</i>	73
6.2.2. <i>Current Functionality</i>	74
6.2.3. <i>Planned Functionality</i>	74
6.3. SEMANTIC EVENT AGGREGATOR.....	74
6.3.1. <i>Implementation Description</i>	75
6.3.2. <i>Current Functionality</i>	76
6.3.3. <i>Planned Functionality</i>	76
6.4. QUERY TRANSLATION TOOLS (QUATRO).....	77
6.4.1. <i>Implementation Description</i>	77
6.4.2. <i>Current Functionality</i>	77
6.4.3. <i>Planned Functionality</i>	78
7. LIST OF VIRTUAL LABORATORY MANUALS.....	79
8. SUMMARY.....	80

List of Figures

FIGURE -1: DIFFERENT LAYERS OF VIROLAB VIRTUAL LABORATORY.....	10
FIGURE -2: VIROLAB VIRTUAL LABORATORY ARCHITECTURE DIAGRAM.....	11
FIGURE -3: IMPLEMENTATION STATUS OF THE FIRST PROTOTYPE.....	12
FIGURE -4: VIRTUAL LABORATORY IMPLEMENTATION TIMELINE (FROM D3.2).....	13
FIGURE -5: MAIN PICTURE OF “FROM GENOTYPE TO DRUG RESISTANCE” EXPERIMENT...	16
FIGURE -6: VIROLAB PORTAL LOGIN SCREEN.....	17
FIGURE -7: PORTAL LOGIN DIALOG EXAMPLE.....	17
FIGURE -8: EXPERIMENT MANAGEMENT INTERFACE (EMI) MAIN VIEW.....	18
FIGURE -9: EXPERIMENT REPOSITORY PORTLET INSIDE EMI.....	18
FIGURE -10: EXPERIMENT CONTEXT PORTLET WITH EXPERIMENT DETAILS.....	19
FIGURE -11: EXPERIMENT EXECUTION ACKNOWLEDGEMENT MESSAGE.....	19
FIGURE -12: USER INPUT AND RESULT BROWSER PORTLET BEING IDLE.....	19
FIGURE -13: USER INPUT PORTLET REQUESTING USER INPUT.....	20
FIGURE -14: USER INPUT PORTLET RECEIVING USER INPUT.....	20
FIGURE -15: USER INPUT PORTLET ACKNOWLEDGING INPUT.....	20
FIGURE -16: EXPERIMENT OUTPUT.....	21
FIGURE -17: MAIN DEMONSTRATION COMPONENTS.....	21
FIGURE -18: DATA QUERY: INITIAL CONCEPT, SELECTION.....	22
FIGURE -19: DATA QUERY: CONSTRUCTION (1).....	23
FIGURE -20: DATA QUERY: CONSTRUCTION (2).....	23
FIGURE -21: DATA QUERY: CONSTRUCTION (3).....	23
FIGURE -22: PROVENANCE QUERY: SELECTING INITIAL CONCEPT.....	24
FIGURE -23: PROVENANCE QUERY: CONSTRUCTION.....	24
FIGURE -24: MLA PORTLET SERVICE COMPOSITION.....	25
FIGURE -25: PORTLET SELECTION WITHIN THE VIROLAB APPLICATION PORTAL.....	27
FIGURE -26: SELECTING A RULESET WITHIN THE MLA PORTLET.....	27
FIGURE -27: SPECIFYING A MUTATION-TYPE WITHIN THE MLA PORTLET.	28

FIGURE -28: SUBMITTING A MUTATION LIST TO THE DRS.....	28
FIGURE -29: RETRIEVING THE NEW RANKING RESULTS FROM THE DRS.....	29
FIGURE -30: VIROLAB EPE WELCOME SCREEN.....	30
FIGURE -31: THE NEW EXPERIMENT WIZARD.....	31
FIGURE -32: SHARE EXPERIMENT MENU OPTION.....	33
FIGURE -33: SELECT RESOURCES DIALOG.....	34
FIGURE -34: STARTING THE IMPORT EXPERIMENT WIZARD.....	35
FIGURE -35: STARTING THE RELEASE EXPERIMENT WIZARD.....	37
FIGURE -36: CHOOSING RELEASE VERSION.....	38
FIGURE -37: DATA MINING USING WEKA.....	39
FIGURE -38: EXPERIMENT EXECUTION MECHANISM WITHIN THE VIRTUAL LABORATORY RUNTIME.....	44
FIGURE -39. EXPLICIT SCRIPT EVALUATION REQUEST AND APPLICATION CODE PROVISIONING SCENARIO IT INDUCES.....	47
FIGURE -40. LOCAL FILE SCRIPT EVALUATION REQUEST TYPE AND THE APPLICATION CODE PROVISIONING SCENARIO IT INDUCES.....	47
FIGURE -41: REPOSITORY STAGED SCRIPT EVALUATION REQUEST TYPE AND THE APPLICATION CODE PROVISIONING SCENARIO IT INDUCES.....	48
FIGURE -42: MAIN COMPONENTS AND INTERFACES OF GSENGINE.....	49
FIGURE -43: CLASS DIAGRAM OF SUPPORTED TYPES OF EVALUATION REQUEST.....	50
FIGURE -44: GRID RESOURCES REGISTRY DECOMPOSITION	52
FIGURE -45: DOMAIN ONTOLOGY STORE DECOMPOSITION.....	55
FIGURE -46: GRID OPERATION INVOKER DECOMPOSITION DIAGRAM AND ITS EXTERNAL DEPENDENCIES.....	57
FIGURE -47: HANDLE CREATION AND QUERY PROCESS INSIDE DAC.....	59
FIGURE -48: GRID APPLICATION OPTIMIZER DECOMPOSITION DIAGRAM – CURRENTLY IMPLEMENTED GRAPPO MAIN COMPONENTS, CONNECTED TO OTHER RUNTIME MODULES: GOI AND GRR.....	62
FIGURE -492: DAS USE CASE OF A TYPICAL DATA ACCESS REQUEST AND OGSA-DAI INTERACTIONS.....	65
FIGURE -50: THE DECOMPOSITION DIAGRAM OF THE SEMANTIC EVENT AGGREGATOR....	75

1. Executive Summary

This document constitutes the description of the first prototype of ViroLab Virtual Laboratory software, developed after 18 months of the project duration and based on the design exemplified in the specification deliverable [D3.2]. A website featuring the properties and capabilities of this Virtual Laboratory can be found at <http://virolab.cyfronet.pl/>.

It describes the overall architecture of the ViroLab Virtual Laboratory prototype and provides an overview of all prototypes of individual functional components of the VL system. More detailed descriptions of all prototype components are included in user and developer manuals, available as appendices to this document and comprising the integral part of this deliverable.

As indicated, the prototype of the Virtual Laboratory incorporates some of the planned functionality of its constituent components. This functionality enables it to be applied to executing real-life experiments from the virology domain. More specifically, the prototype has been successfully demonstrated in the following areas:

- Experiment executing on the example of the *From virus genotype to drug resistance interpretation* process,
- Querying the historical and provenance information about virtual laboratory experiments using the PROToS provenance tracking system and its user interface,
- Assisting a clinical virologist with HIV therapy advice of the Drug Resistance System (based on the Retrogram set of rules), presented through its user interface MLA,
- Simple data mining and classifying, demonstrating possible usage of Weka [WEKA] from virtual laboratory.

Further information regarding the execution and processing of the selected demonstrative use cases can be found in subsequent sections of this document (particularly in section 3). More specifically, the GSEngine module is discussed, along with a number of components which together constitute the Virtual Laboratory – such as the Data Access Services, the Grid Resource Registry and the Experiment Management Interface. Furthermore, a section is devoted to the Experiment Planning Environment as well as to the PROToS provenance tracking system.

The main body of the deliverable is structured as follows:

- Section 2 presents a general overview of the ViroLab Virtual Laboratory and its architecture. This section is not intended as an in-depth description of VL components but rather as a brief summary, preceding subsections devoted to VL functionality elements.
- Section 3 presents how the Virtual Laboratory prototype can be applied to running ViroLab applications from several problem domains, and outlines the expected results.
- Section 4 describes the status of development of each Virtual Laboratory component, according to the architecture presented in D3.2 and other relevant documents.
- Section 5 presents data virtualization and access tools.

- Section 6 is devoted to the Provenance Tracking System (PROToS) and its functionality.
- Section 7 presents the list of manuals which should be treated as appendices to this deliverable.
- Section 8 contains closing remarks.

2. Overview of the First Virtual Laboratory Prototype

This document describes the first prototype of the ViroLab Virtual Laboratory based on the design document enclosed in deliverable [D3.2]. The implementation technology range follows the analysis of the state of the art documents [D2.1] and [D3.1]. Virtual laboratory development will continue until month 30 of the project (August 2008).

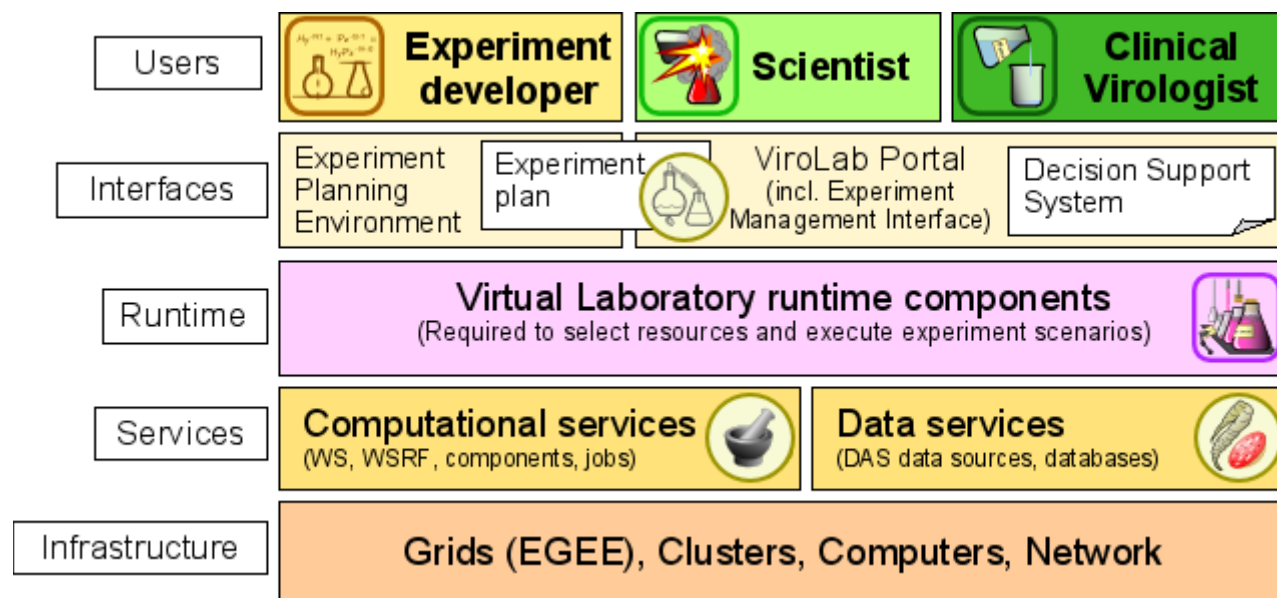


Figure -1: Different layers of ViroLab Virtual Laboratory.

Figure -1 shows in a conceptual way the various layers that constitute the virtual laboratory. While this *layered cake* type of visualization is rather abstract and the real architecture behind is far more complex, it shows the separation of levels well. In the upper part we have a set of users (with two classes introduced lately and another one explained later on). The users perform their tasks using a set of user tools that are grouped in the interfaces part: there are dedicated tools for each group. The centerpiece of the architecture, the unified runtime system, is the part that allows these various users and their tools to understand one another. What is more, this runtime layer server as a bridge to resources dispersed in the virtual laboratory: both data sources and computational services. Finally, these services run on physical equipment that are at their disposal (the last layer). There are multiple solution here that are supported by our virtual laboratory - among others it supports also large Grid computing testbeds (currently an adapter for the **EGEE** testbed through the LCG software is being added and we also plan to initiate work on potential addition of support for **DEISA** resources).

As one may see in Figure -1, the virtual laboratory serves as a natural point of integration of various tools, modules, protocols and interfaces into a single virtual space, where various types of users are able to (collaboratively, if they so choose) perform their tasks.

Figure -2 presents the modules of the ViroLab Virtual Laboratory – the architecture diagram was proposed in the design document and proved valid throughout the initial implementation stage. For completeness' sake, the diagram also contains some modules from Workpackage 2 (for a more detailed description of WP2 architecture see [D2.2]).

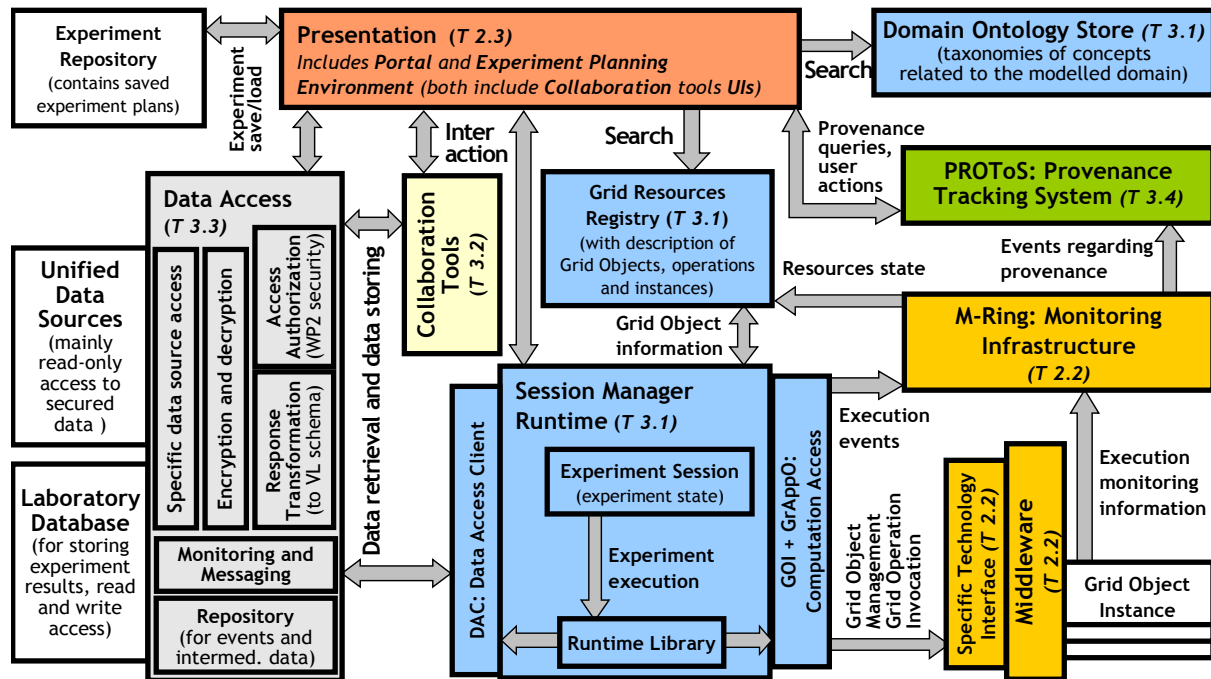


Figure -2: ViroLab Virtual Laboratory architecture diagram.

The main decomposition of the work effort inside Workpackage 3 is as follows:

- runtime components (task 3.1),
- collaboration tools (task 3.2),
- data access infrastructure (task 3.3),
- provenance tracking system (task 3.4).

The virtual laboratory runtime components provide all the functional blocks required for scientific in-silico experiments to run properly. The main execution task is performed by the VL runtime (also called the GridSpace Engine) which, apart from script interpretation capabilities, also provides access to distributed middleware (Grid Operation Invoker GOI) and to unified data sources (Data Access Client DAC). The interpreter is equipped with an execution optimization module (Grid Application Optimizer GrAppO) which contacts the registry server (Grid Resources Registry GRR) to find best matches for remote computation elements required by the experiment being executed. As the amount of available resources grows, the ontological taxonomies of the virology domain (stored inside the Domain Ontology Store DOS) help the experiment developer to find useful resources for a specific experiment. The exact description of the current implementation of this module of the laboratory is presented in Section 4.

The data access module (left-hand part in Figure -2) provides unified access to dispersed sources of medical data required by scientific experiments and applications. As the experiment developers expect the data to be in a uniform format despite the multitude of sources, the dedicated data access protocols together with the transformation module are responsible to provide a so-called

virtual ViroLab database to the end-users. This ensures that all information is visible as being stored inside one single data source while it is dynamically queried from different institutions. The data encryption library and the authentication and authorization modules are used to provide a completely secure environment for the scientists and medical users to work in, also securing the underlying data sources. The description of the current state of development within the data access task is provided in section 5.

The PROvenance Tracking System (PROToS) is responsible for tracking, storage and exposure of interfaces to query provenance data regarding the results of experiments. PROToS collects events from other components of the environment, mainly the GridSpace Engine and the runtime components of experiments. Those events are aggregated and transformed into an ontology-based description of experiment execution. The generic experiment ontology is the core ontology to represent an experiment's execution. In addition, concepts from other ontologies – data ontology and application ontology – are associated with the experiment ontology to enhance the semantic description. The ontology model is also a basis to query the provenance data. Query Translation tOols (QUaTRO) have been constructed to help construct complex queries over provenance, as well as data repositories. A detailed description of this module is presented in Section 6.

Implementation of various modules of the prototype was performed in parallel by the WP2 and WP3 teams, however thanks to the effort of the integration group (working inside WP4) all of them were properly integrated since the early phases of the project.

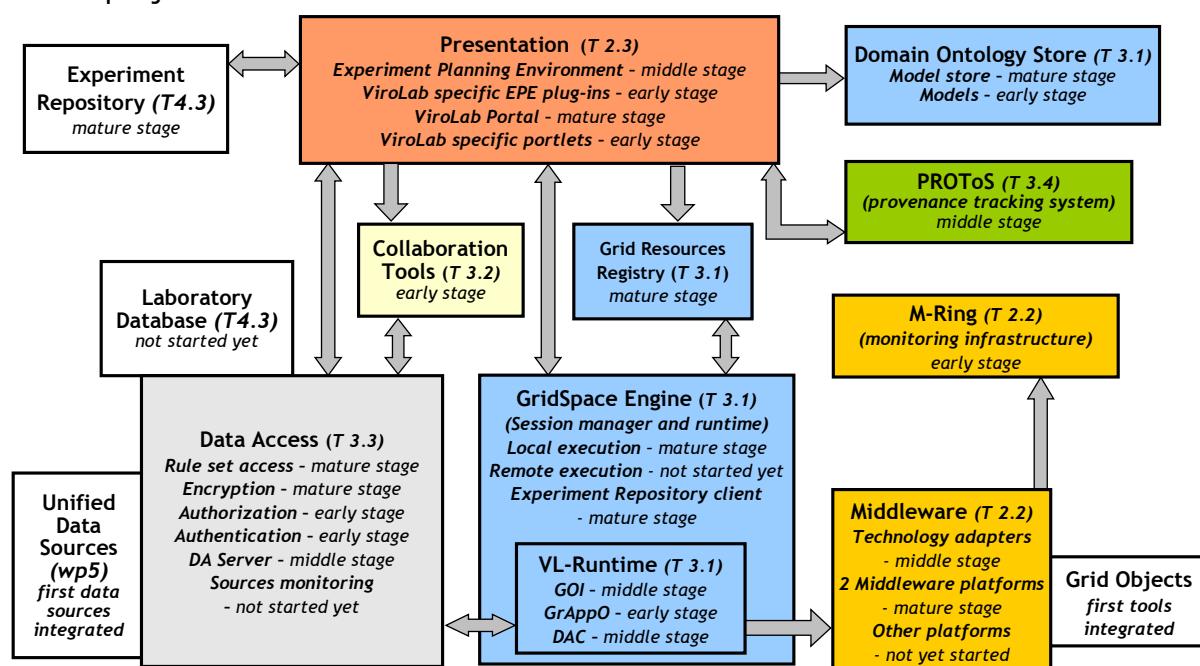


Figure -3: Implementation status of the first prototype.

Figure -3 provides an overview of the current state of development of individual packages and the level of integration with their neighboring modules. One may see that most of the parts are in at least early stages of implementation. The modules that are of critical value for the system have received more focus and are in middle or even mature implementation stages (this group includes local execution of experiments, ViroLab Portal, tools for experiment developers, data access infrastructure and some remote computation technologies). Some other

areas are relatively less developed as the implementation teams decided to focus on them at later stages of the project (collaboration tools, remote execution of experiments, laboratory data store for experiment results, monitoring and provenance tracking systems). The arrows in Figure -3 show which communication channels are already implemented and used in the prototype (although not necessarily in their final form).

The first prototype of the virtual laboratory is a coordinated release of several packages and the following table summarizes which modules build the prototype release. Those which are not standalone servers maintained by their development teams could be downloaded from the Virtual Laboratory website [VIROLAB-VL].

Module	Function
GSEngine	Executes experiments (runtime)
DAS	Service to access medical data from distributed DBs
GRR	Registry that publishes computational resources
DOS	Store that provides domain taxonomies for searching
EMI	Portlet to run experiments inside ViroLab Portal
PROToS	Provenance tracking server and portlet for users
EPE	IDE and plugins for experiment developers
MLA	Portlet for drug resistance information queries

The following sections give more detailed descriptions of the development stage of those virtual laboratory modules that are being developed within Workpackage 3.

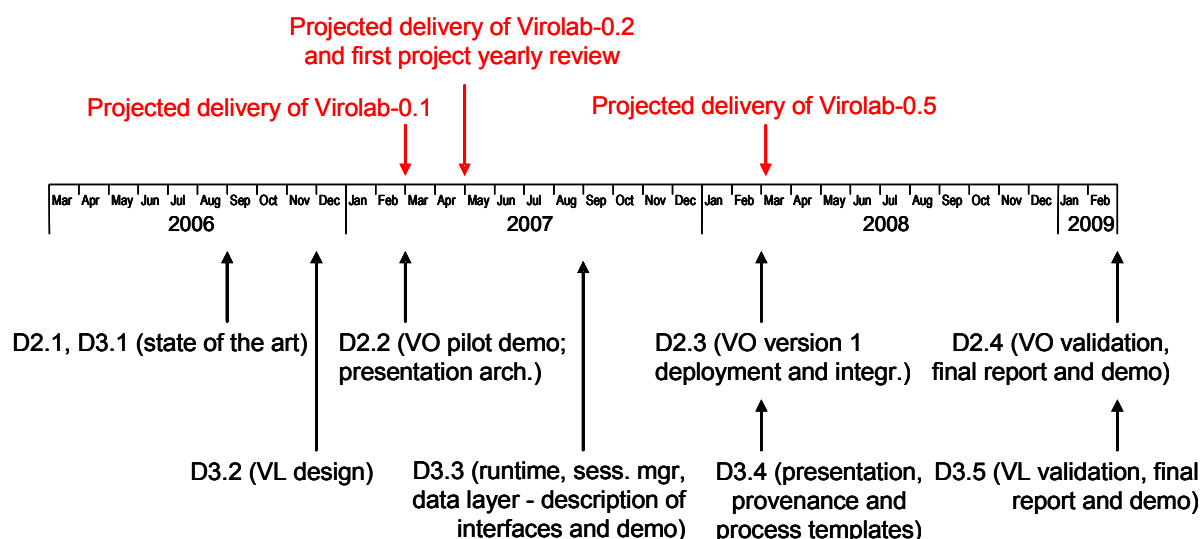


Figure -4: Virtual laboratory implementation timeline (from D3.2).

Figure -4 recalls the development timeline that was presented in Section 2.5 of the design deliverable [D3.2]. As one can see, the implementation is in the middle of the "Virolab-0.2" and "Virolab-0.5" milestones (see the D3.3 arrow in the lower part of the diagram). The two main milestones that have already passed ("ViroLab 0.1: Decision Support System" and "ViroLab 0.2: Genotype Analysis

and Simple Experiment Development Support”) were successfully achieved. This means that both the DSS for the drug resistance interpretation and the first virology experiments are already available in the first prototype of the virtual laboratory. While there are some deviations from the roadmap plan presented in section 2 of D3.2 (for instance the development of the provenance tracking system started in month 13 and the implementation of collaboration tools was postponed until better user feedback and requirements could be acquired) there are no visible indications of any critical changes needed in the design and implementation timetable.

3. Demonstration of the Virtual Laboratory Prototype

The demonstrations below show different aspects of the first prototype of the ViroLab Virtual Laboratory. In order to present a relatively wide range of laboratory capabilities, the demonstrations are designed to cover different functionalities of the system related to various types of potential users (from virology researchers to clinical practice users). The following sections contain demonstrations of:

- Executing an experiment regarding the *From virus genotype to drug resistance interpretation* process,
- Querying the historical and provenance information about virtual laboratory experiments using the PROToS provenance tracking system and its user interface,
- Assisting a clinical virologist with HIV therapy advice of the Drug Resistance System (based on the Retrogram set of rules), presented through its user interface MLA,
- Planning a new experiment to be executed within the ViroLab Virtual Laboratory,
- Simple data mining and classifying scenario, that demonstrating virtual laboratory capabilities in the area of data sets analysis.

3.1. Virological Analysis of HIV Virus Genotype

3.1.1. Description of Experiment

The experiment described below follows the short procedure from the samples of HIV virus nucleotide sequences (obtained from patient viral isolates) through some analysis of those sequences up to the list of rules regarding the resistance levels of those viruses to certain drugs.

The interpretation of HIV drug resistance into the susceptibility of the virus to particular drugs requires some preparatory several steps. Some of these steps have to be done manually. The preparatory measures involve:

- a **blood sample** should be taken from a patient (typically performed in a clinic)
- **isolation of genetic material** from the virus (conducted in a laboratory)
- **sequencing of genetic material** from the virus (involving a manual sequence verification)

Following these steps one obtains a set of valid genetic material of virus mutants and the information is stored in a database. This material is the input we need to start with the next, in-silico part of the experiment. Figure -5 shows its steps.

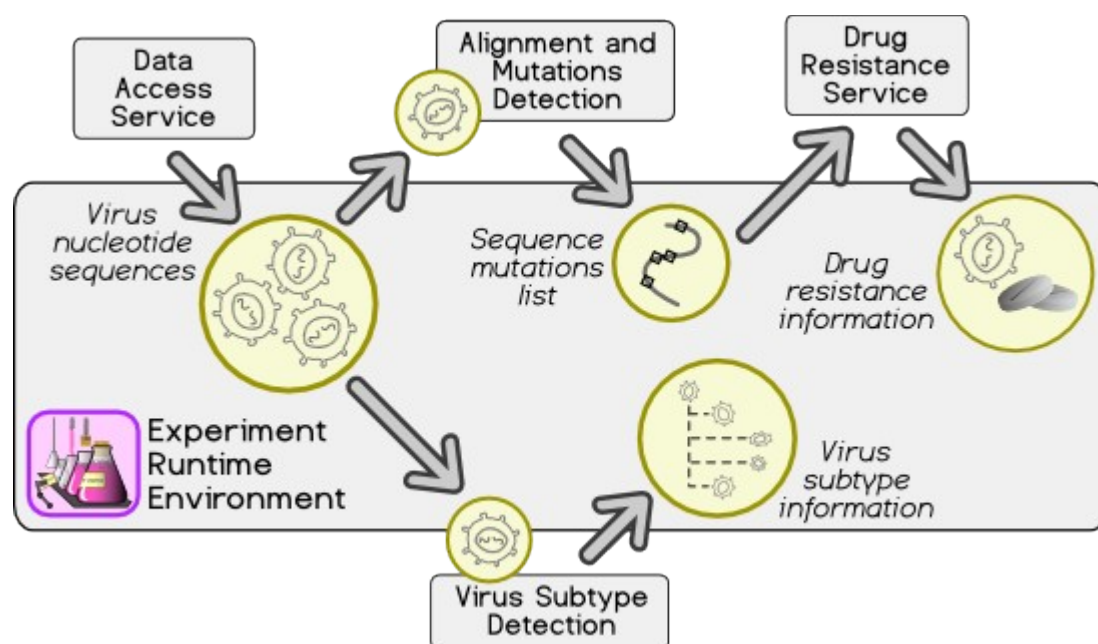


Figure -5: Main picture of “From Genotype to Drug Resistance” experiment.

In short the process runs like this (please follow the diagram in Figure -5):

- the **nucleotide sequences** of the HIV virus are obtained from a data source through the integrated, uniform protocol of DAS,
- these sequences are subject to both **alignment** and virus **subtype detection**,
- a very important outcome of the alignment step is the **list of mutations** a particular virus mutant has in comparison to some well-known reference sequences,
- these mutations are then uploaded to the **drug resistance system**, which returns the virus-to-drug susceptibility values as the final effect.

3.1.2. User Group

The application is intended for users with different requirements for the software. Generally, these users are either clinically or research-oriented. Clinically oriented users will always be interested in the virus’ drug resistance and subtype, but not necessarily in a detailed list of all mutations/substitutions. Conversely, researchers are not necessarily interested in drug resistance but would like to get the subtype and a list of all mutations/substitutions and nucleotide composition per codon. Thus, the experiment is potentially useful for a wide spectrum of users.

3.1.3. Execution Inside the Virtual Laboratory

User Login in the ViroLab Portal

The experiment starts with the ViroLab Portal. Since this document is not a thorough Portal guide, we will go through the login sequence very quickly, not much detailed. On the main ViroLab Portal page (<https://virolab.gridwisetech.pl/>)

the user follows the **Login** link in the top right corner. Afterwards, the main login screen appears (see Figure -6).

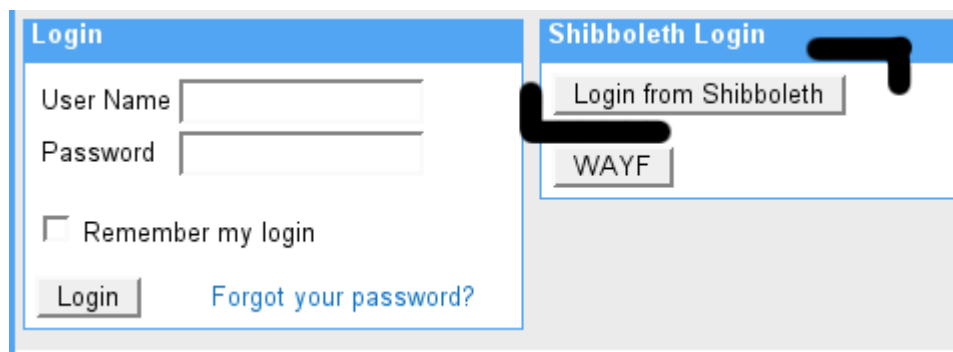


Figure -6: ViroLab Portal login screen.

If the user's home organization uses standard Shibboleth authentication protocols, the user will be faced with a typical web login window like the one in Figure -7. This may, however, look quite different if the security infrastructure at the user's home organization uses different means of authentication.

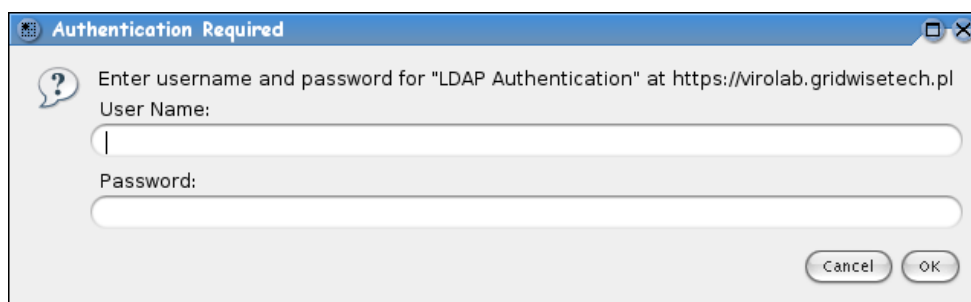


Figure -7: Portal login dialog example.

Whatever the login procedure, the user provides credentials.

Loading an experiment into the workspace

Following successful login, the user opens the Experiment Management Interface (**EMI**) tab inside the Portal - the one that contains the Experiment Management Interface portlets (the tab may also appear right away as the default one). The following text is not a guide on how to use EMI but only a step-by-step experiment execution scenario (for the guide itself please refer to [D3.3USR]).

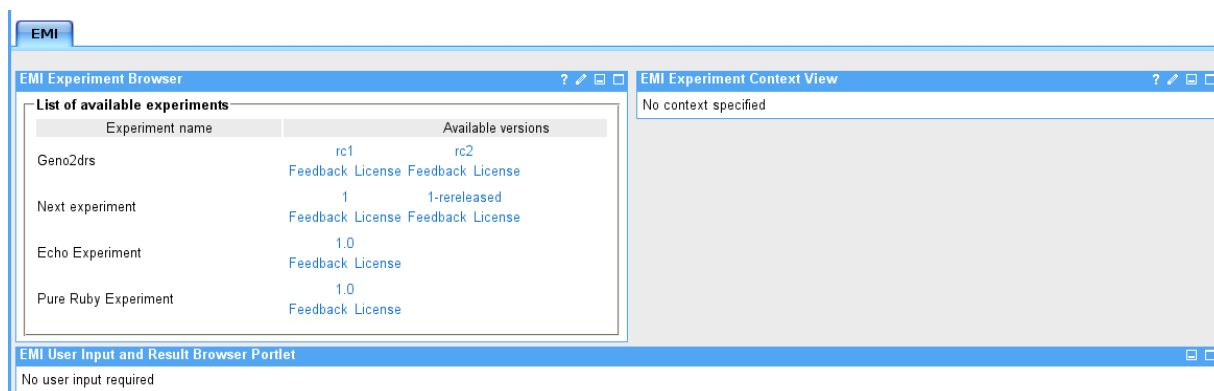


Figure -8: Experiment Management Interface (EMI) main view.

Figure -8 presents a screenshot of the overall EMI area. Please note that in your case the appearance could vary due to a different visual theme, yet the controls should look quite similar. In the left part you see the so-called **Experiment Browser** that shows which experiments are currently released and which versions are available inside the **Experiment Repository** (see also Figure -9). In the current case, the experiment to execute is called *Geno2drs* and has two releases from which the user chooses the later one (*rc2*).

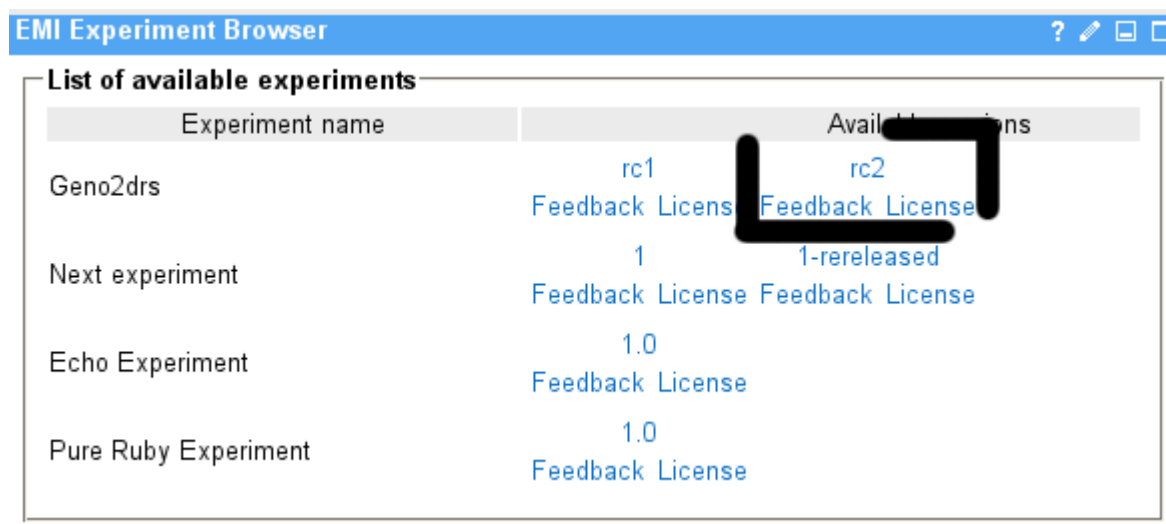


Figure -9: Experiment Repository portlet inside EMI.

This is done by clicking on the version name (the *rc2* label) in the **Experiment Browser** window. The user has to wait a few seconds for the experiment files to be fetched from the remote repository and then the right part of the display, called **Experiment Context** awakes. The content of the window is presented in Figure -10. It provides important, basic information (the upper **Experiment details** part) about the experiment itself, including its authorship, the owner rights and also a short description of the experiment. All these pieces of information were provided by the developer of the experiment.

The experiment is now loaded and ready for execution.

Experiment execution

EMI Experiment Context View

Experiment details

Experiment name	Geno2drs
Contact email	gubala@science.uva.nl
Organization name	ACC CYFRONET AGH
Experiment description	This experiment finds available HIV nucleotide sequences, align them with respect to a reference strain, then finds out mutations in an indicated region and, finally, runs ViroLab's Drug Ranking to learn about resistancy of the virus.

Experiment execution

Experiment version: Geno2drs_rc2 Execute experiment

Figure -10: Experiment Context portlet with experiment details.

The lower part of the **Experiment Browser** context window (see again Figure -10) features the **Experiment execution** part. Here, the user chooses exactly which version of the experiment should be used (the previously clicked *rc2* version is the default choice). It is enough to click on the *Execute experiment* button to start the experiment. If everything goes smoothly, an acknowledgment in form of an information message should appear right in the top part of the **Experiment Context** portlet (see Figure -11).

EMI Experiment Context View

✓ EXPERIMENT SUCCESSFULLY STARTED. PLEASE, AWAIT USER INPUT REQUESTS OR RESULTS.

Figure -11: Experiment execution acknowledgement message.

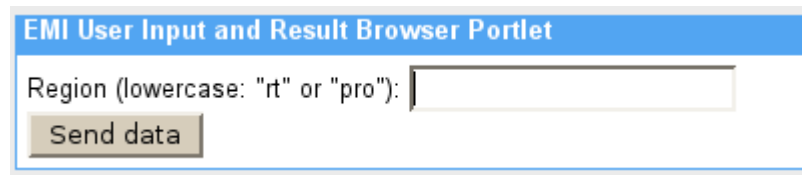
The experiment, after some startup activity, addresses the user again in order to gather required input. Precisely, the user needs to choose which protein (or region of the sequence) is of interest for him. Currently, it may either be protease ("pro") or reverse transcriptase ("rt"). When such a need for user input or decision arises, the **User Input and Result Browser** Portlet (which most of the time, in its ideal state, looks like the one in Figure -12) displays an input form.

EMI User Input and Result Browser Portlet

No user input required

Figure -12: User Input and Result Browser Portlet being idle.

A fresh input form and user-provided information are shown in Figure -13 and Figure -14 respectively.

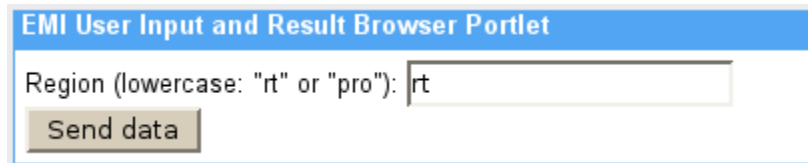


EMI User Input and Result Browser Portlet

Region (lowercase: "rt" or "pro"):

Send data

Figure -13: User Input Portlet requesting user input.



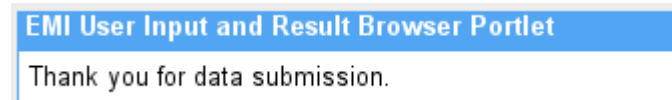
EMI User Input and Result Browser Portlet

Region (lowercase: "rt" or "pro"):

Send data

Figure -14: User Input Portlet receiving user input.

After the user chooses that the region of interest will be the reverse transcriptase, the choice is submitted to the pending experiment by pressing the *Send data* button. If the data transmission was successful, a simple acknowledgment information should appear in the window (see Figure -15).



EMI User Input and Result Browser Portlet

Thank you for data submission.

Figure -15: User Input Portlet acknowledging input.

Experiment result

After waiting for a while (both the alignment and the subtyping algorithms take some time to finish) the experiment finally returns its results to the user (see Figure -16 for a cropping of the most interesting piece of result data). Among others, it shows:

- what nucleotide sequence was the basis for processing,
- what mutations were detected in the interesting region of the sequence,
- what subtype was estimated for this particular virus mutant,
- and finally, what drugs were recommended by the decision support systems.

```
-----
Mutations obtained for region rt:
P1M I2L S3T P4Q E6G T7C V8T P9L V10N K11F V35T T39K S68G E122K D123N S162C K173I
Q174K D177E I178M Q207A R211S L214F K238R
-----
Let's check for the subtype of the sequence
-----
Virus subtype: HIV-1 Subtype A (01_AE)
-----
Now we run the drug resistance system for the list of mutations
-----
Rule EFV330 recommends level 1 drug efavirenz for use if no level 0 drug is available.
Rule DLV300 recommends level 2 drug delavirdine for use if no drug of level 0 or 1 is available.
Rule NVP310 recommends level 2 drug nevirapine for use if no drug of level 0 or 1 is available.
-----
Geno-to-drug resistance: finish
Ok
```

Figure -16: Experiment output.

This step concludes the experiment. As one may check by going back to Figure -5, all the stages of the *From genotype to drug resistance interpretation* process were covered.

3.2. Querying over Provenance and Data

3.2.1. Description

The purpose of this demonstration is to present how users can query repositories of data (biological/virological/medical databases) and provenance (experiment execution traces). The provenance system and data model are also indirectly presented. In this scenario, users such as medical doctors or researchers will use Query Translation Tools to construct and submit queries to underlying repositories of data and provenance. Query construction is user-oriented, thanks to ontology models of experiments, data, and applications, which serve as inter-lingua between users who are non-IT experts and tools.

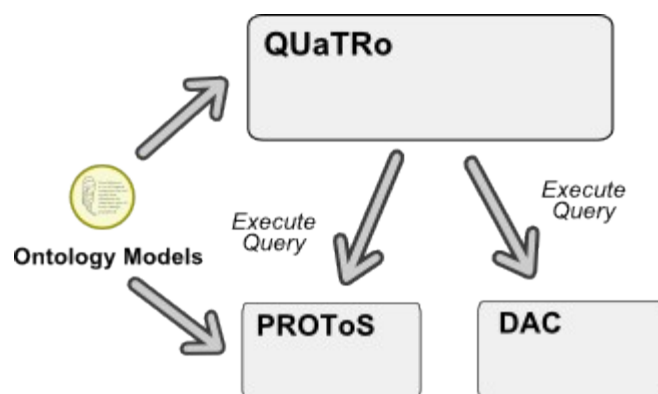


Figure -17: Main demonstration components.

3.2.2.Intended user group

The target user group includes non-IT experts (medical doctors, researchers) who need to query repositories of data or provenance, e.g. to obtain results to serve as input for a new experiment, or information on a previously executed experiment.

3.2.3.Requirements

The following components are involved:

- QUery TRanslation Tools (QUaTRo), which provide a graphical user interface to construct queries in an end-user oriented manner
- PROvenance Tracking System (PROToS), wherein provenance records are stored and which provides interfaces to query provenance data
- Ontology models of experiments, data, and applications
- Additionally, Data Access Client (DAC), to query databases

3.2.4.Execution Inside the Virtual Laboratory

Query 1: Querying over medical concept

The first query concerns medical and virological databases. The query begins with picking a starting concept which determines the domain of the query and, ultimately, the repositories to be queried. Let us suppose the user wants to find some HIV virus aminoacid sequences, which satisfy certain criteria. The query should begin with the *VirusAminoAcidSequence* concept (Figure -18).

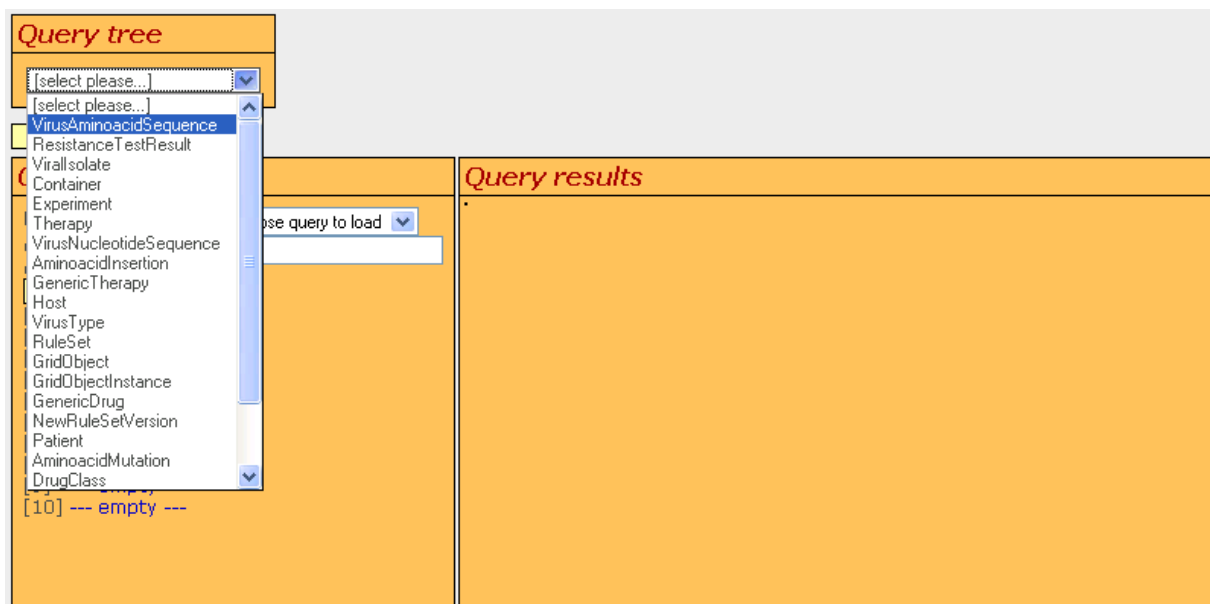


Figure -18: Data query: initial concept, selection.

Query 1: Selection of aminoacid sequence attributes

When the initial concept has been selected, the QUaTRo form is dynamically expanded to present all available relationships and related concepts of database attributes. In the latter case, the database itself is dynamically contacted to obtain a list of attributes as specified in the data model, and also the list of actual attribute values that exist in the current database records. Figure -19, Figure -20, and Figure -21 present the construction of the query. Note how logical *and* operators are used to impose multiple constraints on a given entity.

Figure -19: Data query: construction (1).

Figure -20: Data query: construction (2).

Figure -21: Data query: construction (3).

Query 2: Querying over provenance concept

The screenshot shows a web-based query builder interface. On the left, a 'Query tree' panel contains a list of concepts: Experiment, [select please...], VirusAminoacidSequence, ResistanceTestResult, Virallolate, Container, Experiment (highlighted), Therapy, VirusNucleotideSequence, AminoacidInsertion, GenericTherapy, Host, VirusType, RuleSet, GridObject, GridObjectInstance, GenericDrug, NewRuleSetVersion, Patient, AminoacidMutation, and DrugClass. Below this list, three items are marked as empty: [8] --- empty ---, [9] --- empty ---, and [10] --- empty ---. To the right of the list is a 'select please...' dropdown and an 'and' checkbox. Further right is a 'choose query to load' dropdown. The main area on the right is titled 'Query results' and is currently empty.

Figure -22: Provenance query: selecting initial concept.

The second query will select some data from the provenance repository. We shall retrieve some experiments with specific characteristics. Let us begin with the Experiment concept (Figure -22).

Query 2: Selection of experiment properties and related concepts

Similarly to the data query, the query form is dynamically expanded upon construction. The full query is shown in Figure -23.

The screenshot shows the 'Query tree' interface with a complex query constructed. The query is built using a series of 'and' conditions. The first condition is 'Experiment' with 'ownerLogin' set to 'JamesBond'. The second condition is 'inputData' with 'AminoacidInsertion' and 'AminoacidMutation'. The third condition is 'hasStage' with 'DataAccessCall'. The fourth condition is 'rt_insertion_codon' with 'insertion_codon'. The fifth condition is 'position' with '2'. The sixth condition is 'source' with 'Cyfronet'. The seventh condition is 'dataBase' with 'MedicalData'. The eighth condition is 'ontologyType' with 'Patient'. The ninth condition is 'birth_date' with '1981-02-14'. The tenth condition is 'sourceFile' with 'test'. A 'Launch query' button is at the bottom left. A dropdown menu is open on the right, showing a list of dates: 2006-09-07, 1988-09-07, 1981-02-14, 1982-04-21, and 1984-04-10.

Figure -23: Provenance query: construction.

3.3.Acquiring Drug Resistance Information Regarding HIV Virus

3.3.1.Description

This experiment allows a first version of portlet-based access to our ranking system. In this demo a user can use the Mutation List Analysis (MLA) portlet application to interact with a number of ranking systems, via the ViroLab application portal.

A user uses the MLA portlet to apply a procedure that uses samples of HIV virus nucleotide sequences and submits them to the ViroLab decision ranking system (DRS) that interfaces to a list of rulesets. These rulesets are applied to analyze the specific sequence resistance levels to certain drugs. The MLA portlet therefore serves as an interface between the back-end DRS by using a *DRS portlet container* that invokes the MLA service, which hosts the *MLA portlet* within the ViroLab application portal, as seen in Figure -24. The MLA portlet serves as the interface which gets invoked by the DRS back-end system in order for the user to interact with a number of ranking rulesets.

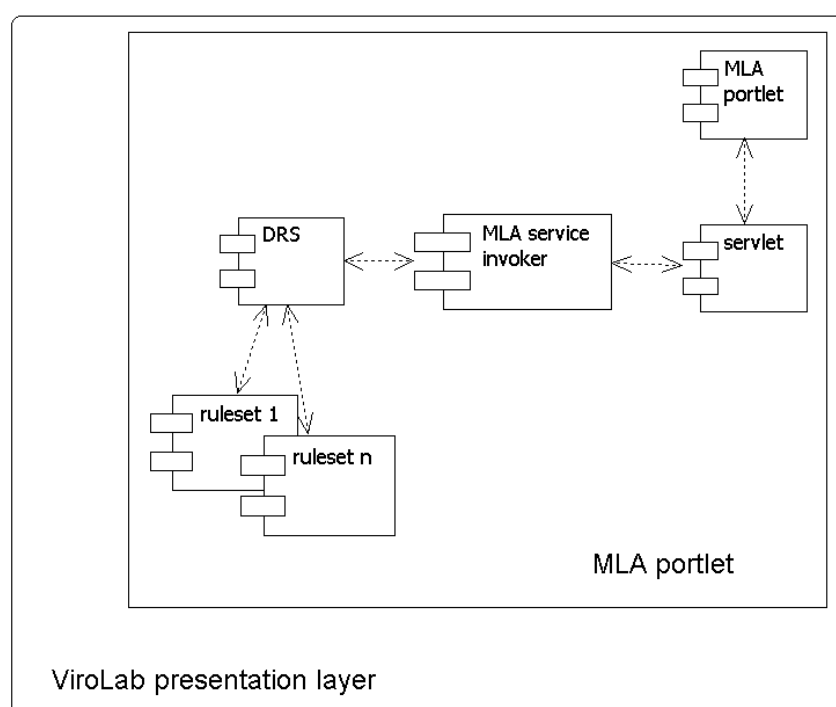


Figure -24: MLA portlet service composition.

The rulesets, mutation types and list of mutations can be entered into the user interface using text boxes and pull down menus implemented with Javascript, within the GridSphere portlet framework.

3.3.2.User Group

The intended user group for this demo is the medical user, which is one of the main users of the ViroLab Virtual Laboratory. These are individuals with a higher

level of education in biomedical fields (e.g., virology), and may have limited knowledge/experience with state of the art information and communication technologies.

3.3.3.Requirements

This demo is based on and hosted by the GridSphere technology framework, so in order to access it the user needs to have:

- a machine capable of running a graphical web browser (e.g., Pentium II or better with at least 256 MB of RAM),
- a network connection,
- a graphical browser such as Mozilla Firefox, Safari or MS Explorer, with JavaScript support and Java 1.4.2 or better.

3.3.4.Execution Process

This demo provides access to the basic capabilities of the new ViroLab ranking system, the DRS. The user first needs to access the Virolab application portal, in the standard way. After the user logs into the application portal:

1. The user select the DRS portlet tab, which contains the MLA portlet.
2. Once at the MLA portlet, the user selects a ruleset from the first drop-down list.
3. The user selects a mutation type from the second drop-down list.
4. The user types each mutation to analyse, upper case, separated by one or more spaces, and submits the mutation list for ranking.
5. Finally, the user scrolls down to get the new rankings.

Loading the DRS Portal

The medical user accesses the ViroLab application portal, using a web browser, at <https://virolab.gridwisetech.pl/> via the ViroLab Shibboleth framework. The user selects the decision ranking service (DRS) tab, which brings up the mutation list analysis portlet (Figure -25). Initially, the user finds as default values *Retrogram* selected as ranking system to use, and *Reverse Transcriptase* as mutation type.

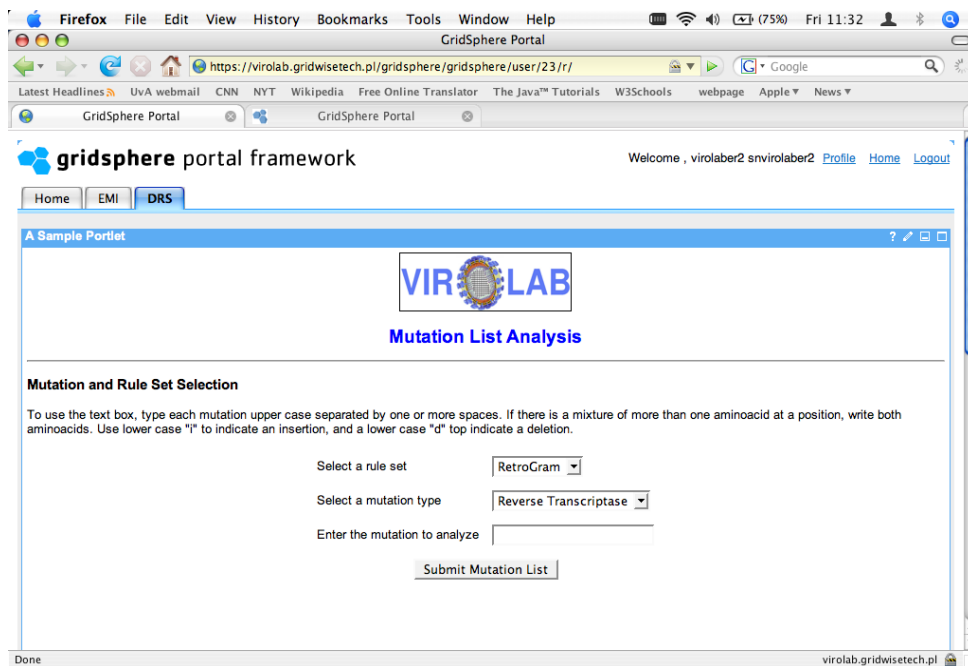


Figure -25: Portlet selection within the ViroLab application portal.

Selecting Ruleset and Mutation Type

The user selects a ruleset from the first drop-down list such as Retrogram, Rega, or Stanford (Figure -26), and the mutation type such as Reverse Transcriptase or Protease (Figure -27).

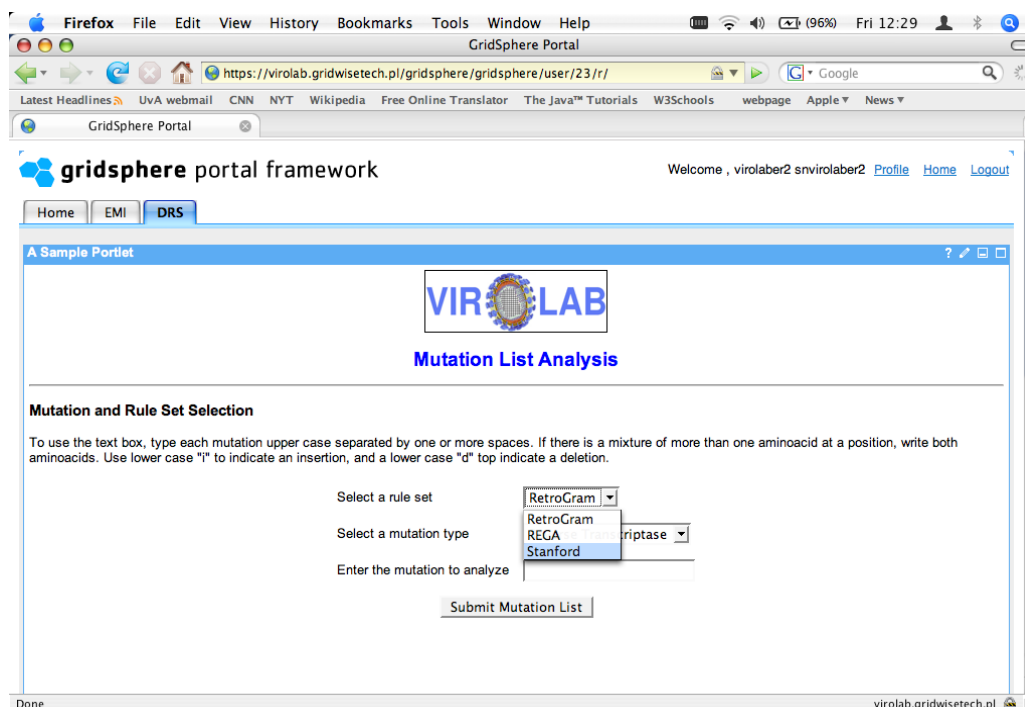


Figure -26: Selecting a ruleset within the MLA portlet.

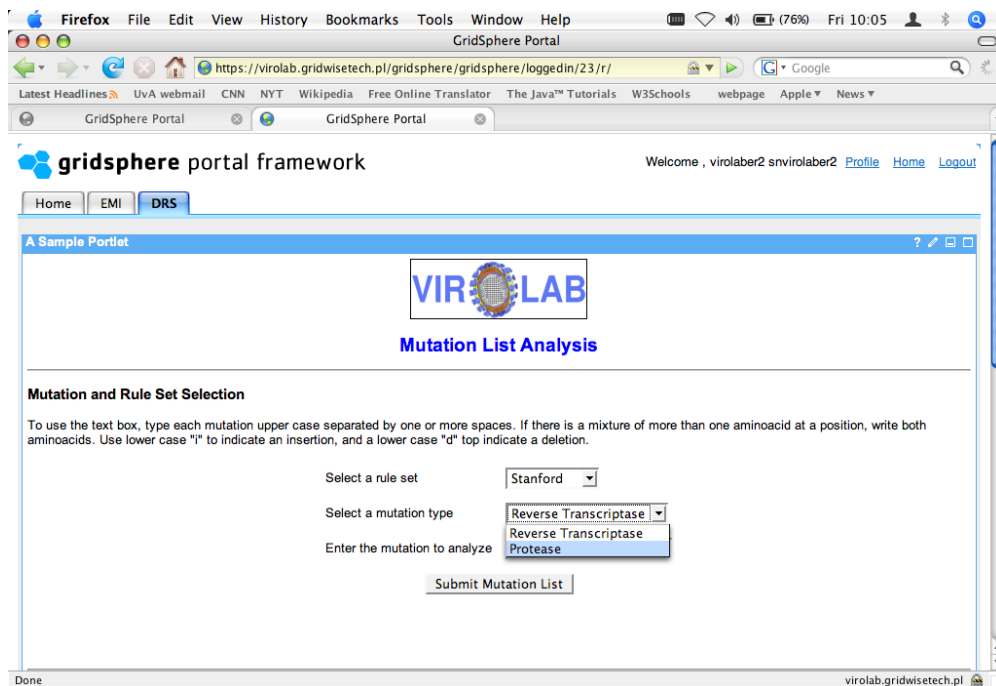


Figure -27: Specifying a mutation-type within the MLA portlet.

Entering and submitting a mutation and retrieving the results

The user can now enter a mutation list to analyse, upper case and separated by one or more spaces, and submits the mutation list for ranking (Figure -28). The user can now scroll down to retrieve the new ranking results, which may take a few seconds to come up (Figure -29).

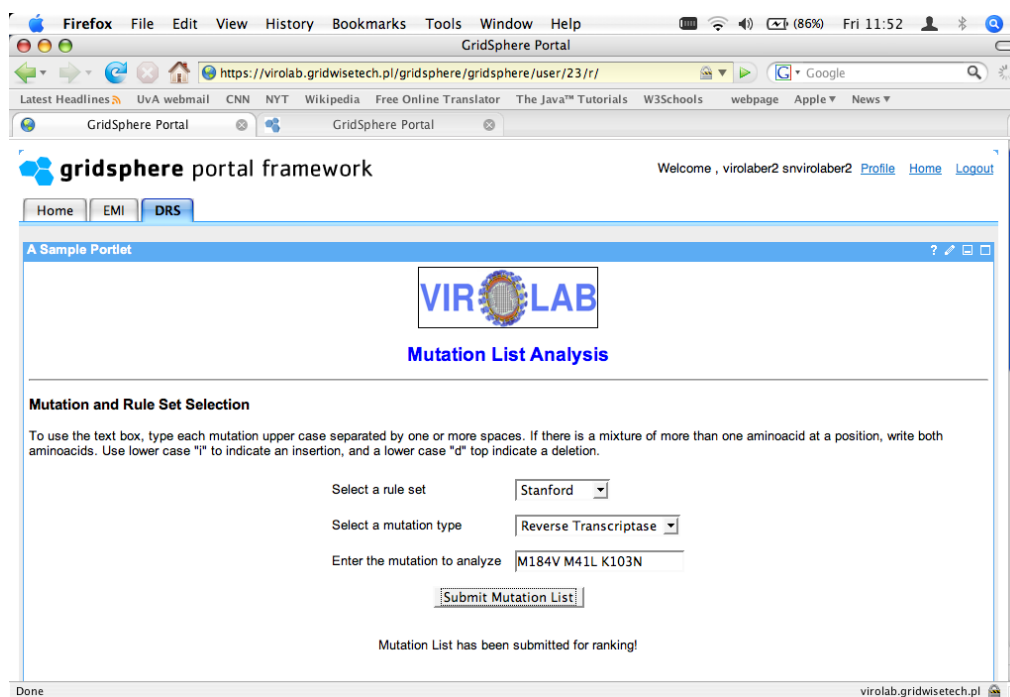


Figure -28: Submitting a mutation list to the DRS.

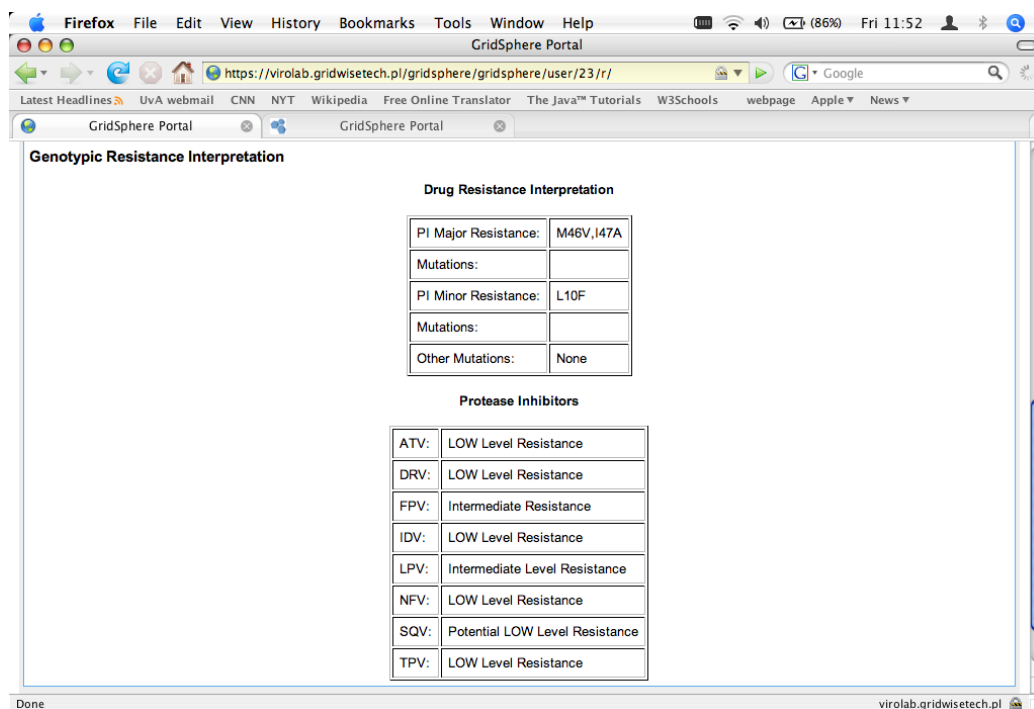


Figure -29: Retrieving the new ranking results from the DRS.

This version of the MLA portlet demo supports access to various rulesets via the DRS back-end system. As future work, the portlet will be fully integrated with the standard ViroLab experiment planning and execution systems, and additional functionalities will be added (e.g., simultaneous use of multiple rulesets for a single mutation list).

3.4.Demonstration of Planning of an Experiment

3.4.1.Description

This demonstration follows the procedure of experiment planning. For the sake of clearness we will use the same "From Genotype to Drug Resistance" experiment that is described in Section 3.1 - please check it to learn about the experiment concept. The experiment presented here is very similar, just slightly reduced in order to keep the demonstration brief and clear.

3.4.2.User Group

This demonstration is intended for the persons that would like to develop new experiments. The steps shown below are carried out using the main experiment developers' tool – the Experiment Planning Environment.

3.4.3.Setting up the work environment

In the course of this demonstration we will use the main tools of the experiment developer: Experiment Planning Environment (EPE) along with the set of ViroLab-dedicated plug-ins. Below are some links that you might want to check before further pursuing the demonstration description:

- EPE installation manual,
- EPE using manual,
- EPE plug-ins installation manual,
- Resources Browser using manual,
- Ontology Browser using manual.

All these resources are to be found in the Developer's Manual appendix to this document [D3.3DEV] and (in the updated, on-line version) on [VIROLAB-VL]. The demonstration approaches the planning scenario in a collaborative way and shows an example of two different people developing the same experiment. The same procedure also works in a single-developer version.

Starting EPE

Let us assume the developer starts work by downloading and installing the ViroLab EPE. In order to start it, he clicks the "virolabEPE" executable, which is placed in the main ViroLabEPE directory. After choosing the workspace location and waiting few moments, the ViroLab EPE *Welcome screen* appears (Figure -30), from where one is able to start planning new experiments.

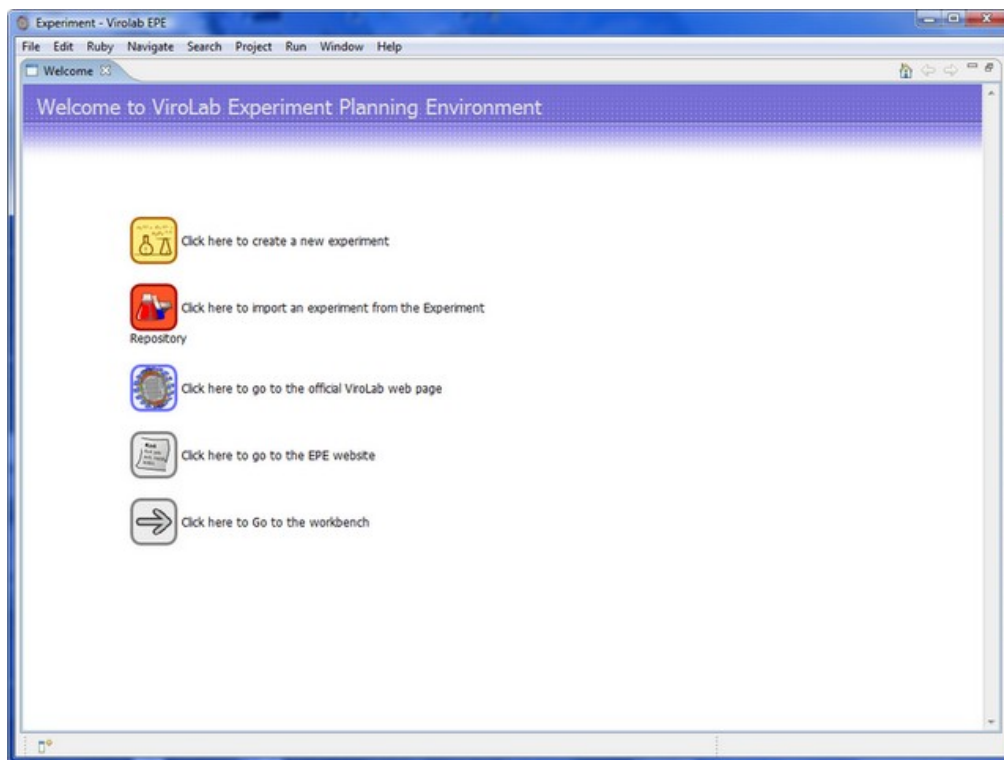


Figure -30: ViroLab EPE Welcome screen.

Creating a new project

In order to create a new experiment, the developer selects the "*Click here to create a new experiment*" icon from the *Welcome screen* (one may also do this from the workbench with the *File* menu). The *New Experiment* wizard should

appear (Figure -31). There are some information (e.g. experiment name and author contact info) to be provided to complete the wizard.

the most important thing

Experiment name: FromGeno2DrugRanking

Author e-mail address:

Organization:

Short description:

Long description:

☒ Use default location

Location: C:/Users/darek/Desktop/ViroLabEPE-0.2.4.1/workspace/FromGeno. Browse...

License

☒ Use ViroLab experiment license

☐ Use blank license file

☐ Provide URL to the license file:

Finish Cancel

Figure -31: The New experiment wizard.

After finishing the wizard, a new experiment project, with a proper structure, should be created in the workspace (or any location chosen when completing the wizard). The developer is now ready to start creating the *FromGenotype2DrugResistance* experiment plan.

3.4.4.Planning the data acquisition part

In this part of experiment plan development we will use the API of the Data Access Client (see [D3.3VLDEV] for details). The planning is started with removal of the generated *puts "Hello from Experiment!"* - we start with just the *require* lines to include all necessary runtime libraries for the data access module:

```
require 'cyfronet/gridspace/dac/DACConnectClass.rb'
```

The main source of medical data that is available inside the ViroLab Virtual Laboratory is the Data Access Service and we will use it through the DAC *connector* class. In its current version, DAC requires from the developer to provide the exact data service endpoint location in order to be able to contact it:

```
das = DACConnector.new("das",
  "angelina.hlrs.de:8080/wsrf/services/DataResourceService"
,
  "", "", "")
```

Apart from the type of data source we'd like to access (`das`) and the address URL, the rest of parameters are left empty (the access to this particular resource in the virtual laboratory is unrestricted). Having obtained the handle (represented by the `das` local variable) the developer is able to form a data query:

```
nt_seqs = das.executeQuery(
  "select nt_sequ from nt_sequences where
  patientID=#{patientID.to_s};" )
```

As one may clearly see it is a fairly standard SQL query of the *select* type. From the table that stores nucleotide sequences detected in viral isolates the query retrieves the ones that were detected in the body of a particular patient (since the data is anonymized one rather uses the artificial patient's ID instead of his/her name). As the query was parametrized with the patient's ID, the developer specifies its value through a local variable `patientID`. Later on, one may add the user request call to allow the experiment user type in the interesting patient's ID during the execution process - in order to learn how to do that, please consult an example of this kind of call provided later.

The entire data acquisition part of our script looks like this:

```
require 'cyfronet/gridspace/dac/DACConnectClass.rb'
patientID = 1
das = DACConnector.new("das",
  "angelina.hlrs.de:8080/wsrf/services/DataResourceService"
,
  "", "", "")
nt_seqs = das.executeQuery(
  "select nt_sequ from nt_sequences where
  patientID=#{patientID.to_s};" )
```

3.4.5. Sharing experiment with other developers

In some situations (e.g. when two or more developers are working simultaneously on the same experiment) the experiment sharing capability is useful. This is a very common situation that occurs in real life. To solve this problem ViroLab has created the Experiment Repository, which allows several (potentially widely separated) developers to collaborate. The ViroLab EPE provides some wizards to make communication with the Experiment Repository as easy as possible.

In order to share an experiment, the developer selects the "Share experiment..." popup menu (mouse right click) option (Figure -32).

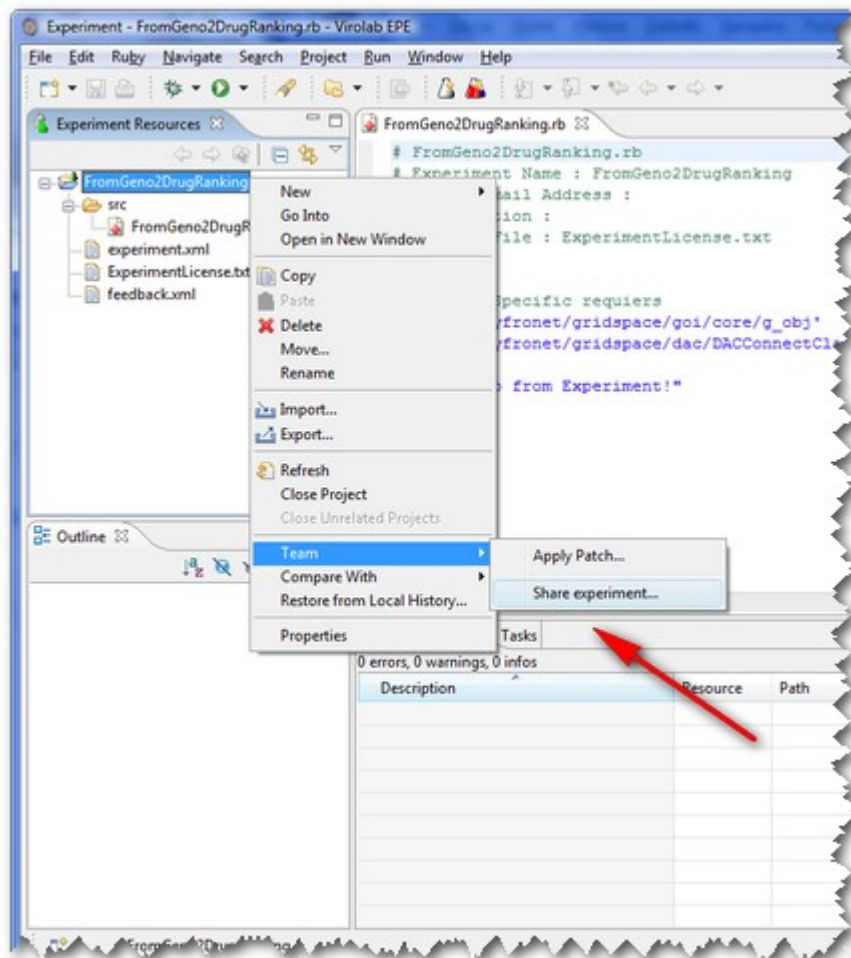


Figure -32: Share experiment menu option.

The wizard is a very simple one, requiring just a minimum effort to complete it.

1. In the first step one provides some basic information about the repository: location of the repository(the default location is already on the list), the access login and the password.
2. Secondly, there is an opportunity to choose the label, which will describe the experiment in the Experiment Repository.
3. Lastly, the developer may type the comment which will be attached to this experiment project revision.

After these steps the dialog appears which allows to decide which resource should be sent to the repository (Figure -33).

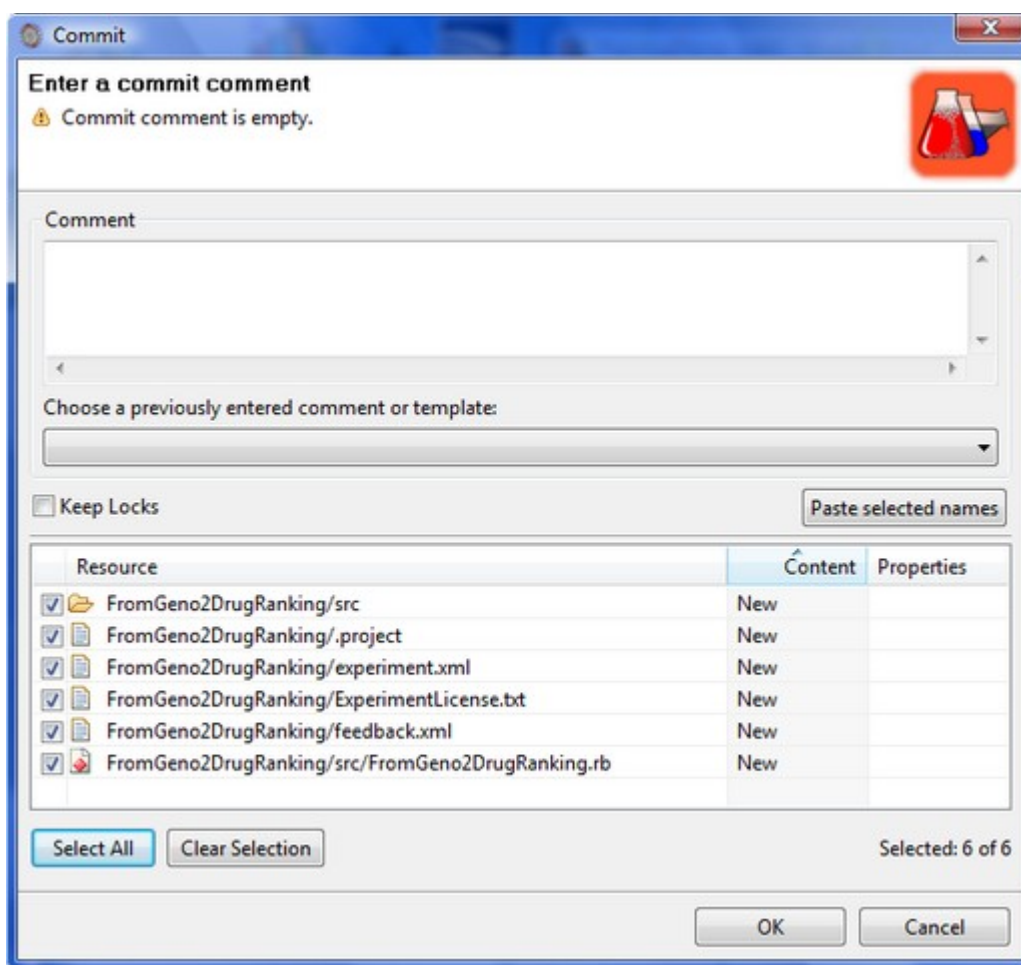


Figure -33: Select resources dialog.

If sharing operation completes successfully, the developer will be able to see that a small decorator has been attached to all resources that have been placed in the repository (this indicated that they are *shared* resources from now on).

3.4.6.Planning the computation access part

Importing a shared experiment.

If someone had shared an experiment using the Experiment Repository, another person can easily import it by using the *Import an experiment from Experiment Repository* wizard. To start this wizard one selects the proper icon from the EPE toolbar (Figure -34).

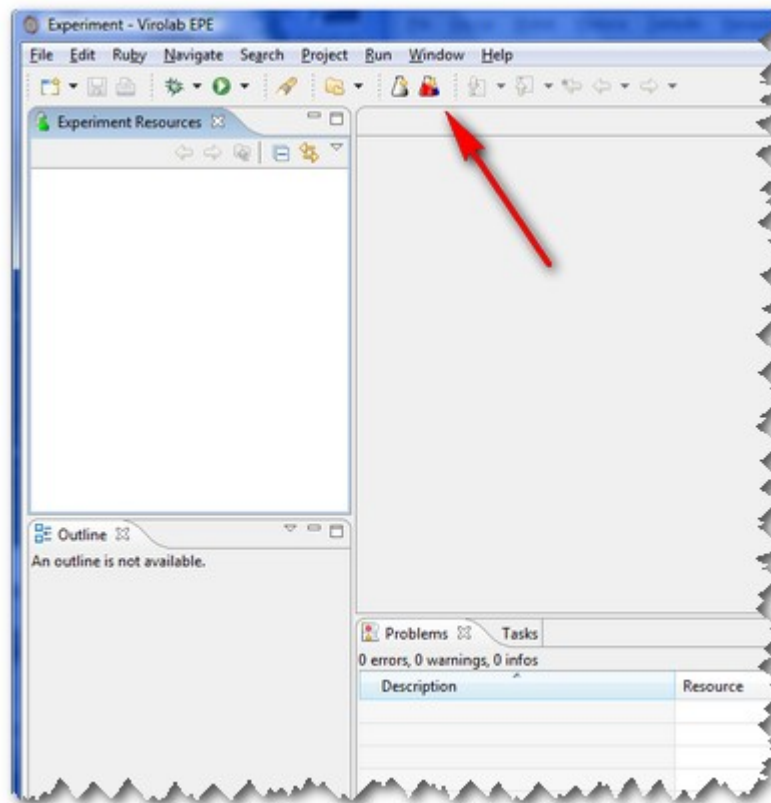


Figure -34: Starting the import experiment wizard.

The following points describe the steps of the experiment import procedure:

1. The first step is about providing information about the repository and it is similar to the one from the "Sharing experiment" wizard.
2. If everything went correctly one should be able to see a tree structure from which the tool downloads the experiment, that the developer is interested in.
3. The last two steps are optional and they regard: providing the experiment name, which will be used in the local workspace and choosing the location (and the working set) where the experiment will be placed.

After clicking the *Finish* button the selected experiment will be placed in the chosen location. After either sharing or importing an experiment, the developer is able to perform dedicated operations like: *committing* changes and *releasing* new versions of the experiment.

Planning the remote processing part.

In order to find the mutations of the virus that are related to drug resistance, their nucleotide sequences need to be aligned. There is a tool that is able to perform that task (published by the Rega Institute from Leuven). In order to acquire a handle to the tool the developer uses the computation access module from the runtime library (switched *on* with another `require`):

```
require 'cyfronet/gridspace/goi/core/g_obj'
alignment = GObj.create('regadb.RegAlignement')
```

As our virus genotype data source returns the sequences with no FASTA header and the alignment tool requires one, we do this simple procedure to add a dummy header:

```
nt_seqs.each_index {|ind|
  nt_seqs[ind] = ">name\n"+nt_seqs.flatten[ind]
}
```

After that the sequences are prepared for the alignment process, that takes place in a designated region of the sequence (that is, for a particular protein) - denoted by the not-yet initialized `region` variable. The result is a record of information from which we get just the mutations list:

```
result = alignment.align(nt_seqs[0], region.upcase)
mutations = result.split(',').last.chop
```

One final step ahead: the mutations are sent to the Drug Ranging System that, based on these mutations and the decision rule set we choose, shows the set of advice regarding the future treatment of the patient. For this example we use the Retrogram set of drug resistance rules (the region indication is also important here).

```
drs = GObj.create('org.virolab.DrugRankingSystem')
resistance = drs.drs('retrogram', region, 100, mutations)
puts resistance
```

The experiment plan is almost finished at this point. One final feature will be to parametrize the experiment so its user may perform it for chosen region (protein). For this we require another module from the runtime library, called *Data Requester*:

```
require 'DataRequester'
```

This module allows us in a single, simple procedure call relay a user input request to the Portal side of the experiment execution. In a few words - it allows as to ask the user to put in some data:

```
region = DataRequester.new.getData(
  "Region (lowercase: \"rt\" or \"pro\")")
```

We are ready to release a version of our experiment plan now.

3.4.7. Releasing the experiment plan for users

When the experiment developers decide that the experiment is stable enough, they can use EPE's built-in functionality to prepare and publish a new experiment release. This makes the experiment visible through the ViroLab Portal for the experiment users (e.g. scientists and clinicians).

In order to release a new version of the experiment, it should have already been placed in the Experiment Repository. To start the release wizard select the *Team* -> *New release...* option from the popup menu (right mouse button click on the experiment name) (Figure -35).

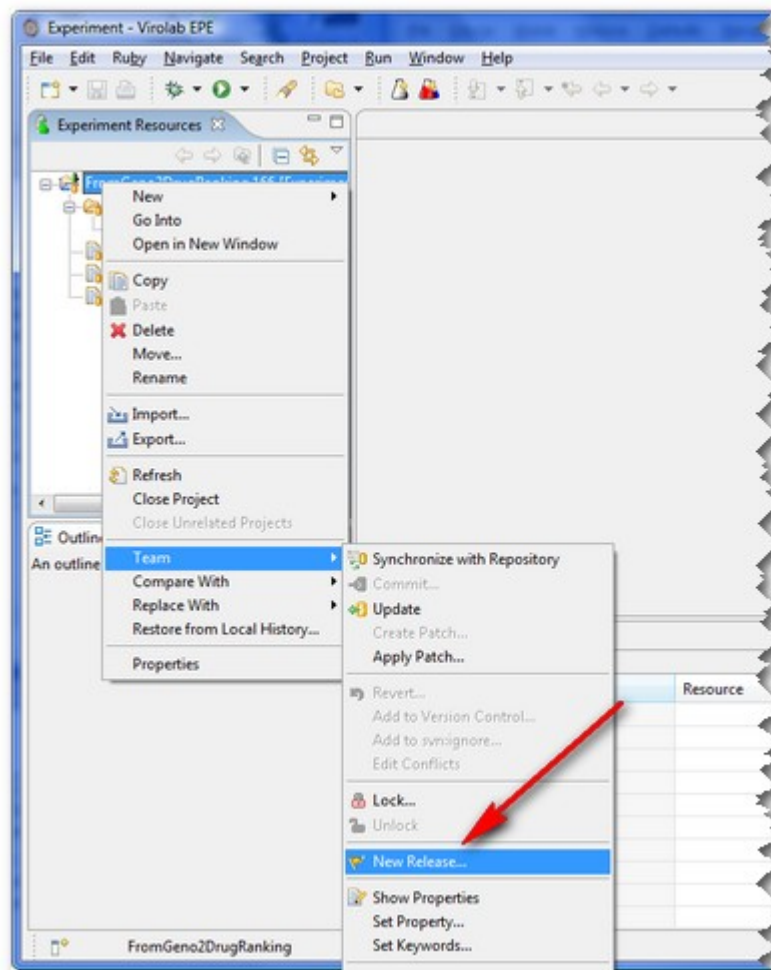


Figure -35: Starting the release experiment wizard.

The only information one is obliged to provide is the experiment release version (Figure -36). Optionally the developer can attach a release comment which may contain information about the release (e.g. changes added since the previous release).

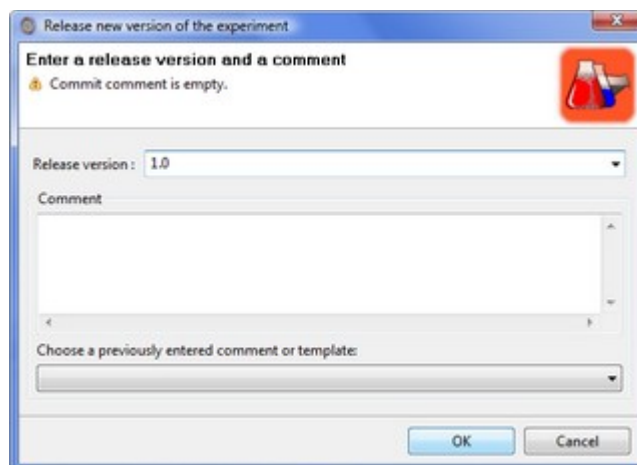


Figure -36: Choosing release version.

And that's all there is to it – the new release of the experiment has been created. Experiment users are now able to execute it using the EMI portlet in the ViroLab Portal. Please consult the experiment execution demonstration in Section 3.1 to see how this experiment plan works when executed.

3.5.Data Mining for a Classification Pattern

3.5.1.Description

This sample “toy” experiment demonstrates how we can perform data analysis using Weka [WEKA] data mining toolkit. The experiment has the following steps:

1. A sample dataset is retrieved from a database. In this experiment we use the 'contact_lenses' dataset, which is one of sample datasets from Weka distribution. It contains the patient data such as age, spectacle prescription, astigmatism and tear production rate, accompanied by recommended contact lenses type.
2. The contact lenses dataset is split between a training set and a testing set.
3. A classifier is trained and it produces rules, which can be used for classification. In this example we use the simplest "One Rule" classifier, which actually relies only on a rule based on one attribute.
4. A trained classifier is used to predict the data from the testing dataset.
5. The results of prediction can be compared to the actual data from the testing set. The prediction quality of the classifier can be determined in this way.

A schematic data flow and order of operations is shown in Figure -37.

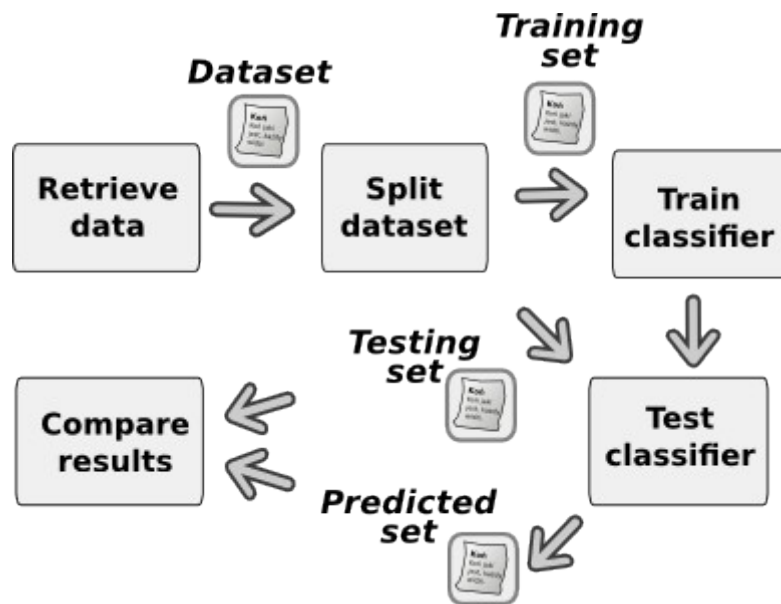


Figure -37: Data mining using Weka.

3.5.2.Intended user group

The data analysis experiment intends to show the usage of the Virtual Laboratory for scientists such as virologists. It demonstrates how an experiment plan has to be prepared by an experiment developer (using Ruby). It is also shown how the experiment can be executed locally using the command-line `gsengine` runtime. Such a scenario is mostly dedicated to advanced users of the virtual laboratory, who intend to create, modify, and "experiment" with various experiments in their scientific work. The demonstration also shows the possible usage of different middleware technologies, such as Web Services and MOCCA components.

3.5.3.Technical Perspective

This test experiment presents ability to invoke in one script both Web Services and MOCCA components. The main task of this script is to predict the contact lenses type using patient data stored in a database and check the quality of predicted information.

Realization

1. The "WekaGem" Web Service (stateless) and "OneRuleClassifier" (stateful) are available on a remote server
2. "WekaGem" has 3 methods:
 - loadDataFromDatabase: imports data from database and convert into ARFF [WEKA] format
 - splitData: splits data into training and testing data
 - compare: compares testing data with predicted data.
3. "OneRuleClassifier" has 2 methods:
 - train: trains this stateful component
 - classify: tries to predict result using information gained during training process.
4. Both "WekaGem" and "OneRuleClassifier" are registered inside the Grid Resources Registry (GRR)
5. Grid Operations Invoker is able to instantiate these gems (as Grid Object Instance) and invoke their operations

3.5.4.Requirements

Running this example experiment requires the following items to be available:

1. gsengine binary distribution installed on user's machine
2. VLSampleExperiments package downloaded and extracted to a local disk
3. ViroLab infrastructure set up and running (GRR, H2O kernel for MOCCA components, Weka deployed as gems, database with sample data). It is assumed that all these items are made available for experiment users and developers by virtual laboratory administrators

3.5.5.Detailed code explanation

The entire code is written in the Ruby programming language and is executed with the JRuby interpreter.

Highlights:

```
require 'cyfronet/gridspace/goi/core/g_obj'
```

This includes the main part of the Grid Operation Invoker (GOI) to be used later on.

```
logger = JLogger.getLogger('goi.wekaexperiment')
logger.info('Start of weka experiment !!')
```

JLogger is an specific GridSpace class that is responsible for logging specific application events/messages. Inspiration for this class is log4j framework (for more information see [LOG4J])

```
QUERY = 'select * from contact_lenses limit 100;'
DATABASE = "jdbc:mysql://127.0.0.1/test"
USER = 'testuser'
PASSWORD = ''

retriever = GObj.create('cyfronet.gridspace.gem.weka.WekaGem')
A = retriever.loadDataFromDatabase(DATABASE, QUERY, USER, PASSWORD)
puts 'Data retrieved from DB :' + A
```

In this part of the experiment "WekaGem" is created and data from a database is loaded to variable A. At the end information about loaded data is logged.

```
B = retriever.splitData(A, 50)
trainA = B.trainingData
testA = B.testingData
```

After the data is loaded it has then to be split into training and testing data. To do that the second "WekaGem" method is used. 50% of data will be used as testing data other 50% as training data. The result is assigned to trainA and testA variables.


```
Classifier = GObj.create('cyfronet.gridspace.gem.weka.OneRuleClassifier')
```

We have loaded and split data. Next task of this experiment is to create classifier that is able to predict the contact lenses type. OneRuleClassifier is instantiated. It is a stateful MOCCA component.

```
attributeName = 'contact_lenses'  
classifier.train(trainA, attributeName)
```

This component has to be trained before it is able to predict any information.

```
prediction = classifier.classify(testA)  
puts 'Predicted data:' + prediction
```

We can now predict the necessary information. Of course, important information can be logged.

```
classificationPercentage = retriever.compare(testA, prediction, attributeName)  
  
puts 'Prediction quality:' + classificationPercentage  
puts 'End of weka experiment !!'
```

The last important item is to check how good this prediction was. WekaGem Web Services are used here. The result of the prediction is printed.

3.5.6. Running the experiment

In this example we run the experiment from the command-line. This requires that a user has GSEngine installed, as described in GSEngineUserManual. Weka experiment is the one included in the VLSampleExperiments package.

In order to invoke the experiment, type:

```
virolab:~/sample-experiments$ gsengine weka_experiment_lenses.rb
```

First, we get some initial messages:

```
No evaluation request specified - using default.evaluation.request.xml...  
544 [INFO] goi.wekaexperiment - Start of weka experiment !!
```

Subsequently, the initial dataset is printed out:

```
Data retrieved from DB:@relation contact_lenses  
  
@attribute age {young,pre-presbyopic,presbyopic}  
@attribute spectacle_prescrip {myope,hypermetrope}
```

```
@attribute astigmatism {no,yes}
@attribute tear_prod_rate {reduced,normal}
@attribute contact_lenses {none,soft,hard}

@data
young,myope,no,reduced,none
young,myope,no,normal,soft
young,myope,yes,reduced,none
young,myope,yes,normal,hard
...
```

Later we can see the training ...

```
Data for training: @relation contact_lenses
...
@data
young,myope,no,reduced,none
young,myope,no,normal,soft
young,myope,yes,reduced,none
...
```

... and testing dataset:

```
Testing data: @relation contact_lenses
...
@data
pre-presbyopic,hypermetrope,no,reduced,none
pre-presbyopic,hypermetrope,no,normal,soft
pre-presbyopic,hypermetrope,yes,reduced,none
pre-presbyopic,hypermetrope,yes,normal,none
...
```

Creating a MOCCA component prints out some debug information:

```
Connecting          to          kernel:          http.tunnel://virolab.cyf-
kr.edu.pl:7781:7799/
```

After the classifier is trained, a prediction result on a testing dataset is printed out:

```
Predicted data:@relation contact_lenses
...
@data
pre-presbyopic,hypermetrope,no,reduced,none
pre-presbyopic,hypermetrope,no,normal,soft
pre-presbyopic,hypermetrope,yes,reduced,none
```

```
pre-presbyopic,hypermetrope,yes,normal,soft
...
```

It can be seen that the prediction quality of One Rule classifier is far from perfect, as it is the simplest possible classifier. This is confirmed by the result of the compare method:

```
Prediction quality: 0.6666667
```

Finally, some closing messages are produced by the experiment. It can be seen that the experiment does not return any value.

```
End of weka experiment !!
[EvaluationCallback] result returned:
```

This completes the data analysis experiment.

This experiment demonstrates, that a scientist can easily modify the experiment plan by editing the experiment script and run it from command-line. It is also possible to use Interactive Ruby (IRB) to conduct experiments interactively.

4. Runtime System

The main purpose of this module is to provide all the functionality of the virtual laboratory runtime system in the moment of experiment script execution. The developer expects the runtime layer of the laboratory to include all the APIs and implementations required for activities such as accessing the registry to gather important data on available services, obtaining appropriate experiment input data, or running a remote computation. The listed functionality is inside a library and it is loaded into the interpreter whenever an execution of a ViroLab experiment plan is demanded. For more detailed description of the runtime system purposes and designed functionality please check Section 6.1 in [D3.2].

The following sections list the current state of development of various parts of the runtime. However, to see how they are integrated together, let us consider the experiment execution mechanism.

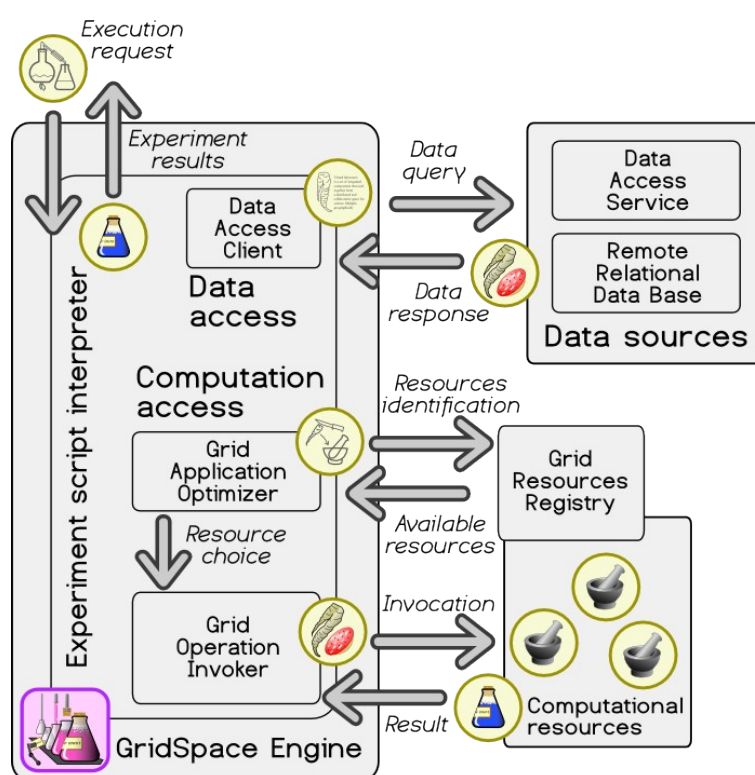


Figure -38: Experiment execution mechanism within the virtual laboratory runtime.

Figure -38 decomposes the entire mechanism of experiment execution into some notable parts. First of all, the virtual laboratory runtime component (GSEngine) needs the experiment plan - depending on the execution mode it is either provided directly or is downloaded by the GridSpace Engine (a part of the runtime) from the Experiment Repository. Now, the most important part of the plan is the experiment script that models the procedure flow - this script is now being interpreted by a built-in experiment script interpreter (that is a JRuby [JRUBY] interpreter).

To better describe the notion of an experiment script, let us use an example from the clinical virology domain. There are the three basic blocks of the experiment: acquiring input data, processing the data, and obtaining the result. In this

particular case the script provides virus-to-drug resistance information to the user.

```
patientID = 6

rdb = DACConnector.new("mysql","virolab.cyfronet.pl","test","testuser","")
mutations = rdb.executeQuery
  ("select mutations from nt_sequence where patient_ii=#{patientID.to_s};")

drs = GObj.create('org.virolab.DrugRankingSystem')
resistance = drs.rank("protease", mutations)

puts resistance
```

The three main steps are as follows:

- Get the mutations of viruses for a certain patient (lines 2-3)
- Ask for a drug resistance ranking based on the retrieved mutations (lines 4-5)
- Present the result to the user (line 6)

Mutations data is retrieved from a remote relational MySQL database with a specified query. Here, the developer uses the Generic Data Access Client (DAC) (see Section 4.5 for more in-depth explanations) module to acquire a proper database handle (**rdb**). With this handle, the query can be issued and the result is stored in a local **mutations** variable. Later on, the developer uses the Grid Operation Invoker (GOI) (see Section 4.4) API to create a local stub of a remote service called *DrugRankingSystem* - this stub is represented as a local variable **drs**. This is in turn used to remotely invoke the **ranking** operation, which returns proper information related to virus-to-drug resistance. Finally, this information is presented to the user with the simple terminal printout operation **puts**.

During the script interpretation phase, there are two specifically interesting stages: remote data access and remote computation access. These two are very important virtual laboratory's operations and supported by the Generic Data Access Client (the data access part) and the Grid Operations Invoker, Grid Resources Registry, and Grid Application Optimizer (the computation access part). All these modules are presented in Figure -38 and their current implementation state is described in detail below.

4.1.GridSpace Engine

GridSpace Engine (in short *GSEngine*) is the core part of the Virtual Laboratory where experiments are submitted, evaluated, and where the status of evaluation is monitored and stored, and which finally provides an experiment executor with intermediary experiment results as well as the ultimate ones. GridSpace Engine provides the runtime environment within which a Ruby-based GScript [D3.2] [RUBY] is evaluated and some parts of Virtual Laboratory such as Grid Operation Invoker (GOI) (Section 4.4), Generic Data Access Client (DAC) (Section 4.5) operate. It is accessed by the Portal's Experiment Management Interface (EMI) and the Eclipse-based IDE of Experiment Planning Environment (EPE).

In general, *GSEngine* is a service that accepts evaluation requests, which carry all information needed in order to evaluate a GScript application:

- Configuration information related to GridSpace environment e.g. URLs to GridSpace services such as Grid Resources Registry (Section 4.2) or Grid Application Optimizer (Section 4.6) as well as policies to apply, credentials etc.
- Either application code or all indispensable information in order to locate and get the application code.

GSEngine clients can asynchronously submit such evaluation requests and trace the evaluation in the notification mode thanks to an evaluation callback mechanism. Since a *GSEngine* client launches the evaluation asynchronously and in non-blocking manner it has to provide an evaluation callback object, that handles notifications related to the submitted evaluation.

Not only evaluation notifications are sent back to the client, since *GSEngine* supports interactive mode of data input. Each time the application waits for input from the application executor, the data input callback is sent back to the client, which returns with the data input request satisfied. That is handled by the evaluation callback object as well. *GSEngine* provides a dedicated library for interactive user data input requests, which is available for application developers.

There are several available evaluation request types which vary in the way how they provide an application code. Each type has a corresponding scenario for providing application code:

- Explicit Script Evaluation Request – the code of the script or scripts to evaluate is carried along with evaluation request (see: Figure -39)
- Local File Script Evaluation Request – the script or scripts to evaluate are stored in the local file system, and the evaluation request specifies paths to them (see: Figure -40)
- Repository Staged Script Evaluation Request – the script or scripts to evaluate are stored in the Application Repository, and the evaluation request specifies the repository URL, repository credentials, and the scripts location in the repository (see: Figure -41)

In order to support the Repository Staged Script Evaluation Request type, the *GSEngine* is enabled to incorporate a Source Code Management (SCM) system client. Thanks to its open architecture any SCM system may be supported that implements *GSEngine* App Repo API, which will be discussed in details further. The primary implementation of the *GSEngine* Application Repository API, which comes along with the *GSEngine* distribution is the implementation based on Subversion (SVN) [SVN] SCM system.

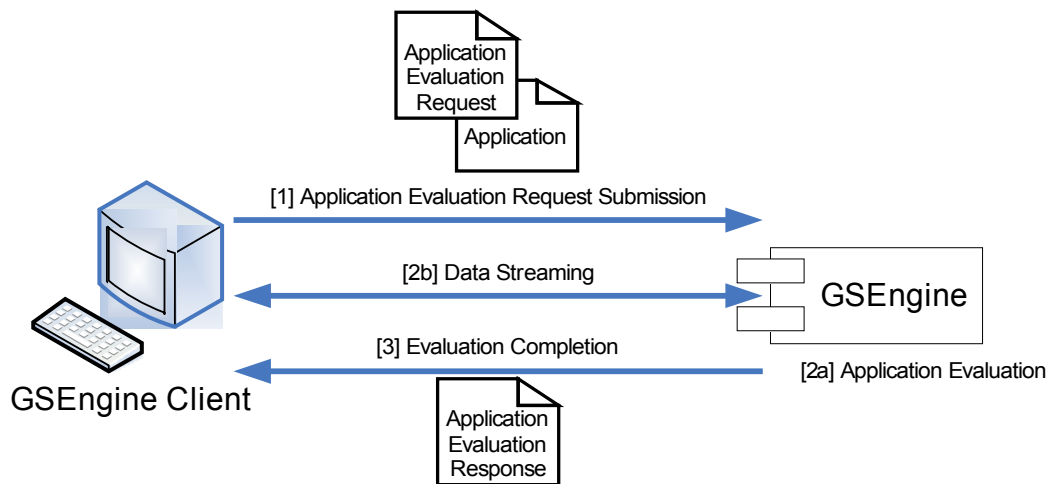


Figure -39. Explicit Script Evaluation Request and application code provisioning scenario it induces.

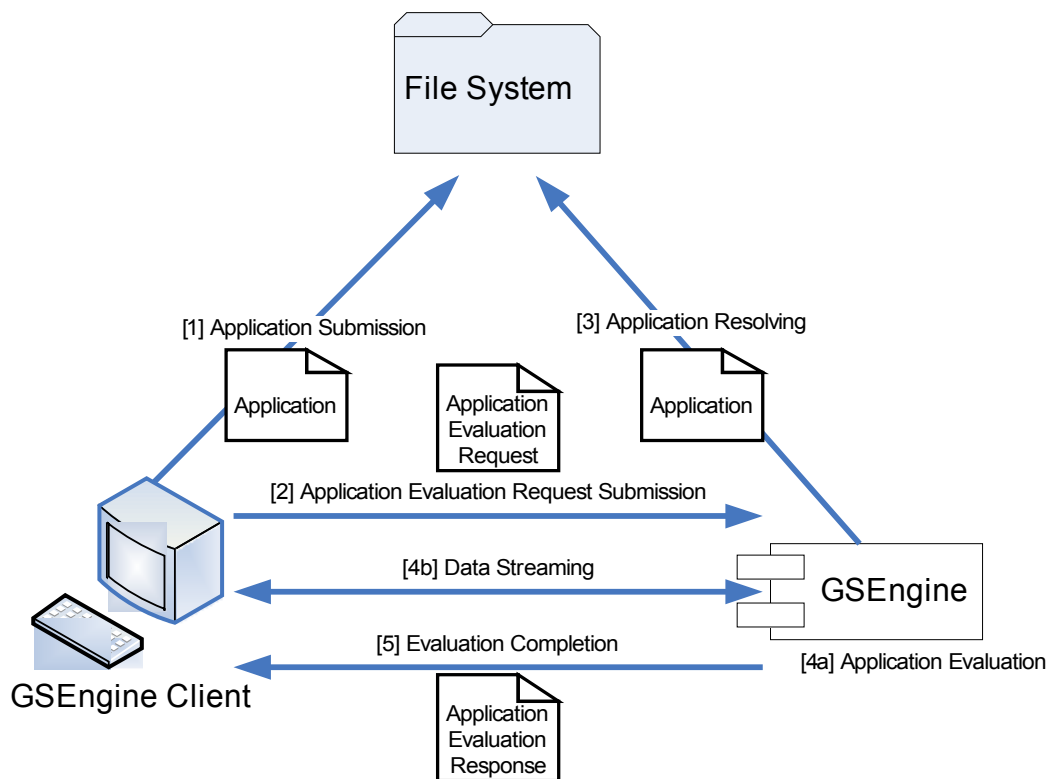


Figure -40. Local File Script Evaluation Request type and the application code provisioning scenario it induces.

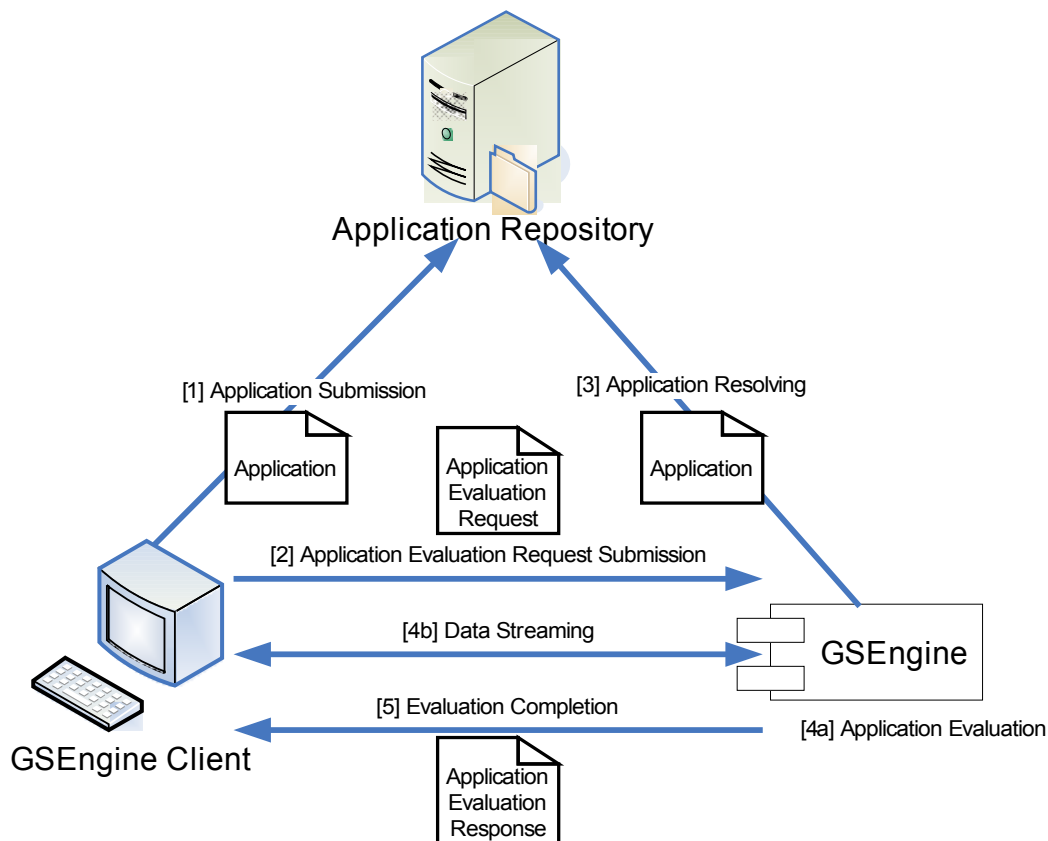


Figure -41: Repository Staged Script Evaluation Request type and the application code provisioning scenario it induces.

GSEngine may be used in two ways:

- by a dedicated command-line tool – *gsengine*
- via Java APIs

Considering *GSEngine* from the perspective of the other Virtual Laboratory modules incorporated in the Runtime Environment such as GOI and DAC, the *GSEngine* provides an application evaluation context containing configuration information as well as application-specific information. Moreover, *GSEngine* takes a significant part in the experiment monitoring performed by the monitoring system since it traces and collects monitoring data relevant for experiment execution. For that reason, all Virtual Laboratory modules consider *GSEngine* as the monitoring event logging system and feed it with monitoring events.

4.1.1.Implementation Description

GridSpace Engine implementation involves several tightly-coupled components.

The internal components (those developed in the scope of GridSpace Engine effort) are **GSEngine Core** that implements **GSEngine API** and uses **GSEngine SVN App Repo** that, in turn, implements **GSEngine App Repo API**. *GSEngine Core* provides GridSpace Engine core functionality, while *GSEngine SVN App Repo* is an extension to the core with the support of the Application Repository. Subversion (SVN) [SVN] as the source code management (SCM) system fits well into the idea of the Application Repository since it enables versioning and collaboration by default. The interfaces and implementing components are

decoupled in order to enable the usage of other implementations, e.g. those of the Application Repository.

The aforementioned components use external libraries such as JRuby 1.0 and SVNKit 1.1.2. Having in mind that GScript is actually a Ruby language with external libraries provided, JRuby [JRUBY] is used as a Ruby [RUBY] language implementation written in Java that allows integration and cooperation between Java code and Ruby code. SVNKit [SVNKIT] is used as a client of the SVN-based implementation of the Application Repository. The dependencies between all these internal as well as external components of GridSpace Engine are depicted in Figure -42.

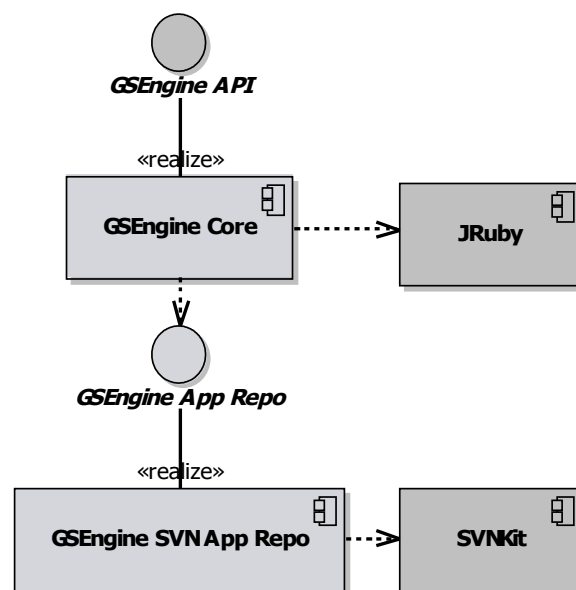


Figure -42: Main components and interfaces of GSEngine.

GridSpace Engine incorporates the Virolab-specific libraries of GOI and DAC by including them in the JRuby interpreter classpath, assuming that they are placed in the directory pointed by a `GS_HOME` environment variable. Moreover, Virolab-specific libraries are provided with a GridSpace Engine Application Context that contains all the parameters required by the libraries and that is passed by GridSpace Engine as Ruby constants to the JRuby interpreter runtime.

For the monitoring purpose a dedicated Ruby library for event logging is designed that will be used by GOI and DAC. Moreover, the GScript developers will be equipped with a library for handling data inputs.

The GridSpace Engine shall provide its clients realizations of `InterpreterFacade` interface for embedded evaluation (carried out in the same Java Virtual Machine) as well as for evaluation performed in the remote service. At the current stage only the `EmbeddedInterpreter` realization is available.

However, for the present, *GSEngine* supports above discussed set of evaluation requests types, this set is open to expand in the future by providing an appropriate evaluation request subclass and the corresponding evaluation request support subclass following the design pattern shown in Figure -43.

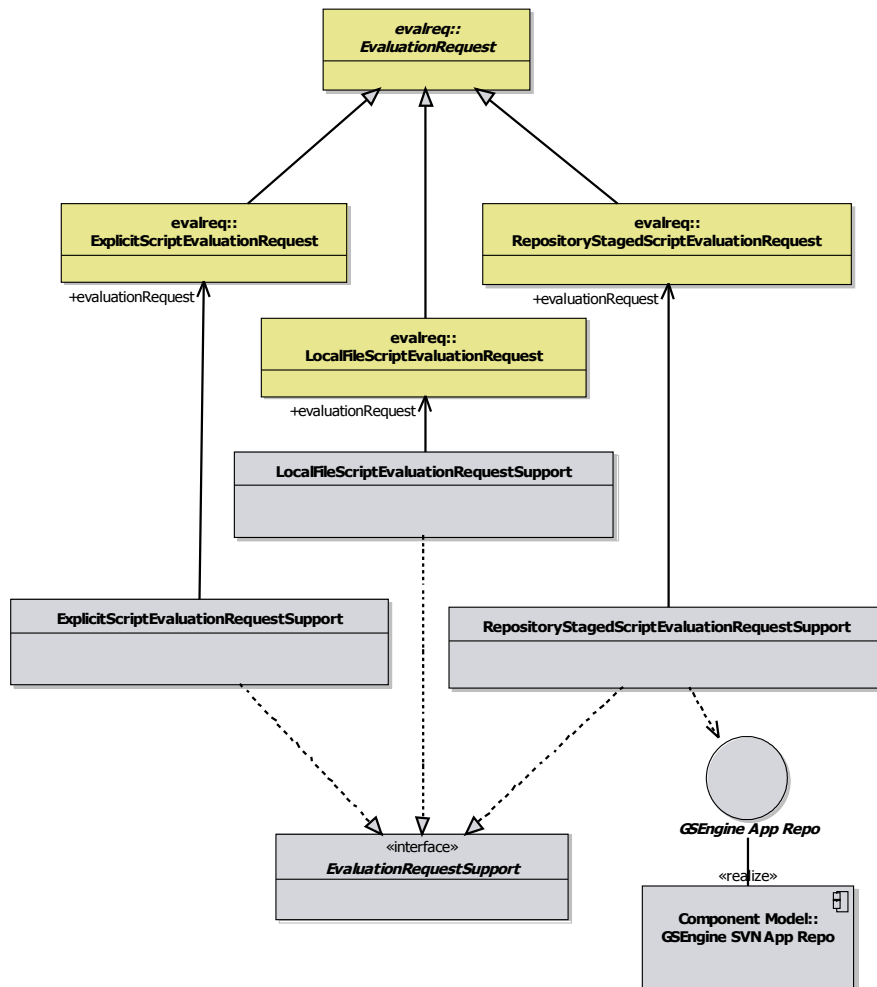


Figure -43: Class diagram of supported types of evaluation request.

4.1.2.Current Functionality

At the current stage of development, the GridSpace Engine contains the following features:

Available realizations of Interpreter Façade:

- Embedded Interpreter that is embedded in the caller's Java Virtual Machine

Supported evaluation request types:

- Explicit Script Evaluation Request
- Local File Script Evaluation Request
- Repository Staged Script Evaluation Request

Other features:

- Command-line toolkit for the Embedded Interpreter:
 - `gsengine` tool for submission of evaluation requests
 - `gsquery` tool for performing queries via Generic Data Access Client
- Support for SVN-based Application Repository
- Foundation for a data input library

- Foundation for a monitoring event logging library

4.1.3.Planned Functionality

New additions are planned for the GridSpace Engine module:

- GridSpace Engine Server: a remotely accessible service fulfilling GridSpace Engine functionality that is accessed via a dedicated client realizing Interpreter Façade interface
- Experiment Planning Environment (EPE) plugin for submission of evaluation requests and experiment run monitoring
- Robust data input library – a library based on a declarative XML-based language to specify the user interface of data input forms featured with a wide range of widgets. Supporting data input library by EMI, EPE and *GSEngine* command line tool.
- Full implementation of the monitoring event logging library – adjustment of the monitoring event logging library with emerging monitoring and provenance systems.

4.2.Grid Resources Registry

A problem of the existing grid solutions is the complexity of creating grid applications. Consequently, the developer has to have knowledge about many complex technologies. To ease the process of creating complex distributed applications, a Grid Resources Registry (GRR) is created. The main purpose of this component is to present all available resources that can be used during the creation and execution of an experiment scenario. What is more important, by providing three layers of resources description [D3.2] scenario development is much simpler and a ViroLab user may not worry about the technology of the invoked service.

4.2.1.Implementation Description

The current architecture of the Grid Resources Registry is presented in Figure -44 . Components with light background (GRR Admin Plug-in, Resources Browser Plug-in, Registry Service, Registry core and its subcomponents) are internal GRR elements, the darker ones (Grid Operation Invoker, GrAppO Optimizer, Domain Ontology Store and Monitoring System) are a part of GSEngine and monitoring system.

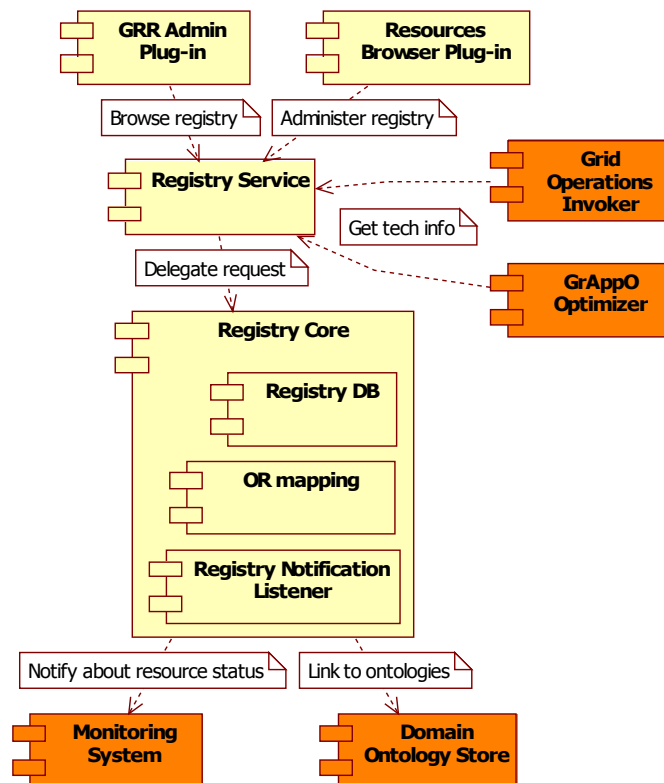


Figure -44: Grid Resources Registry decomposition

The central element of the GRR is **Registry Core** that is responsible for communication with the GRR user interface, GSEngine and monitoring systems. It is implemented as a Java [JAVA] application that is deployed into the `tomcat` [TOMCAT] container. The application was designed and is developed using the MVC pattern, it uses `mysql` to store information about resources and the `hibernate` framework [HIBERNATE] is used to provide object/relational persistence (model). Using this mapping, GRR logic was implemented (controller) and `Xfire` Web Service framework is used for communication with a GRR user interface (Eclipse plug-in) and other external components (view). All these layers are connected using `Spring` framework that uses the IoC (Inversion of Control) pattern [IOC].

During the development stage, the user is able to browse resources available in the ViroLab environment using the GRR user interface (**Resources Browser**). Resources Browser uses Eclipse RCP [RCP] mechanisms to create views that can be easily integrated into the EPE environment (see Section 3.2 of [D2.3] for details). The component connects to Registry Core using a Web Service (`Xfire` framework [XFIRE]) and is able to present three layers of resources description (Grid Objects and their operations, Grid Object Implementation and Grid Object Instances). The plug-in is integrated with the EPE script editor and allows to easily insert a code line that creates a selected Grid Object. To find a proper Grid Object the user can browse resources within the Resources Browser or the user can use the highest level of resources abstraction – ontologies. To provide this functionality, the Resources Browser is integrated with the **Ontology Browser**. It is a two way integration, the user is able to search Grid Objects and their

operations that fulfill ontology queries and, moreover, is able to find the ontology meaning of a concrete operation.

For browsing resources outside the EPE, a web version of GRR is implemented. It has limited plug-in functionality and allows only to browse information about resources.

The second type of communication with Resources Core is when an experiment script is executed by GSEngine. In case of invoking remote resources GrAppO optimizer asks the registry about all implementations and instances of a concrete grid object. After that Grid Operation Invoker receives a resource technical description that allows to invoke different technologies. Both GrAppO and GOI use the Web Services technology to communicate with GRR.

Integration with the monitoring system is planned to be implemented at the next stage of the project.

4.2.2.Current Functionality

At the current stage of development, Grid Resources Registry provides the following capabilities:

- Grid Resources Registry core:
 - Enable to register and browse information about resources
 - Support for Web Services, MOCCA components, EGEE (Job submission) and WTC technologies
 - Integration with Grid Operations Invoker and GrAppO optimizer
- Grid Resources Registry EPE plug-in:
 - Capability to search and present information about resources to users
 - Fully configurable using dedicated properties page and VO properties
 - Integration with EPE script editor and Ontology Browser
- Web Resources Browser:
 - Capable to browse all information about resources
 - Configurable using VO properties

4.2.3.Planned Functionality

The following enhancements are planned:

- Grid Resources Registry core:
 - Support for new technologies (WS-RF, DEISA)
 - New functionalities that allow to manage information about resources in a more comfortable way for the user
 - Integration with the GEMINI monitoring system
 - Additional information about resources that allows to browse and search resources in a more user friendly way
- Resources Browser plug-in:
 - GRR administration EPE plug-in (adding, removing, editing Grid Objects their operations, Grid Object Implementations and Instances)
 - Adding additional context items that allow to invoke administration wizards from Resources Browser plug-in.

- Further integration with EPE and Ontology Browser plug-in
- Web Resources Browser:
 - New mechanisms that allow to log in to the Web Resources Browser and customize GRRs that will be browsed

4.2.4.Deviations from the Design Document

The only change compared to the design deliverable is the technology that GRR is implemented in. Instead of using Ruby frameworks like `Ruby on Rails` [ROR] or `Nitro` [NITRO], `Java` [JAVA], `hibernate` [HIBERNATE], `Xfire` [XFIRE], and `Spring` [SPRING] were chosen to implement this component. The main reason for such a design change was that GRR communicates with many additional components and some communication protocols (e.g. `Ice`, `WSRF`) are not provided yet in Ruby.

4.3.Domain Ontology Store

The main purpose of the Domain Ontology Store (DOS) is to support a common view of the modeled application domain, which is the viral diseases, for virtual laboratory systems and users. This helps build a collaborative environment whose functions are naturally distributed, with many parties contributing from around the world, and which is meant to provide a non-trivial level of integration between various activities performed by different users of the laboratory. For this purpose the domain knowledge is modeled and represented with ontologies to allow human users to easily identify common items of interest. Therefore, the main responsibility of the Domain Ontology Store is to contain ontological models in non-volatile storage and to provide their contents online for both read and write access. The detailed design of DOS, together with use case analysis, decomposition diagrams and sequence charts, which model a dynamic behavior, is given in Section 6.1.3 of [D3.2].

4.3.1.Implementation Description

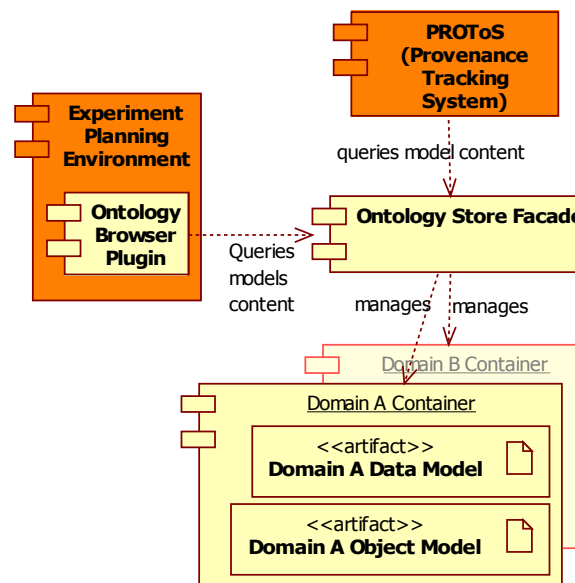


Figure -45: Domain Ontology Store decomposition.

The current composition of the Domain Ontology Store is presented in Figure -45 . The components and artifacts depicted with the light background are internal (that is, are provided by this module) while darker ones are external.

The central piece, which is the **Ontology Store Façade**, is currently implemented with a server that is available online. The server provides an HTTP endpoint where any suitable client may access the internals of the store. The technology used here is provided by Sesame [SESAME] (a repository to store and publish RDF Schema documents [RDFSCHEMA]) and is running inside an instance of Java Virtual Machine [JAVA].

The models are developed as XML documents adherent to the Web Ontology Language [OWL]. During the development, the Protégé [PROTÉGÉ] framework is used that effectively shortens the design time and makes the relatively complex process of devising ontology taxonomies much easier. All the models are strictly versioned, one may always ask for the version number of a taxonomy inside the store.

The container where the models are located and where the Façade is able to browse them, is based on the OpenRDF triples [RDF] storing technology. While the contents of taxonomies physically reside inside a MySQL [MYSQL] database, the OpenRDF libraries are used for efficient *to* and *from* translations of OWL documents, triples and final relational data model.

Finally, the developer of experiments is able to use the models through the **Ontology Browser Plugin** that is deployed inside the Experiment Planning Environment (EPE, see Section 3.2 of [D2.2] for a detailed description). The tool uses the Eclipse [RCP] plugin development framework to provide the graphical user interface and the Jung graph toolkit [JUNG] to visualize ontology models. The client side of the Sesame library is used to contact the ontology store on-line. Currently, the last release of the plugin (0.1.2) is not only integrated into EPE but also cooperates with the Resource Browser Plugin for faster search of

remote computations (Grid Objects) that are available to the experiment developers.

The remaining part of the picture in Figure -45, the interface to the **PROToS** provenance tracking system, is not present at the moment – currently PROToS Core stores the required ontologies locally and does not query DOS online. However, both tools use exactly the same ontology models, which are developed cooperatively by both development teams (DOS and PROToS). Since both subsystems use exactly the same ontology models the integration on the level of common understanding of important concepts is achieved.

4.3.2.Current Functionality

At the current stage of development, Domain Ontology Store provides the following features:

Ontology model container and its façade:

- Full querying capabilities (all needed types of queries are supported)
- Stability and scalability is satisfactory
- The HTTP-protocol façade fulfills all the requirements of DOS

Domain models:

- Virology data model is loaded in its first version
- Current model is based on the RegaDB data schema proposed by the partners from KULeuven
- Development of concept-to-data mapping models for data sources is started

Ontology Browser Plugin:

- Able to browse, reload, and change models
- Able to search Grid Objects using semantic meaning annotations attached to these objects (input and output parameter search is supported)
- Well integrated with both Experiment Planning Environment and the Resources Browser Plugin

4.3.3.Planned Functionality

Further development includes:

Domain models:

- The final data model for ViroLab (when the common ViroLab data schema is finally decided upon), based on current stub
- A complementary model of experiment activities and computations
- Mapping models for better ontology-to-database integration (needed for planned data querying support)

Ontology Browser Plugin:

- Integration with the Data Access Client façade (when available) to gather introspective information on data sources and help developers query them using ontologies

- Further integration with the Grid Resources Browser to support searching for Grid Objects also using the future model of experiment activities

4.4.Grid Operation Invoker

Grid Operation Invoker (later referred to as GOI) is responsible for providing a uniform interface for invocation of operations in various communication protocols. It enables high level Grid programming because it provides means to access software deployed on remote computational resources, regardless of the middleware technology, thus GOI facilitates creating complex experiments.

To fulfill its responsibilities, GOI should support all leading technologies, such as Web Services, Grid components or jobs. It is necessary that GOI can be easily extended in order to support emerging middleware technologies, therefore technology adapters are loaded during runtime. This mechanism as well as the full design of GOI, including use case analysis, module decomposition, sequence diagrams, and external dependencies can be found in Section 6.1.5 of [D3.2].

4.4.1.Implementation Description

The architecture of Grid Operation Invoker is depicted in Figure -46. All components within Grid Operation Invoker are provided by this module. Registry and Optimizer are external to GOI.

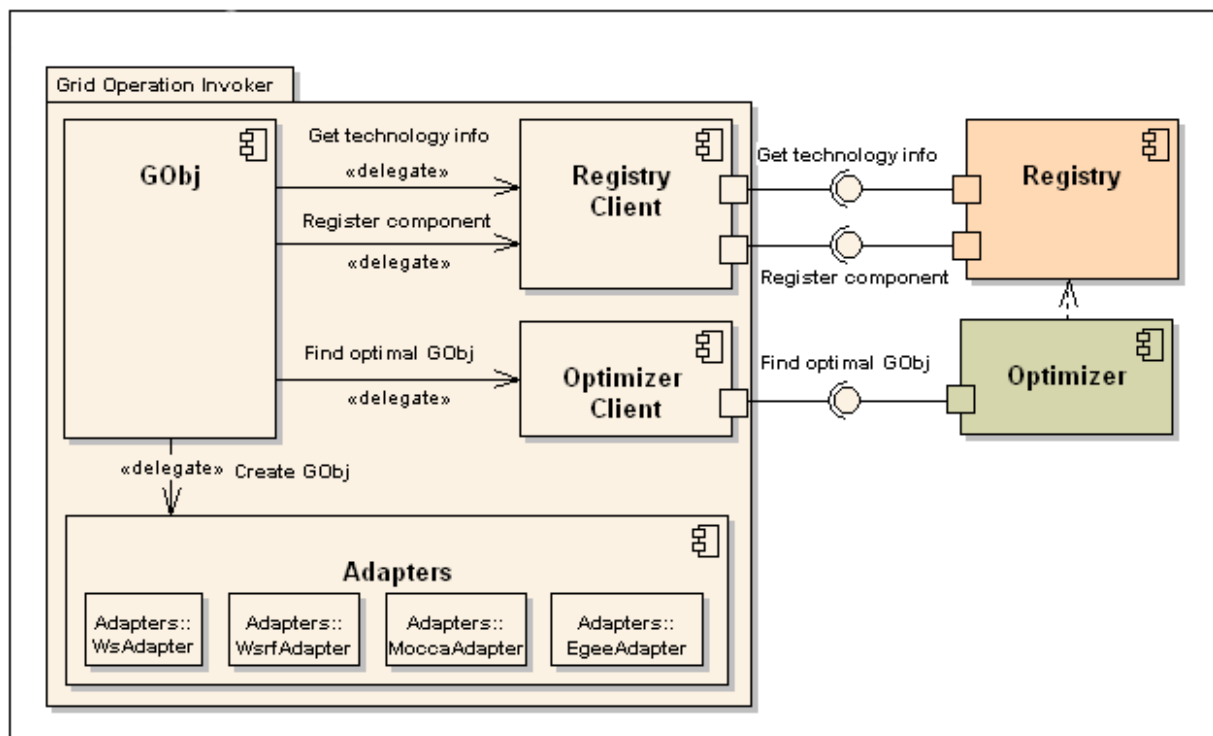


Figure -46: Grid Operation Invoker decomposition diagram and its external dependencies.

GObj is an interface for creating objects providing functionality of remote software. It queries the **Optimizer** (see Section 4.6) for the id of an optimal Grid Object Instance of the class requested in the script (experiment) by the user. Next, it queries the **Registry** (see Section 4.2) for technical information describing the selected instance. Finally, the **GObj** loads an appropriate technology adapter and delegates creating an GridObject to it. Both, the

Registry Client and the **Optimizer Client** delegate queries from the **GObj** to the **Registry (Grid Resource Registry)** and the **Optimizer (Grid Application Optimizer)**.

The Grid Operation Invoker is implemented in JRuby [JRUBY]. All components and objects are Ruby [RUBY] classes but some of them import Java objects.

The GOI is designed and implemented in accordance with object-oriented programming paradigm and uses Adaptor, Proxy and Abstract Factory design patterns from [DPIR].

4.4.2.Current Functionality

At the current stage of development, Grid Operation Invoker enables invocation of Grid Operations within the experiment on instances published in the following technologies:

- Web Service
- MOCCA
- Jobs on EGEE
- WTS (see [WTS])

While creating a representative for a Grid Object Instance, user is offered three options:

- Create a representative for an instance of a given Grid Object Class, which is selected on the basis of provided functionality. User provide the name of Grid Object Class and GrAppO finds the optimal instance.
- Create a representative for a concrete instance by providing the id of the instance.
- Create a representative using low level API that requires technical information.

Further information on GOI API can be found in Section 5.3 of [D3.3DEV].

4.4.3.Planned Functionality

Next efforts will be targeted at adding support for more middleware technologies. This will be achieved by implementing adapters for WSRF, AHE and UNICORE/DEISA (optional). In addition, GOI will provide event notifications, such as informing about operation invocation or deployment of new MOCCA component, to monitoring infrastructure. This data will be then used by PROToS (see Section 6) and GrAppO (see Section 4.6). Finally, we are working on integrating GOI with security infrastructure (see Section 5 of [D2.2]).

4.5.Generic Data Access Client

The purpose of the Generic Data Access Client (hereafter called the Data Access Client or DAC for short) is to integrate data imported from various types of data sources with the ViroLab Virtual Laboratory architecture and to present experiment developers with a uniform way of accessing and manipulating data elements in their experiment scripts.

4.5.1.Implementation Description

As presented in [D3.2], the DAC – within the ViroLab context – operates with two types of data sources:

- Standalone databases
- ViroLab hospital data sources, covered by the Data Access Service which is under development at HLRS.

The implementation of DAC envisions that all data sources (be it external databases or the DAS) should be accessible with a simple and generic interface within the experiment scripts so that experiment developers do not need to involve themselves with the technical aspects of communicating with actual services and databases used to expose the data they are working with.

The functionality of DAC is implemented by a DACConnector Class, which is in effect a factory for the creation of Data Source Objects (see the diagram in Figure -47). All such objects are treated as experiment resources and are available to the experiment developer (requestor) within the context of the application script.

Connectivity with the Data Access Service is ensured by a WSRF client link, with precompiled stubs for the service deployed at HLRS. The Data Access Service further overlays the linked data sources with an OGSA-DAI database aggregation, and facilitates multi-source queries with the use of a specialized language. While this functionality is still under development, the DAC can formulate proper queries to the DAS in order to retrieve data from data sources (please see [OGSADAI] for further information).

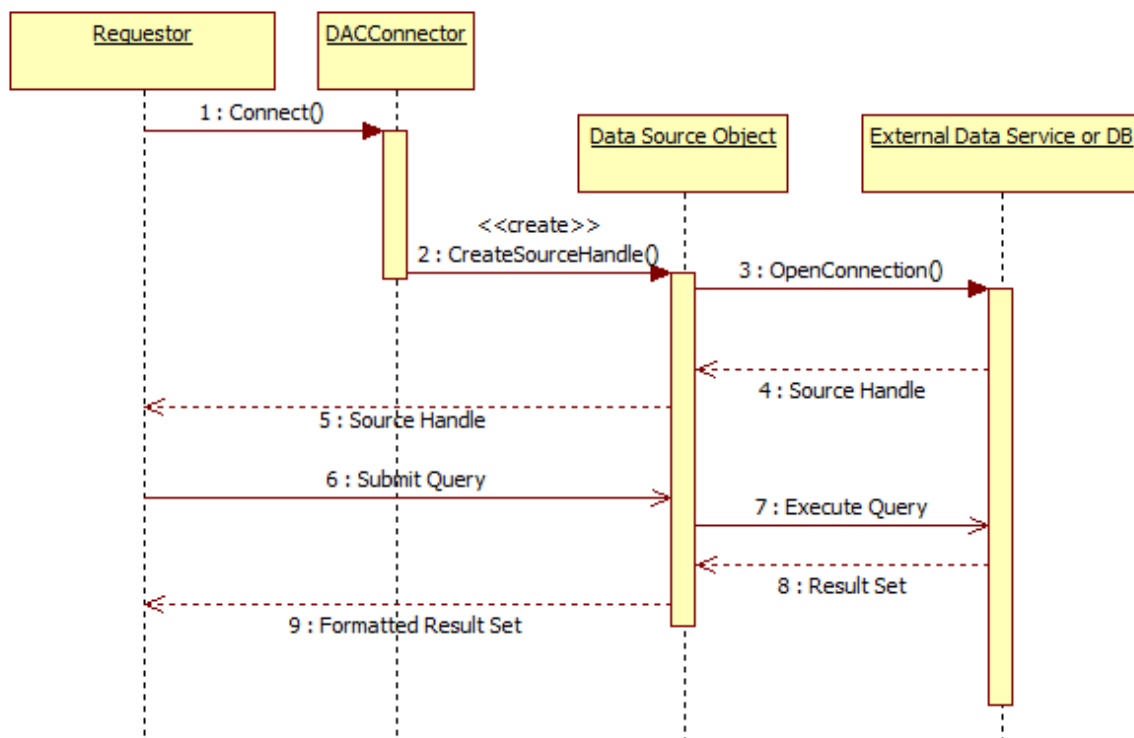


Figure -47: Handle creation and query process inside DAC.

A Data Source Object is capable of querying the data source which it represents and returning results to the user. Results can be formatted in various ways, at the developer's behest - the default mode is returning result data as 2D Ruby structures (lists of lists, as Ruby does not directly support an "array" datatype).

Implementation of DAC has made use of the following technologies:

- JDBC for access to standalone databases (enabled through JRuby/Java integration),
- Apache WSRF for access to the Data Access Service (powered by the Apache Axis framework with WS-Addressing support and service stubs)
- JRuby tools for constructing queries and formatting data

4.5.2.Current Functionality

The current version of DAC

- has the first version of DAC integrated with VL runtime
- is capable of querying standalone databases and interfacing with DAS
- Uses JRuby [JRUBY], JDBC [JDBC] and Axis/WSRF [AXIS] (as explained above)

Queries can be submitted to standard types of relational databases (including MySQL and PostgreSQL), as well as to DAS data sources directly. Moreover, special methods supplied by the DAS are covered (such as **RequestRuleSets**) and can be invoked within VL scripts on DAC connector objects. All queries are sanitized to prevent SQL injection attacks.

Results can be returned as plain data structures (typically lists of lists – for 2D SQL data tables) or as JRuby objects with getter/setter functions and methods to iterate through result sets.

4.5.3.Planned Functionality

Current work concentrates on a uniform representation of DAC Data Objects (i.e. any type of data retrieved from a Data Source). By creating an object-oriented framework for the representation of Data Objects in experiment scripts, further integration of diverse data storage mechanisms could be achieved, and furthermore, it would become possible to store persistent Data Objects which could be accessed in between execution of scripts.

Work is also underway on the following issues:

- Dynamic loading of DAS interfaces and client instantiation
- Development of a Data Access plugin
- Work towards a unified VL data resource repository

4.5.4.Deviations from the Design Document

The sequence diagram presented above has changed somewhat with respect to the plans included in [D3.2]. This is the result of better understanding of user needs and discussions on how to ease the integration of DAC with the Virtual Laboratory runtime system. However, the underlying principles remain unchanged.

4.6.Grid Application Optimizer

As explained in Section 4.4, the Grid Operation Invoker is responsible for invocation of Grid Object's operations, however, the component itself is not able to decide on which Grid Object Instance (for the terminology related to Grid Objects please see Section 4.2 of [D3.2]) the operation is going to be invoked. For the purpose of making decisions, which Grid Object Instance is optimum to be used by Grid Object Invoker, the Grid Application Optimizer (in short GrAppO) was created.

On the base of information obtained from the Grid Resources Registry, the Monitoring Service and the Provenance Service, the Grid Application Optimizer chooses an optimum solution – either a ready Grid Object Instance or Grid Object Implementation with a resource on which its instance (GOB Instance) can be created.

In order to support multiple optimization criteria and strategies, the GrAppO provides a configuration mechanism – Optimization Policy.

The details of Grid Application Optimizer design and communication channels were provided in Section 5.1.7 of [D3.2] with changes specified in Section 4.6.4 of this document.

4.6.1.Implementation Description

At the current stage of development Grid Application Optimizer operates in its simplest mode – short-sighted optimization with a basic algorithm.

As depicted in Figure -48, optimization is requested by **Grid Operation Invoker** – through a local connection. After receiving such a request, which contains a specification of the Grid Object Class to be used, the main GrAppO component – the **GrAppO Manager** obtains all necessary data concerning this Grid Object Class from **Grid Resources Registry** (via a remote connection, using SOAP). The data includes information about all Grid Object Implementations of the required GOB Class and their instances. The information are then passed to **Optimization Engine** with a request for optimization. The optimization itself is performed by this component with regard to the configuration specified by the **Optimization Policy**, which includes an optimization algorithm to be used and options pertaining to the strategy of optimization. Depending on the strategy, a request for estimation of possible solutions can be performed by **Performance Predictor**, which will be able in the future to use data obtained from Monitoring Service and Provenance Service (see Section 4.6.3).

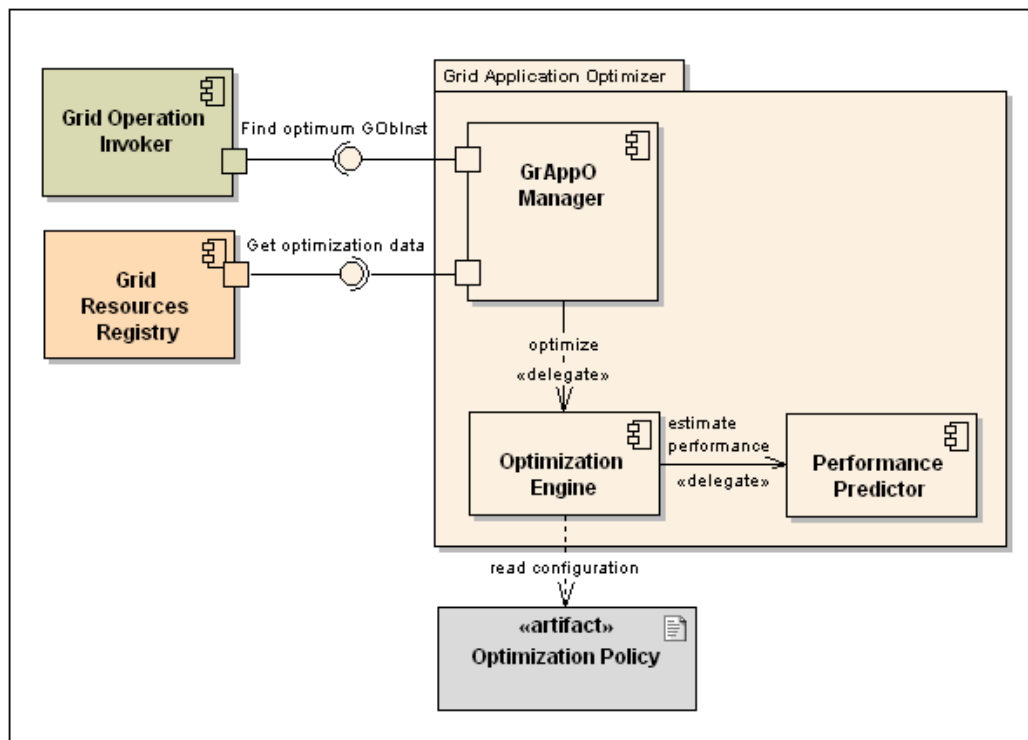


Figure -48: Grid Application Optimizer decomposition diagram – currently implemented GrAppO main components, connected to other runtime modules: GOI and GRR

The **Optimization Policy** can be passed either as a ready Java Object or as a location of an XML file storing the configuration for the Grid Application Optimizer constructor.

All the components of the Grid Application Optimizer are implemented in Java [JAVA]. For the implementation of a connection to the Grid Resource Registry XFire [XFIRE] libraries were used. JDom [JDOM] libraries were utilized for reading XML configuration files.

4.6.2.Current Functionality

The current version of GrAppO offers:

- Stateless optimization with use of simple service selection policy
- Integration with Grid Resources Registry (connection through SOAP) and Grid Operation Invoker (local connection)

4.6.3.Planned Functionality

In the future releases of GrAppO, the following features will be included:

- Integration with monitoring and provenance – data obtained from these components are vital for more advanced optimization meaning a more accurate service choice
- Implementation of advanced optimization algorithms – if the data from monitoring and provenance are available, some more advanced algorithms can be used for optimization
- Development of far-sighted optimization – although optimization of the whole application at a time can bring some improvement to the application's performance, it requires all the previously mentioned featured

and some powerful mechanisms for the application's structure analysis. However, since the solution improvement is not guaranteed, this feature has the lowest priority

4.6.4.Deviations from the Design Document

In comparison to [D3.2, Section 5.1.7], the following changes were introduced to Grid Application Optimizer:

- The name of the component is changed – the name 'Scheduler' or 'Grid Resource Scheduler' used previously is replaced by 'Grid Application Optimizer' or simply 'Optimizer'. This is because the functionality of Grid Application Optimizer focuses on finding an optimum service, which is much narrower than functionality of a typical scheduler, thus the name 'optimizer' seems more accurate. Along with the name of the component, the term 'scheduling' is replaced by 'optimization' and the names of modes in which the component may operate are changed to 'short-sighted optimization' (instead of 'simple scheduling') and 'far-sighted optimization' (instead of 'pre-scheduling')
- An external dependency added – a new communication channel to the Middleware Provenance Service will be created in order to gain information about performance of the services from which the GrAppO should choose the optimum in previous invocations.

5. Data Virtualization and Access

Data virtualization and access shall hide the underlying data resource technologies from the users by providing specific virtualization services forming a single point of access and using standard user interfaces. These interfaces shall encapsulate several functionalities for querying, delivering, and transforming data but shall also make those interactions secure and reliable. Therefore, different modules have been independently designed, which offer appropriate capabilities based on standard web service interfaces to their users, while all of them together compose a sophisticated data management system for accessing distributed and at the same time heterogeneous data resources.

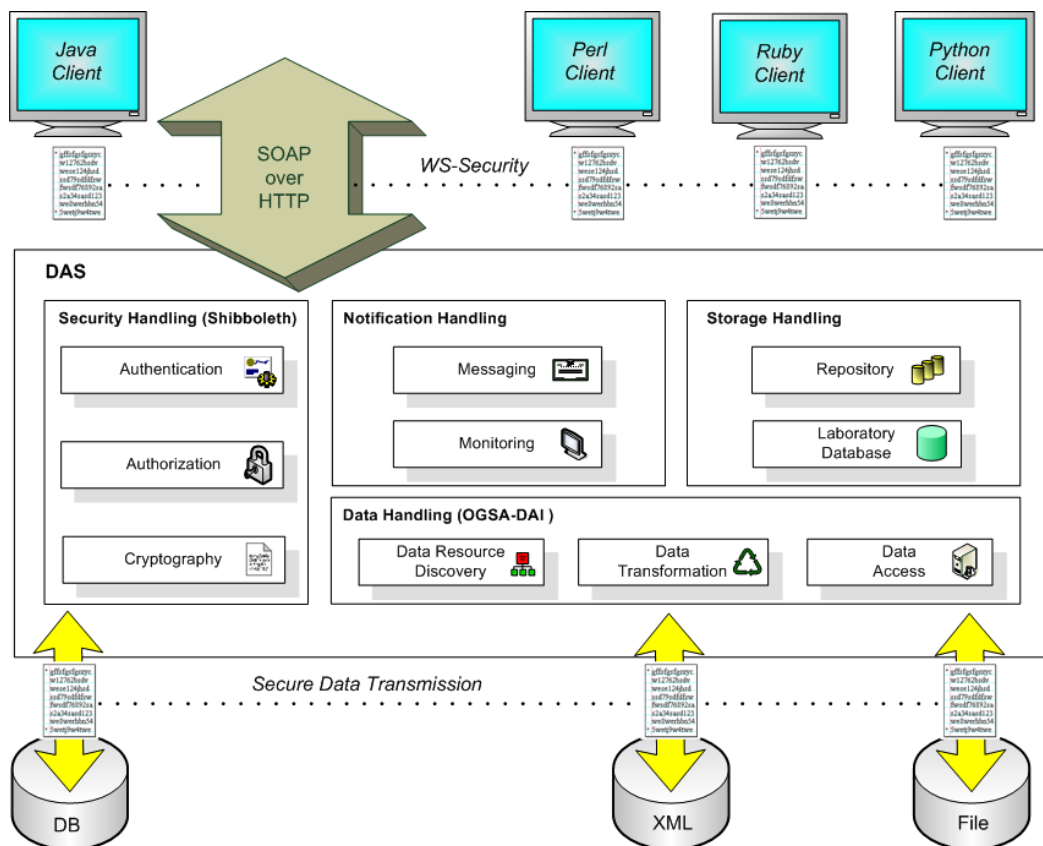


Figure -1: Overall data management architecture.

5.1.Data Access and Handling

The data access and handling infrastructure is the most important part of the virtualization layer. It shall provide interfaces to access different types of resources - including relational and XML databases as well as files - in a transparent and consistent way and allow users to perform different activities on these resources, like querying, updating, and delivering data. Therefore, the OGSA-DAI [OGSADAI] framework provides various data resource-dependent data handling activities - so called Data Resource Accessors - to connect with corresponding types of resources. With these accessors one is able to access the data resources in a common way basically using standard SQL statements.

5.1.1.Implementation Description

Most of the interfaces provided are directly connected with corresponding OGSA-DAI interfaces. Figure -492 depicts the specific use case where a user wants to invoke a query using the interfaces of the DAS. On the right side of the picture, the involved components of OGSA-DAI and their interactions are listed. Based on the request, different activities are performed after an authorization mechanism has granted access to them. Those are typically connected with one data resource accessor, which uses a database-dependent driver to establish a connection with the underlying resource.

When dealing with multiple data providers, each of them usually has its own installation of a data access system including the data access service linked with an OGSA-DAI data service. The coordination of all these single systems requires one central entry point, which acts as the only “visible” and accessible data access system, and which hides all other data access systems from the users. In theory users should be unaware that they are using a federation rather than a single data resource. Currently, the DAS offers one specific functionality that handles a federated query. The main operations are similar to the ones shown in Figure -492 with the difference that this has to be done many times.

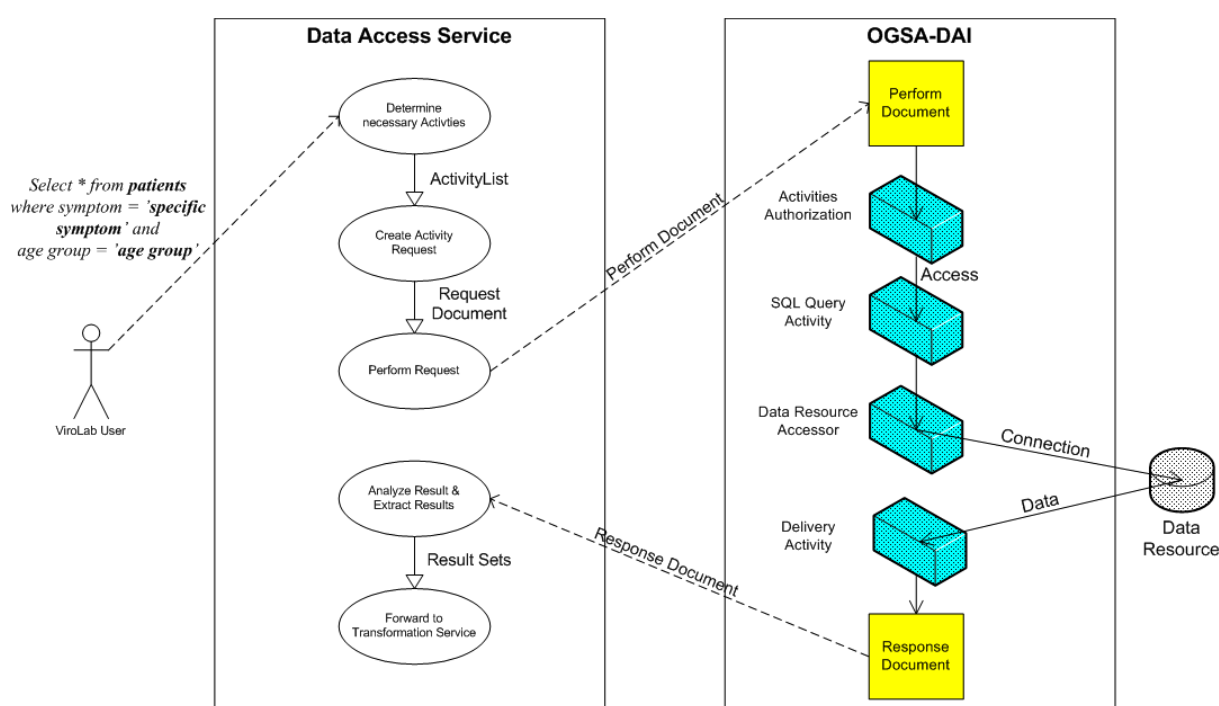


Figure -492: DAS use case of a typical data access request and OGSA-DAI interactions.

5.1.2.Current Functionality

The module currently offers a set of different functionalities for accessing distributed data resources. The basic features allow the interaction with underlying databases in a common way but also provide specific methods such as distributed queries, download of publicly available rule sets, and more. Details on the interfaces currently available are described in [DAS-API].

For more information about their usage and their functionality, please refer also to the DAS manual provided in the appendix of this document.

5.1.3.Planned Functionality

The following functional enhancements are planned for future releases:

- Parallelization of distributed queries: An algorithm that enables parallel processing of a distributed query – currently done in a sequential order - to increase performance and reliability of complex user queries
- Application-specific transformations that transform data or data formats for specific needs of ViroLab applications

5.2.Data Resource Discovery

Data resource discovery virtualizes the location of data resources and forms one of the basic services. Applications specify data resources in terms of logical names qualified by predicates – using a so called Meta Query Language (MQL) - rather than giving exact queries. The discovery service maps this terms onto actual data resource-dependent statements. A query can be, for example "*Find records for patient with specific symptom and in specific age group*". Discovery then involves the mapping of the logical name "*patient*" with the predicates on "*age*" and "*symptom*" into statements and locations of data resources (e.g. hospitals) that could contain relevant records.

5.2.1.Implementation Description

The current implementation is mainly based on standard SQL statements, which are used to obtain relevant schema information. These statements can be simply used in combination with corresponding OGSA-DAI interfaces that map the queries according to the underlying database technology. For more information about the design and implementation, please refer also to the design document (Deliverable D3.2) [D3.2].

5.2.2.Current Functionality

The current version of this module offers:

- Simple functionalities for retrieving schema information: The specifications can only be requested using corresponding SQL statements like *Describe table* for MySQL databases
- The data access interfaces must be queried using concrete SQL statements instead of an abstract query language (MQL)

5.2.3.Planned Functionality

- Meta query language to simplify communication with services: Developing a higher-level language that allows application users as well as developers to query resources without using real SQL statements but rather common well-known terms
- Functionalities that enable easy and efficient requesting of schema specifications

5.3.Security Handling (Authentication, Authorization and Cryptography)

This part of the module is basically responsible for any kind of security related issues. This includes mainly user authorization and message/communication security whereas user authentication will be handled by a different component (see Section 5.3.4). User authorization or access control shall be handled in combination with particular Shibboleth components and message encryption/decryption shall be principally guaranteed using appropriate mechanisms provided directly by the Grid middleware.

5.3.1.Implementation Description

Authorization is performed by using the standard libraries shipped with Shibboleth. They include lots of functionalities to request user attributes from corresponding home organizations. The authorization decision is currently implemented in a static way which means as long as a requester carries a specific attribute (his/her role such as medical doctor, virologist, epidemiologist, etc.), the user will then be allowed to access the resource so that he/she can query everything (every data set) provided by the resource and accessible through the current database management system user account.

The current release uses the cryptographic mechanisms provided by the Globus Toolkit in combination with OGSA-DAI. In order to ensure encrypted message transfer between the data access module and corresponding applications, one simply needs to change some values within some property files.

5.3.2.Current Functionality

An advanced feature in the current release is the integration of the services together with the authorization principle of Shibboleth. It is in an early stage of development so that the final user authorization is currently implemented in a static way, meaning that as long as a user holds specific attributes such as a role, institution, etc. he/she might be allowed to access a particular resource. There is no real dynamic procedure for access control available at the moment but future releases will also contain such a dynamic authorization model where a so-called Policy Decision Point (PDP) can be asked whether a user has the necessary attributes for accessing a particular resource.

Message Encryption is also in an early stage of development but the main communication lanes are secured using the Grid Security Infrastructure principle provided by the Globus Toolkit [GT].

5.3.3.Planned Functionality

- Dynamic user authorization: Using a PDP and user-defined policies (usually created and managed by the data providers themselves) to control the access to their own resources
- Advanced data encryption mechanisms including data stream encryption

5.3.4.Deviations from the Design Document

Due to the usage of the Shibboleth [SHIBBOLETH] technology as the basic security infrastructure component, the data management services only handle access control to particular resources whereas user authentication is directly and completely performed by the user's home organization coordinated from the corresponding Shibboleth module (IdP – Identity Provider). More information on the security principles can be found in [D2.2].

5.4.Notification, Messaging, Monitoring

The notification handling part consists of two separate modules independent of each other. The messaging service shall provide functions for distributing events such as information or error notifications and therefore needs capabilities for subscribing as consumer as well as for registering as producer in order to send/receive messages.

Observing the current status and reporting events constitutes the task of the monitoring service. This subcomponent should be able to interact with other services, mainly with the security and data handling services to monitor their behaviour.

5.4.1.Implementation Description

Since monitoring and event handling are also required by other ViroLab components and one specific component of the virtual laboratory– the Monitoring Infrastructure (M-Ring) – that is responsible for principally handling those tasks, this part has not yet been explicitly started and seen as an optional feature in the context of the DAS. The actual implementation tasks mainly concentrated on the basic functionalities of the virtualization services in order to deal with corresponding data resources. However, simple logging mechanisms have been included to monitor basic user interactions.

5.4.2.Current Functionality

Currently, the data management module makes use of the logging functionalities provided by OGSA-DAI and Globus Toolkit, which are based on the LOG4J library provided by the Apache Foundation [LOG4J]. Those include very simple mechanisms for monitoring general procedures by collecting and storing the main user interactions.

5.4.3.Planned Functionality

- Enhanced monitoring/logging functionalities particularly for any kind of data access and exchange
- Interaction with other components of the virtual laboratory providing similar functionalities such as messaging, logging etc.

5.5.Data Storage and Laboratory Database

The storage handling component does not principally provide any services for external usage but acts as a separate data resource that can be accessed via the data access component in order to store different types of processing data such as intermediate data - data produced during the execution of applications including monitoring information as well as intermediate data from running processes – or experimental data from finished application processes including their input values.

5.5.1.Implementation Description

The basic implementation activities are restricted to a simple set up of a central database – currently a MySQL database installed at USTUTT – which can be used for storing different application data such as input values and processing results.

This data store can also be directly accessed via the interfaces provided by data access module.

5.5.2.Current Functionality

The current version of Data Storage and Laboratory Database offers:

- A separate database for storing relevant application data. Currently, limited only to the DRS application but can be easily adapted also to other applications
- Stored data contains a specific key for accessing corresponding data sets directly from e.g. the provenance system

5.5.3.Planned Functionality

- Store input and output data from several applications used in ViroLab
- Store also intermediate data mainly obtained from long-time running mathematical simulations or computations

6. Provenance Tracking System – PROToS

PROToS is a knowledge-based, lightweight and fully distributed set of components responsible for gathering provenance data, exposing it for later mining and building advanced provenance queries.

From the technical point of view, the whole system was designed with storage space and query processing speed in mind (see Section 6.3.7 in [D3.2]). It features many modern solutions, such as ontology processing logic and data routing, XML storage or XQuery support. From the technical point of view it uses such technologies as Dependency Injection containers (Spring), XML persistent storage, portlet with AJAX GUI technologies and JMX based configuration utilities. PROToS design is described in details in Section 6.3 of [D3.2].

From the user's point of view we were aiming at ease of use for all types of users, both technicians and medical people. It expresses in technologies we have chosen and in Graphical User Interface (QUaTRO component, see Section 6.3.3 of [D3.2], fragment elaborating on QUaTRO).

6.1. PROToS Core

The core of PROToS is the crucial component in the overall architecture. It exposes external, Web Services based interfaces for provenance storage and retrieval, thus providing single point of provenance processing. It employs and manages hierarchical, distributed storage of provenance data, which is crucial for speed of the provenance processing tools, such as QUaTRO. Incoming data is balanced and routed using various algorithms to ensure fulfilling requirements defined at the project's beginning. Design of the component is described in great details in Section 6.3.4 of [D3.2].

6.1.1. Implementation Description

Aiming to provide a technically advanced system for provenance tracking we have chosen to use industry-standard technologies/solutions in the PROToS implementation. Some of them could be found in Section 6.3.2 of [D3.2]. The following list comprises selected technologies with short explanations.

- Web Services - XFire (<http://xfire.codehaus.org>).

XFire is a next-generation java SOAP framework, with easy to use API and including support for many standards, as WS-Security or JAX-B. It is built on a low memory footprint StAX model, so it's also a very good performer. We have chosen XFire over Apache Axis/Axis2 because of performance reasons. In test, XFire is 2 to 6 times faster as Axis with 1 - 1/5 of it's latency. We are convinced that these numbers are the best justification.

- Internal middleware - RMI

(<http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp>)

RMI stands for Remote Method Invocation and is Java-native object oriented API for RPC. While being pure Java, it's performance is very good. RMI provides also tight integration with Java security and encryption mechanisms, making good use of SSL/TLS. All mentioned reasons convinced us to choose RMI over popular Web Services based solution. Of course performance was the most important reason of our choice.

- Component container - Spring (<http://www.springframework.org>)

Spring is a container for POJO components that makes use of Dependency Injection design pattern. It enables PROToS developers to define instances that will implement specified interfaces, and 'inject' them into components that are dependent of them. This unique feature simplified PROToS design making it much more robust and cleaner. Spring is much more than DI-compliant container, being the leading java, non-EJB application framework. It offers full support for such technologies as JMX, RMI and XFire-based Web Services.

- Management technology - JMX

(<http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/>)

JMX stands for Java Management eXtensions and is the standard for managing and monitoring of Java applications. JMX is highly standardized and provides out-of-the-box tools for building modular, dynamic solutions for managing systems like PROToS. We are convinced that using this technology will saved us much time and make our system cleaner and more robust.

- Ontology API - Jena2 Ontology API (<http://jena.sourceforge.net/>)

Jena is pure Java framework for building Semantic Web applications, with excellent API for OWL language. It includes own SPARQL implementation, called ARQ with query engine and rule-based reasoner. Jena allows other reasoners to be deployed and could perform lot of PROToS ontology-related processing as validation of OWL classes and individuals. What is more, Jena supports natively reading and storing OWL in XML format, which is crucial as we have chosen XML database for our storage. Even though Jena isn't a very good performer, we believe that it's the best available framework with many useful features that would be very difficult to achieve otherwise.

- OWL Reasoners - Pellet (<http://pellet.owldl.com/>)

Pellet is the leading-edge reasoner for OWL ontologies that provides a very good performance and full compliance with OWL-DL dialect. It also provides a set of untypical features as ontology analysis and repair, species validation, and datatype reasoning, which could be used internally by PROToS.

- XML storage - eXist (<http://exist.sourceforge.net>)

eXist is an Open Source, XML native database with built-in XQuery support. It also supports XUpdate, an open language for modifying XML data. Being native XML database eXist is also a quite good performer. On the Java side, eXist supports XML:DB API that provides a common interface to XML databases. The main reason for this choice was XQuery support, as it's one of the primary assumptions for PROToS system. Another major reason was support for XML:DB API, which was chosen as main interface for the Storage Peer component, thus speeding up whole PROToS implementation process.

As mentioned earlier, and defined in the D3.2, section 6.3.4 ("PROToS Core Overview"), PROToS Core is organized as a set of components. Implementation solutions used in each component are listed below.

- The core, built on Spring and distributed as Java Web Archive (WAR), which requires Java Servlet 2.5 compliant container, as Tomcat

(<http://tomcat.apache.org/>) to operate and uses XFire based Web Services for external interfaces. Jena and Pellet are employed for ontology processing. The whole component can be configured using JMX or by hand – configuration is stored using XStream (<http://xstream.codehaus.org>) syntax. Communication with storage components is made using secured RMI.

- Storage Super Node is also Spring based, but being distributed as standard Java application (JAR) does not require managed environment to operate. Ontology processing and configuration options are the same as in the case of the core. Every instance runs own RMI naming service, exposing itself for remote access from the core component.

Storage Peer is currently implemented as sole eXist instance.

6.1.2.Current Functionality

Current functionality of the PROToS component could be viewed from two different standpoints: infrastructure (developer) and usage (end user).

Looking from the first point of view, we have managed to accomplish:

- Structure of the component with full compliance to the design presented in D.3.2, that is divided into three physical components separately distributed.
- Internal infrastructure prepared for further extensions planned by implementing suitable solutions, as design patterns: Strategy, Factory, Bridge, Adaptor and others.
- Management infrastructure, based on JMX with common model for all components. Further extensions of components will be automatically included in their management options.
- Persistent state / configuration functionality. All components are statefull and keep their state / configuration between lifecycle events.
- Common annotation-based ontology event model service, with validation based upon semantic information. Infrastructure allows for automatic service of newly generated event of standard type and defines new types, by implementing dedicated interfaces.
- Common query support.

From the second standpoint following features have been implemented:

- Gathering of provenance data provided by the monitoring infrastructure and generated by the EGT (Event Generation Tool, see section 5.2)
- Support for retrieval of provenance data by using XQuery-based queries
- Simple validation of incoming data and queries, based on the semantic information where applicable

Support for user ontologies, divided into application, data and experiment, located anywhere in the net and configurable by standard mechanisms (JMX, XML). Ontologies can be downloaded and distributed among components active in the current configuration.

6.1.3.Planned Functionality

For new releases of the PROToS we are planning significant upgrades both in the infrastructure and end-user functionalities.

The first huge improvement in the field of infrastructure is an implementation of the fully distributed storage for provenance data. Currently, PROToS accepts as many Storage Nodes as configured, but configures and uses only one of them. This improvement needs implementation of efficient data distribution algorithms (more than one is planned) and an algorithm that will allow for querying XML data distributed on many component. Although this functionality is not an easy task we believe it will take performance of the PROToS to the next level.

Next functionality we are planning to implement is broader support of the ontology processing, which will bring additional features as consistency validation of incoming provenance data and support for such query languages as SPARQL and RDQL. One more thing that will also be enabled is semantic reasoning over provenance data. This improvement is anticipated, but in our opinion in the current ViroLab Virtual Laboratory environment performance plays a more important role, thus distributed storage should be implemented earlier.

Last improvement that is planned is implementing own solution for the Storage Peer component. It will stick to our common component model with all its peculiarities, such as statefulness and JMX configuration.

6.2.Event Generation Tool

The provenance tracking system exposes an interface for experiment events deliverance. Because information is internally represented in ontologies, there is a need for a tool generating event java classes wrapping ontology individuals.

Event Generation Tool explores delivered ontology and generates event java bean classes encapsulating individual URI, data type properties, functional properties and additional attributes specific to PROToS. The generated classes are dedicated to be utilized by all components delivering events to the provenance system.

The attributes referring to ontological concepts are extended with proper annotations necessary to instantiate individuals. These annotations are used both by Semantic Event Aggregator when instantiating monitoring events and by the PROToS Core when storing individuals from incoming events.

6.2.1.Implementation Description

The Event Generation Tool is a single independent component. The generation process respects several decisions on the classes specification:

- the *serializable* interface is implemented
- both class and attribute names are identical with respect to ontological names
- setters and getters for all attributes and argumentless public constructor are provided
- functional properties are mapped to single attributes while multiple properties are mapped to collections of values
- attributes collections are implemented respecting Java *Generics* approach and are instantiated by constructor
- object properties are mapped to attributes containing a related individual URI

- all properties have a *protected* scope to enable convenient future inheritance

The java classes are related in a derivation hierarchy corresponding to an ontology concepts hierarchy. Due to such an approach, generated events encapsulate only properties defined explicitly in ontology concepts omitting inherited properties. By default, all classes derive from *Thing*. However, it is possible to configure the top class of derivation hierarchy, in mostly cases *ViroLabDataEntity*.

Every semantically valuable attribute is associated with an annotation describing its meaning. These annotations address:

- ontological class URI
- ontological property URI
- individual URI

All generated classes are placed in corresponding packages - the mapping between ontological name spaces and package names is defined in an xml configuration file.

The Event Generation Tool component is implemented in Java [JAVA]. Jena framework [JENA] is utilized to explore OWL ontologies [OWL].

6.2.2.Current Functionality

At the current stage of development Event Generation Tool offers following functionality:

- generation of annotated bean classes from ontologies respecting a derivation hierarchy
- restrictions on derivation hierarchy
- mapping between namespaces and package names
- mapping between ontological data types and Java data types

6.2.3.Planned Functionality

The future Event Generation Tool release should be mindful of the information coherency issue. In order to ensure that provenance system works with the latest version of information model, the tool should be automatically notified of changes applied to domain ontology and automatically generate and deploy a new version of the library.

6.3.Semantic Event Aggregator

Semantic Event Aggregator is a component responsible for building ontological information from monitoring events data. It provides transformation between monitoring events at two distinct levels of abstraction:

- xml files published by producers
- ontology individuals represented in OWL language

The aggregator exposes an interface for gathering xml events. The interface can be accessed directly by event producers or via monitoring middleware. There are few restrictions on delivered xml events data, the main content is expected to be augmented with event types and ACID, what can be easily achieved by utilizing

vldr – ViroLab data representation library containing proper factories and helpers.

The aggregation process highly bases on aggregation rules, described semantically. Every aggregation rule defines:

- what events should be aggregated
- how events ACID should correspond with each other
- what actions should be undertaken when the rule is sufficient

The mapping between xml data and ontological information is represented as an **ontology extension** - a dedicated ontology extending the created ontology with annotation properties, but not influencing the ontology itself. Ontology extension contains derivation concepts that define how to build the experiment ontology:

- what elements in xml hierarchy refer to ontological concepts
- what elements and attributes correspond with particular data type and object properties
- what additional operations are desirable in order to establish individual properties - these operations are expected to be encapsulated in Java classes placed in aggregator classpath by an ontology extension author

Aggregator presents ontology independence in order to enable future changes in the ontology structure.

The created ontology in form of individuals collection is enclosed in dedicated wrappers. It is then transformed utilizing events library generated by Event Generation Tool and delivered to the provenance tracking system.

6.3.1.Implementation Description

The architecture of the Semantic Event Aggregator is depicted in Figure -50.

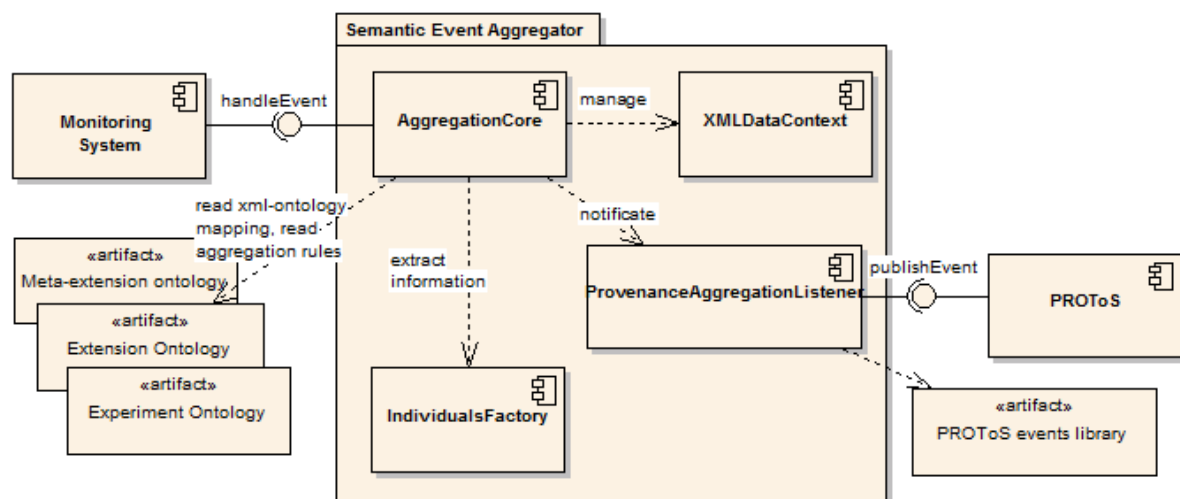


Figure -50: The decomposition diagram of the Semantic Event Aggregator

The aggregator manages **xml data context** – a current monitoring data set that has been collected but not yet aggregated. The context is realized in form of a hierarchical mapping structure which enables efficient access through addressing by ACID and event type.

The context is influenced in the following activities:

- registration of new events provided by monitoring middleware
The events are preprocessed by *EventHandler* which validates and extracts ACID before passing a document to the context
- monitoring context regarding aggregation rules
Whenever a new event is published, all the rules referring to this particular event type are checked.

When an aggregation rule is satisfied, the aggregator removes from context documents referring to aggregated events and passes the document to the **IndividualFactory**. This component instantiates individual wrappers in two steps:

- create all individuals and associate datatype properties
- discover relations between created individuals and associate object properties - this step includes the search for an xml document tree for the elements referring to a proper ontology concept

The wrappers are operated by registered listeners. Currently, only the **ProvenanceAggregationListener** is provided, which is responsible for the instantiation of PROToS events utilizing a library generated by the Event Generation Tool, however, new listeners for additional consumers are possible. It addresses the situation when other components subscribe for ontological events and possibly expect different representation.

All Semantic Event Aggregator components are implemented in Java. Jena framework is used to manage ontologies. JDom is used to parse XML documents [JDOM].

6.3.2.Current Functionality

At the current stage of development, the Semantic Events Aggregator offers following functionality:

- gathering events delivered via remote logger (log4j specification)
- aggregation of events at different levels of ACID coherency
- building of single ontology (currently, the experiment ontology is supported), including individuals data type properties
- extraction multiple individuals from single xml files and discovery of object properties
- utilizing external classes to extend aggregation process with more complex operations, such existing ontology querying

6.3.3.Planned Functionality

The future aggregator releases should offer the following functionality:

- integration with the GEMINI monitoring system via adapters to publish/subscribe a model
- define more than one ontology extension in order to build more than one ontology (currently, an experiment ontology is supported)
- enable adding properties to individuals already existing in PROToS ontology

6.4. Query Translation Tools (QUaTRO)

QUeRY TRaNslation tOols (QUaTRO), is a set of tools which allow constructing complex queries over both data and provenance repositories, expressed in terms of the domain familiar to end users of the ViroLab VL (scientists, clinicians). Each tool is equipped with a carefully designed Graphical User Interface in the form of a portlet that will be integrated in the ViroLab portal. Design of the QUaTRO GUI aims at ease of use without sacrificing ability to create complicated queries. This is possible because of extensive use of the ViroLab ontologies from all domains: data, application and experiment.

Some details of tools that we are planning to implement in the QUaTRO suite could be found in the section 6.3.3 of the D.3.2. More advanced description of the QUaTRO and its tools is available in [PROV].

6.4.1. Implementation Description

As for now we have implemented the Ontology-based Query Construction Tool, described in the publication mentioned in the previous section. This tool is more advanced version of the Ontology Query Wizard described in the D.3.2, section 6.3.3. It follows the basic ideas of the Ontology Query Wizard but allows for construction of much more complicated queries. All implementation details below refer to the Ontology-based Query Construction Tool.

Because QUaTRO aims to be part of the ViroLab Portal, tool was designed to be portlet, running on GridSphere portlet container. Thus it is distributed as Java Web Archive (WAR). Technologies used:

- Google Web Toolkit (GWT, <http://code.google.com/webtoolkit/>), the AJAX framework that allows greater interactivity of our applications GUI. One of the biggest profits of using GWT is elimination of unnecessary reloads of whole page when accessing data (for example loading input values from DB). Moreover, elements of the GUI (such as boxes and buttons) could be added or removed when needed, for example when adding new branch to a query.
- Hibernate (<http://www.hibernate.org/>). Together with DB backend used for query storage. Queries are persisted on per user basis, so no one is able to see other users queries.
- Jena Semantic Web Framework (<http://jena.sourceforge.net/>), used for loading and processing domain ontologies inside the QUaTRO application.
- XFire (<http://xfire.codehaus.org/>) used for accessing Web Services – based components as the PROToS Core.
- DAC (Data Access Client, <http://gforge.cyfronet.pl/projects/dac>) used by the tool for accessing and loading application data.

Whole project and all of its sub components are managed by Maven, thus simplifying such tasks as dependencies, build and installation.

6.4.2. Current Functionality

Application at its current stage is nearly fully-functional. It allows user to construct query in an easy, natural way. Query could be constructed from ontology concepts or properties, application data (loaded in real-time by DAC) and value operators, such as equality or 'contains'. Query tree can be extended at any level by joining new branches – suitable operator 'and' is available in the

GUI. Constructed query trees are automatically converted to the XQuery form and sent to the PROToS Core. Retrieved results are rendered and presented to user by adequate portlets. User can also store queries for later use and load previously saved ones. Storage could be permanent or temporary – GUI provide so called 'quick slots' for the second type of storage. Queries loaded from persistent storage are presented in an editable, tree-like form and can be changed before reuse.

All options required for the tool operation could be configured by XML files. Currently configuration is divided into following files:

- QUaTRO core configuration. Allows to choose which ontologies will be loaded into GUI, configure ontology – data mappings and so on.
- Persistent storage configuration – database backend details for the DAC Client (QUaTRO-DAC). Configuration of the DAC service to be used, data sources and details, including databases available and connection credentials.

6.4.3.Planned Functionality

At present we are experiencing some minor bugs in our response-rendering code, which are to be eliminated as fast as possible. Next we plan to take our code to the production quality. This process will include depth unit and functional testing. Also some refactoring will be applied, oriented on cleaning our code and implementing standard design patterns where applicable.

For next releases of the "Ontology-based Query Construction Tool" we plan many improvements. Most important ones are listed below.

- New class of value operators, such as 'count' or 'at least'. This kind of operators is known as 'aggregation operators' and currently is partially implemented by the QUaTRO logic, but not available in the GUI yet.
- Ability to define relation (expressed by an operator) between attributes, residing in different branches of a query. At present, only relation between attributes and constant values can be expressed.
- Ability to traverse ontology graph in both ways. This will be done by defining and handling of the reverse properties for ontology concepts.

All mentioned above improvements are fruits of countless meetings with possible end users and our analysis of possible, real-world queries that could be constructed in this QUaTRO tool.

Our long-term plans include designing and implementing more than one QUaTRO tool. Especially we are preparing to build second tool described in the section 6.3.3 of the D.3.2 document, namely 'Language-to-query Translation Tool'. Although this tool will be definitely not an easy task, we believe that it will enable end users to define and process more complicated queries in easier way.

7. List of Virtual Laboratory Manuals

The main body of this deliverable is accompanied by a set of appendices that provide more detailed information on the tools and components building the first prototype of the virtual laboratory. These appendices are structured as manuals and are targeted for specific user groups. The list contains:

- Appendix 1: *Experiment Users' Manual*
- Appendix 2: *Experiment Developers' Manual*
- Appendix 3: *Virtual Laboratory Developers' Manual*

8. Summary

This document presents the core components of the ViroLab WP3 and the status of their implementation after 12 months of the project. It also presents experiments which can be executed using the current structure of the ViroLab Virtual Laboratory. For each component, a list of implemented features was presented. If the implementation of a component strayed from the design description outlined in D3.2, the appropriate rationale was also provided. This document is expected to serve as a report on the state of the ViroLab Virtual Laboratory and also to present guidelines and plans for future developments of the system within the scope of ViroLab.

As described in the design deliverable, we follow a phased approach, implementing the most crucial components early on in order to progress to more advanced elements of functionality at later stages of development. Implementation progress as well as any emerging problems and opportunities will be described in subsequent WP3 deliverables.

The work presented within this deliverable is also covered by a number of publications prepared in parallel with project development, as detailed below:

- Publications on the ViroLab Virtual Laboratory in general
 1. *Virtual Laboratory in ViroLab*, Marian Bubak, Tomasz Gubala, Maciej Malawski, Marek Kasztelnik, Tomasz Bartynski, Piotr Nowakowski; Cracow Grid Workshop CGW'06
 2. *From Molecule to Man: Decision Support in Individualized E-Health*, Peter M.A. Sloot, Ilkay Altintas, Marian Bubak, Charles A. Boucher; IEEE Computer Society, vol 39, no.11, pp. 40-46, Nov., 2006
 3. *The ViroLab Virtual Laboratory for Viral Disease Treatment*, M. Bubak, T. Gubala, P. Nowakowski; iSTGW bulletin (submitted)
- Publications on specific parts of the virtual laboratory
 1. *GScript Editor as a Part of the ViroLab Presentation Layer*, Włodzimierz Funika, Piotr Pegiel; Cracow Grid Workshop CGW'06
 2. [*Optimization of Grid Application Execution*](#), Joanna Kocot, Iwona Ryszka; Master of Science Thesis supervised by Marian Bubak; AGH University of Science and Technology, June 2007, Krakow, Poland; [See presentation](#)
 3. [*Monitoring of Component-Based Applications*](#), Eryk Ciepiela; Master of Science Thesis supervised by Marian Bubak; AGH University of Science and Technology, June 2007, Krakow, Poland
- Publications related to the GridSpace platform
 1. [*GridSpace - Semantic Programming Environment for the Grid*](#), Tomasz Gubala, Marian Bubak; 6-th International Conference on Parallel Processing and Applied Mathematics PPAM'2005, LNCS 3911, pp. 172-179, 2006
 2. [*Experiments with distributed component computing across Grid boundaries*](#) Maciej Malawski, Marian Bubak, Michał Placek, Dawid Kurzyniec, and

Vaidy Sunderam. In Proceedings of the HPC-GECO/CompFrame workshop in conjunction with HPDC 2006, Paris, 2006.

An up-to-date list of Virtual Laboratory-related publications can be found at <http://virolab.cyfronet.pl>.

Abbreviations

Abbreviation/Term	Explanation
AAS	Aminoacid Sequence
ACID	Application Correlation Identifier
API	Application Programmer's Interface
ARID	Application Run Identifier
CCA	Common Component Architecture
DAC	Data Access Client
DAS	Data Access Services
DB	Database
DEISA	Distributed European Infrastructure for Supercomputing
DGE	Data Gathering Engine
DOS	Domain Ontology Store
DRAM	Drug Resistance Associated Mutations
DRE	Data Retrieval Engine
DRS	Drug Ranking System
DS	Distributed Storage
DSS	Decision Support System
EGEE	Enabling Grids for e-Science in Europe
EMI	Experiment Management Interface
EPE	Experiment Planning Environment
EPL	Experiment Planning Language
FLOWR	For-Let-Where-Order by-Return
GOB	Grid Object Class
GOBI	Grid Object Instance
GOBID	Grid Object Identifier
GOBImpl	Grid Object Implementation
GOp	Grid Operation
GOI	Grid Operation Invoker
GrAppO	Grid Application Optimizer
GRR	Grid Resources Registry
GSEngine	GridSpace Engine
GT	Globus Toolkit
GUI	Graphical User Interface
HIV	Human Immunodeficiency Virus
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment

Abbreviation/Term	Explanation
IEEE	Institute of Electrical and Electronic Engineers
IoC	Inversion of Control
JMX	Java Management Extensions
JSR	Java Specification Request
JVMTI	Java Virtual Machine Tool Interface
LCG	LHC Computing Grid
LHC	Large Hadron Collider
LOB	Large Object
MLA	Mutation List Analysis
MQL	Meta Query Language
MVC	Model-View-Controller
M-Ring	ViroLab Virtual Laboratory Monitoring Infrastructure
NS	Nucleotide Sequence
OGSA	Open Grid Services Architecture
OGSA-DAI	Open Grid Services Architecture – Data Access Integration
OGSA-DQP	Open Grid Services Architecture – Distributed Query Processing
OO	Object-Oriented
OR	Object-Relational
OWL	Web Ontology Language
QUaTRO	Query Translation Tools
PDP	Policy Decision Point
PROToS	Provenance Tracking System
RAD	Rapid Application Development
RBAC	Role-Based Access Content
RDF	Resource Description Framework
RDQL	Resource Description Framework Data Query Language
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SCM	Source Code Management
SN	Storage Node
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSH	Secure Shell
SSL	Secure Socket Layer
SSN	Storage Super Node
SSO	Single Sign-On
SVN	Subversion

Abbreviation/Term	Explanation
TLS	Transport Level Security
UI	User Interface
UML	Unified Modeling Language
URI	United Resource Identifier
URL	Unified Resource Locator
UTF8	8-bit Unicode Transformation Format
VL	Virtual Laboratory
VM	Virtual Machine
VO	Virtual Organization
VPN	Virtual Private Network
WP	Workpackage
WS	Web Service
WS-I	Web Services Integration
WSDL	Web Services Definition Language
WSRF	Web Services Resource Framework
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language

References

- [AXIS] The Apache Axis project, a Java platform for creating and deploying Web Services applications, <http://ws.apache.org/axis/>
- [BIOMED] Assel M., Krammer B., Loehden A. *Management and Access of Biomedical Data in a Grid Environment*. HLRS – High Performance Computing Center of University Stuttgart, Cracow Grid Workshop, 2006
- [D2.1] ViroLab Project. *D2.1 – State of the art Survey, Design and Workpackage Specification*. ViroLab Project Consortium, 2006
- [D2.2] ViroLab Project. *D2.2 – Architecture for Presentation Layer. VO Pilot Deployment with Basic Middleware for Data Access, Resource Management and Information*. ViroLab Project Consortium, 2007
- [D3.1] ViroLab Project. *D3.1 - State of the art Survey, Design and Workpackage Specification*. ViroLab Project Consortium, 2006
- [D3.2] ViroLab Project. *D3.2 – Design of the Virtual Laboratory*. ViroLab Project Consortium, 2007
- [D3.3USR] ViroLab Project Consortium: *Deliverable 3.3 Appendix 1: Experiment Developer Tools Manual*, August 2007
- [D3.3DEV] ViroLab Project Consortium: *Deliverable 3.3 Appendix 1: Experiment User Tools Manual*, August 2007
- [D3.3VLDEV] ViroLab Project Consortium: *Deliverable 3.3 Appendix 1: ViroLab Runtime Components Documentation*, August 2007
- [DAS-API] Matthias Assel: *Data Access Service Application Programming Interface*, <http://www.hlr.de/organization/ds/projects/virolab/dasapi/index.html>
- [DPIR] *Example design Patterns in Ruby* on <http://www.rubygarden.org/>
- [GT] The Globus Toolkit Homepage. <http://www.globus.org/toolkit>
- [H2O] Pawel Jurczyk, Maciej Golenia, Maciej Malawski, Dawid Kurzyniec, Marian Bubak, and Vaidy S. Sunderam. *A system for distributed computing based on H2O and JXTA*. In Cracow Grid Workshop 2004, Kraków, Poland, 2004
- [HIBERNATE] Hibernate Relational Persistence for Java and .NET, www.hibernate.org
- [IOC] Inversion of Control Containers and the Dependency Injection pattern

	http://www.martinfowler.com/articles/injection.html
[JAVA]	Sun Corporation, Java Programming Language, http://java.sun.com
[JAVALL]	Denis Caromel, Wilfried Klauser and Julien Vayssi`ere, <i>Towards seamless computing and metacomputing in Java</i> , Concurrency Practice and Experience, volume 10, number 11-13
[JDBC]	Java Database Connectivity, http://java.sun.com/javase/technologies/database/
[JDOM]	JDOM XML parser, http://www.jdom.org/
[JENA]	Jena – a Semantic Web Framework, http://jena.sourceforge.net
[JRUBY]	JRuby – Java powered Ruby implementation, http://jruby.codehaus.org
[JUNG]	JUNG — the Java Universal Network/Graph Framework, http://jung.sourceforge.net/
[LOG4J]	Apache logging service, http://jakarta.apache.org/log4j/
[MOCCA]	Maciej Malawski, Dawid Kurzyniec, and Vaidy Sunderam, <i>MOCCA – towards a distributed CCA framework for metacomputing</i> . In Proceedings of the 10th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS2005), 2005
[MOCCA_DIST]	MOCCA homepage http://www.icsr.agh.edu.pl/mambo/mocca
[MYSQL]	MySQL Open Source Database, MySQL AB 1995-2006, http://www.mysql.com/
[NITRO]	Nitro project, http://www.nitroproject.org
[OGSADAI]	The OGSA-DAI Project. http://www.ogsadai.org.uk/index.php
[OWL]	M.K. Smith and Ch. Welty and D.L. McGuinness (eds.), <i>OWL Web Ontology Language Guide</i> , W3C Recommendation 10 February 2004, http://www.w3.org/TR/owl-guide/
[PROTÉGÉ]	Stanford University, Protégé knowledge-base framework, http://protege.stanford.edu/
[PROV]	B. Balis, M. Bubak, J. Wach <i>User Oriented Querying over Repositories of Data and Provenance</i>
[RDF]	F. Manola and E. Miller (eds.), <i>RDF Primer</i> , W3C Recommendation 10 February 2004, http://www.w3.org/TR/rdf-primer/
[RDFSCHEMA]	D. Brickley and R.V. Guha (eds.), <i>RDF Vocabulary Description Language 1.0: RDF Schema</i> , W3C

	Recommendation 10 February 2004, http://www.w3.org/TR/rdf-schema/
[RCP]	Rich client platform (RCP) applications http://www.eclipse.org/community/rcp.php
[RDQL]	A. Seaborne, <i>RDQL - A Query Language for RDF</i> , W3C Member Submission 9 January 2004, http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/
[RMIX]	Dawid Kurzyniec, Vaidy Sunderam, <i>Semantic Aspects of Asynchronous RMI: the RMIX Approach</i> , (Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04))
[ROR]	Ruby on Rails open-source web framework, http://www.rubyonrails.org
[RUBY]	Ruby language, http://www.ruby-lang.org
[SESAME]	Sesame RDF Framework, Aduna and NLnet Foundation 1997-2006, http://openrdf.org/
[SHIBBOLETH]	The Shibboleth project. http://shibboleth.internet2.edu
[SOAP]	Simple Object Access Protocol, http://www.w3.org/TR/soap/
[SPRING]	Spring Framework, www.springframework.org
[SVN]	Subversion, version control system, http://subversion.tigris.org/
[SVNKIT]	SVN Kit, pure Java Subversion implementation http://svnkit.com/
[VIROLAB]	The ViroLab Project Website. http://www.virolab.org
[VIROLAB-VL]	The ViroLab Virtual Laboratory Website. http://virolab.cyfronet.pl/
[WEKA]	Weka Data Mining Toolkit website http://www.cs.waikato.ac.nz/~ml/weka
[WTS]	Pieter Libin, Bart De Deckere, Joris Van Santvoort: <i>Wts: a stateful web service infrastructure</i> , http://wts.sf.net/
[XFIRE]	XFire, http://xfire.codehaus.org
[XPATH]	XML Path Language Website. http://www.w3.org/TR/xpath
[XSL]	XSL Transformations. http://www.w3.org/TR/xslt