VIR LAB                                                    @Health

VIR LAB

# Deliverable D3.4

# Integration of presentation layer and session manager, workflow provenance system and process flow template - 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: | 29-02-2008 |
|---|---|
| Delivery: | 04-03-2008 |
| Lead Partner: | ACC CYFRONET AGH |
| Dissemination Level: | Public |
| Status: | Final |
| Approved: | Steeting Committee, Quality Board |
| Version: | 1.0 |

# Log of Document

| Version | Date | Changes Summary | Authors |
|---|---|---|---|
| 0.1 | 23/01/2008 | Table of contents proposition | Tomasz Gubala |
| 0.2 | 24/01/2008 | Fixed content, ready for distribution inside Consortium | Marian Bubak, Tomasz Gubala |
| 0.3 | 18/02/2008 | Introductory part (2.) added | Tomasz Gubala |
| 0.4 | 19/02/2008 | Added figures to part 2. | Tomasz Gubala |
| 0.5 | 25/02/2008 | Added Application Repository Client status description | Joanna Kocot |
| 0.6 | 26/02/2008 | Added description of the GOI and computation access security mechanism | Tomasz Bartyński |
| 0.65 | 26/02/2008 | Included summaries | Piotr Nowakowski |
| 0.7 | 26/02/2008 | Added initial description of monitoring, GRR and GrAppO integration | Bartłomiej Łabno |
| 0.8 | 27/02/2008 | Added section reporting the current status of GSEngine | Eryk Ciepiela |
| 0.9 | 27/02/2008 | Added section reporting the current status of Grid Resources Registry | Marek Kasztelnik |
| 0.10 | 27/02/2008 | Added section reporting integration of GSEngine with EPE and EMI | Eryk Ciepiela |
| 0.11 | 27/02/2008 | Added description of DAC | Piotr Nowakowski |
| 0.12 | 27/02/2008 | Updated publications list | Tomasz Bartyński |
| 0.13 | 27/02/2008 | Merged contributions from HLRS and UvA; status figure updated | Matthias Assel, Alfredo Tirado-Ramos, Tomasz Gubala |
| 0.14 | 27/02/2008 | Extending description of integration between GRR, GRR and monitoring system, updating articles section | Marek Kasztelnik |
| 0.15 | 27/02/2008 | Integration of GrAppO and monitoring system | Joanna Kocot |
| 0.16 | 28/02/2008 | Section on EMI integration with GSEngine Server (3.2) extended | Daniel Harezlak |
| 0.17 | 28/02/2008 | Minor corrections to 3.3, GrAppO in 2. | Joanna Kocot |
| 0.18 | 28/02/2008 | Provenance section (2.3). PROToS – QUaTRO integration (6.2). | Jakub Wach |
| 0.19 | 29/02/2008 | Provenance integration with monitoring infrastructure (section 6.1) | Michał Pelczar |
| 0.20 | 29/02/2008 | Expended section reporting integration of GSEngine with EPE | Włodzimierz Funika, Dariusz Król |
| 0.21 | 29/02/2008 | EMI – GSEngine use case text improvements | Daniel Harezlak |
| 0.22 | 29/02/2008 | Moved part of EPE-GSEngine integration to D2.3 | Eryk Ciepiela, Maciej Malawski |

| Version | Date | Changes Summary | Authors |
|---------|------|-----------------|---------|
| 0.23 | 29/02/2008 | Minor changes, collab status section added, references, bibliography | Tomasz Gubala |
| 0.24 | 29/02/2008 | Update of sections 2.3, 6.1 and 6.2. Provenance publications added to publication list in section 7 and to references. | Bartosz Balis |
| 0.25 | 03/03/2008 | Editing changes | Marian Bubak, Tomasz Gubala |
| 1.0 | 04/03/2008 | Final additions | Tomasz Gubala |

# Table of Contents

# List of Figures

# 1. Executive Summary

This document constitutes the description of the second prototype of ViroLab Virtual Laboratory software, developed after 24 months of the project duration and encompassing integration of presentation layer, collaboration services and the workflow provenance system. This prototype is based on the design defined 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/.

Here, a brief description of the overall architecture of the ViroLab Virtual Laboratory prototype is provided and the main focus of this deliverable is a presentation of the integration of VL components. As indicated, the prototype of the Virtual Laboratory incorporates specific functionality of its constituent components. This functionality enables it to be applied to executing real-life experiments from the virology domain.

The deliverable is structured as follows:

- Section 2 presents the current status of Virtual Laboratory development. 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 and detailing their current status.

- Section 3 discusses integration of the runtime system with middleware and presentation layers.

- Section 4 presents collaboration tools that have been integrated for use with the ViroLab Virtual Laboratory.

- Section 5 presents the Data Access Service and its integration with the ViroLab Portal.

- Section 6 presents the provenance tracking system and its interoperation with other components of the ViroLab Virtual Laboratory.

- Section 7 contains a list of publications relevant to Virtual Laboratory development.

- Section 8 contains closing remarks.

The demonstration of the capabilities of the integrated system is distributed throughout the document. Instead of having one, possibly confusing chapter on demonstration the state of integration is presented in each section (through an example use case within the system enabled by a certain integration point).

# 2. Current Status of the Virtual Laboratory

The ViroLab Virtual Laboratory [Bubak07, Gubala07] combines a set of tools and services that provide the HIV research groups with a collaborative workspace to perform virtual (*in-silico*) experiments, gather their results, share ideas among the members and, finally, help the medical doctors treat HIV-infected patients. In order to achieve this ambitious goal, the WP3 team performed a series of important steps that resulted in specific deliverable documents produced in the earlier stages:

- The state-of-the-art research to learn what solutions, methods, and technologies are already available that could help building the Virtual Laboratory infrastructure (see [D3.1]);

- The ViroLab Virtual Laboratory design that encloses the architecture and the future behavior of the system, its use cases, functionalities and implementation plans (see [D3.2]);

- After the first part of the realization process the work package delivered a first prototype of the core Virtual Laboratory components along with accompanying manuals and documentation (see [D3.3] and appendices).
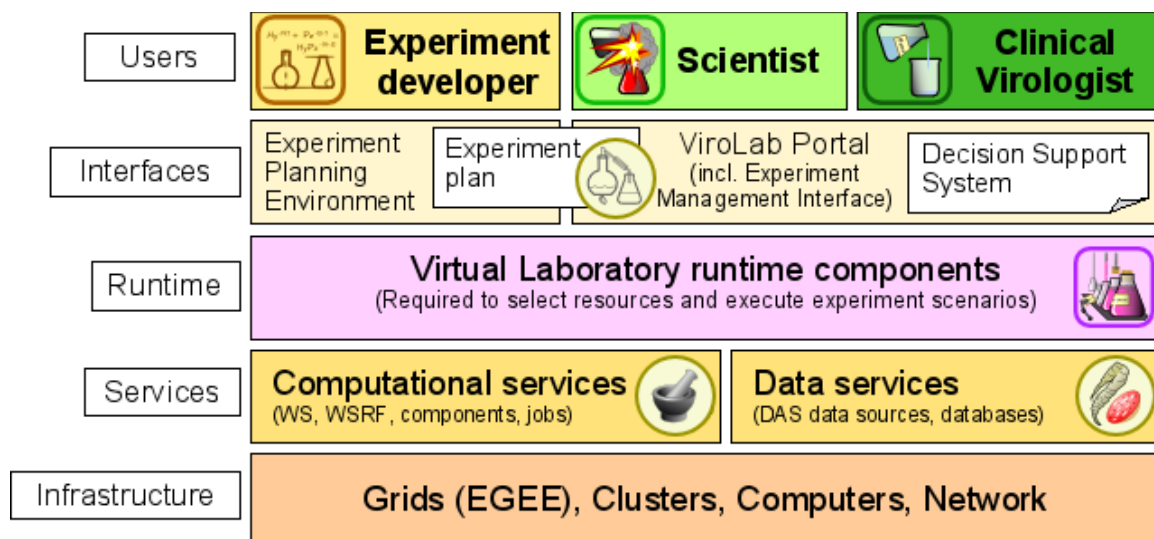


*Figure - 1: Different layers of the ViroLab Virtual Laboratory.*

To briefly introduce the idea of the ViroLab Virtual Laboratory (Figure -1) let us start with the conceptual view of the solution and the intended classes of users:

- Experiment developers, who combine their technical skills and the knowledge of the virus analysis domain to plan new types of virtual experiments taking place inside the Virtual Laboratory

- Scientists, who are the main recipients of the experiments built by developers and who use these experiments to extend their knowledge of the virology and epidemiology field

- Clinical virologists, who use the knowledge of (possibly many) virology experts, and dedicated decision support tools, to help the medical doctors in treating HIV-positive patients.

These users perform their respective tasks employing dedicated tools (that are grouped in the interfaces layer of Figure -1). Then, a generic runtime component is provided to be able to support (with software services and hardware machines) all the tasks performed by the users with their upper-layer tools. This runtime part serves as a connection to both other tools and to lower-layer resources dispersed in the virtual laboratory. Those are mainly (clinical) data sources and analysis computational services, although some specific task-oriented services are also available there (e.g. Experiment Repository, laboratory Result Data Store etc. – please consult [D3.3] for more detailed description of runtime services). Finally, these services run on physical equipment (the bottom layer).

This document reports on integration of the components of the Virtual Laboratory and gives the development progress report. All the changes and advancements in this document are given in relation to the state of the First Prototype that is described in [D3.3] and the authors assume the reader is familiar with this document. Below there is a summary of the advancements reached in the parts of the Virtual Laboratory that are developed within WP3 (see Figure -2) and later sections will present the progress and the state of integration in more detail.
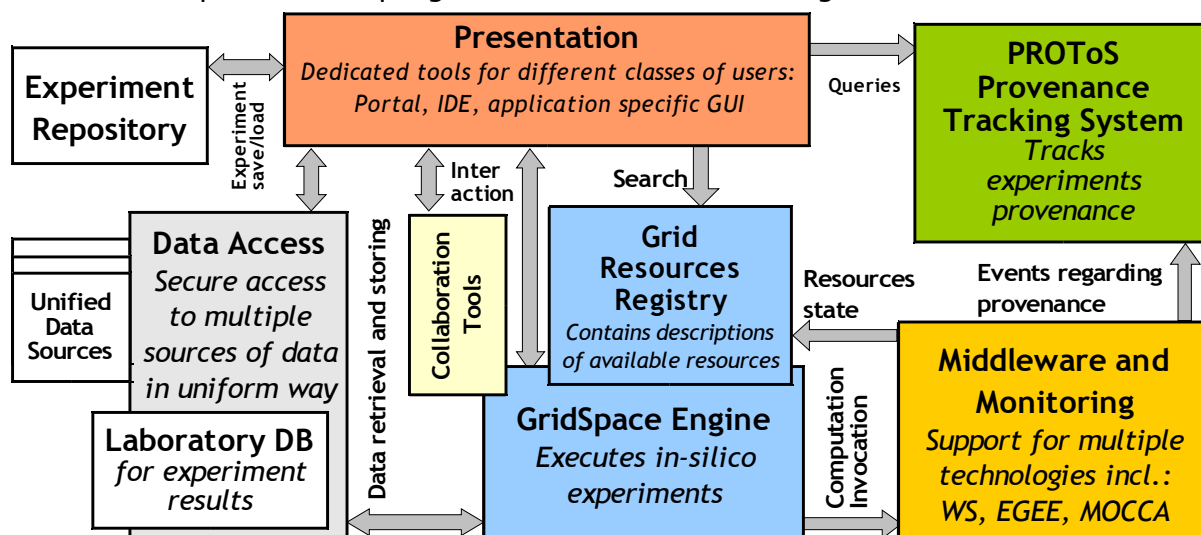
*Figure -2: Simplified architecture of the ViroLab Virtual Laboratory*

# 2.1. GridSpace Engine and Runtime Services

As thoroughly described in [D.3.3], GridSpace Engine [Ciepiela07] is considered as a facade service that provides access to virtual laboratory functionalities. Such a facade is intended to be employed by vast range of existing and foreseen dedicated tools e.g. by those contained in presentation layer. Most notably, it is an architectural unit responsible for running in-silico experiments as well as a proxy facade for Data Access Service. In order to fulfil these responsibilities GridSpace Engine directly engages several specialized architectural units such: Grid Operations Invoker, Data Access Client and Experiment Repository Client whose status is discussed in details in further paragraphs.

## 2.1.1.GridSpace Engine

Compared to status from Section 4.1 of [D.3.3] a significant progress took place that results in a number of new features, enhancements and corrections.

- **GSEngine API** – GSEngine provides simplified and compact yet powerful Java API that constitutes uniform access for GSEngine capabilities concealing underlying implementations e.g. local, embedded engine implementation or remote engine access though wire protocol.

    o **Application URI** – GSEngine API introduced the concept of Application URI that unifies identification of application. This way of application identification is easily extendible since it's plain hierarchical URI. It denotes where application is stored and how it is to be provided to the engine. E.g. prefix 'apprepo' denotes that the source code of the application can be found in the Application Repository e.g. virtual laboratory Experiement Repository. Prefix 'apprepo:svn' denotes that application is stored in the Application Repository implemented over Subversion (SVN) source code management server. Furthermore, the 'callback' prefix denotes that application code would be provided by caller tool through callbacks (discussed in below). Currently supported prefixes are: 'apprepo:svn' and 'callback'.

    o **Evaluation Callback** – application while calling GSEngine API has to have provided implementation of Evaluation Callback interface, that enables interaction between engine and caller application. It is employed for e.g. user data inputs, accessing locally stored files or, optionally, to provide application source code.

- **Ruby Interpreter** – GSEngine has embedded Ruby interpreter, namely JRuby, properly customized to fit to GSEngine.

    o **JRuby 1.1** – Currently GSEngine uses the newest version of JRuby – 1.1-RC1 that surpasses its predecessor with enhancements such as mechanism for easily pluggable gems and Java libraries or unification of Ruby loadpath and Java classpath that improves interoperability between Ruby and Java.

    o **Source code resolvers** – JRuby is extended with several source code resolvers that are able to upload application source code e.g. from Application Repository through Application Repository Client, or to ask caller application for application source code through Evaluation Callback interface.

    o **Injection of application context objects** – Applications interpreted within JRuby interpreter are provided with application context information through injection of global constant objects. It allows for instance accessing functionalities of GSEngine such as interactive user data input.

- **Runtime libraries** – Ruby interpreter of GSEngine comes along with a set of runtime libraries that enables to access functionality of virtual laboratory from the level of experiment code. New libraries are easily addable by simply placing them in the classpath of GSEngine. Aside the most notable and robust virtual laboratory runtime libraries such as Data Access Client or Grid Operations Invoker (discussed further) some additional libraries have been and are being provided.

  - **Data input library** – A dedicated library is provided to access files stored locally on caller application file system, and to interact with caller application in order to get some user data through e.g. forms.

  - **Domain ontology events publisher** – A dedicated library for publishing domain-related events e.g. reporting new drug ranking issued, that can be used from the level of source code of application.

- **Data Access Facade** – Apart from its interpreter GSEngine proxies Data Access Server so that data sources can be accessed not only from the level of code of application through Data Access Client runtime library but also by arbitrary tools through GSEngine API.

- **Monitoring ability –** Application launched within GSEngine interpreter are monitorable via events reporting. Engine reports events corresponding to application start, application end, grid operation invocation start and grid operation invocation end.

  - **Application Correlation Identifier (ACID)** – Events are correlated owing to unique identification of application run. Each run is assigned with unique ACID that supplements each event reported by GSEngine.

- **Integration with Shibboleth security infrastructure** – GSEngine can operate in Shibboleth security infrastructure since it accepts application evaluation requests that contain credential of 'user handle' which is afterwards stored in application context and passed to security-enabled Data Access Client, Grid Operations Invoker and Application Repository Client.

  - **Authorized Access** – GSEngine is secured from access by unauthorized parties, namely, parties not belonging to any trusted Shibboleth Home Organization.

- **Remote Access** – GSEngine API has currently two implementations: one is the GSEngine embeddable in the same Java Virtual Machine as caller application, and the second one is accessing remotely staged GSEngine Server. The latter one involves several items.

  - **GSEngine Protocol** – a protocol was developed that enables interaction between caller application and GSEngine Server over a wire. It defines messages exchanged between the client and the server.

- o **CompTalks** – a novel simple framework for specifying interaction between communicating peers, over which variety protocols may be defined. It enables pluggable transport layers. The default, built-in transport employs TCP. GSEngine Protocol was build upon this framework.

- o **GSEngine Server** – GSEngine core wrapped as CompTalks peer able to interact through GSEngine Protocol over the wire.

- o **GSEngine Client** – a library that implements GSEngine API that is able to interact wirth GSEngine Server through GSEngine Protocol over the wire.

- **Command line tools** – GSEnigne is itself an API and a library that comes along with the simple command line tools that simply foster the engine to be launched via command line:

  - o **gsel** – a tool to run **GSE**ngine **l**ocally only for a one run of given application.

  - o **gses** – a tool to run **GSE**ngine **S**erver that embeds GSEngine and accepts application evaluation requests.

  - o **gsec** – a tool to run **GSE**ngine **C**lient that can interact with remote GSEngine Server.

  - o **Integration with Shibboleth IDP Client** – both 'gsel' and 'gsec' command line tools are integrated with developed Shibboleth IDP Client that can authenticate user and acquire Shibboleth handle for user's Shibboleth Home Organization.

- **Security mechanisms in computation access** - The GSEngine creates a global variable that holds the Shibboleth handle. The Grid Operation Invoker (described later) is able to invoke Grid Operations on "Shibboleth enabled" Web Services. Such services require a client to authorize by providing the handle passed by the GSEngine handle. If a computational resource is indicated as being secured, the GOI transparently includes Shibboleth handle in the operation invocation.

The future work on GSEngine will be carried out in several directions:

- To make GSEngine Server full-featured container providing isolation and secure sandbox for applications' runs.
- To secured transport layer in GSEngine Server and GSEngine Client communication.
- To design and develop a generic user interface libraries to render application UI on the caller application-side

## 2.1.2. Data Access Client (DAC)

The Data Access Client provides experiment developers with access to remote data sources. The DAC is a part of the GridSpace engine devoted to communicating with the underlying data sources, which comprise the ViroLab

architecture. Specifically, the Data Access Client currently enables access to the following:

- **Standalone databases** – any SQL-compliant database can be directly referenced from within the experiment script. It is only necessary to supply the URI at which the DBMS is located and the name of the database schema in question. Communication is enabled via JDBC, through JRuby.

- **Data Access Service** – the Data Access Service overlays a number of data sources which represent the hospitals participating in the ViroLab project. It enables secure and uniformized access to hospital laboratory data, using the REGA schema. As the DAS is exposed as a WSRF service, the DAC takes care of communicating with this service and formulating the proper queries. The experiment developer is provided with a simple API which enables submission of queries using a JRuby data access object. Security is supported, following integration of the Shibboleth authentication mechanism. Moreover, the Data Access Client provides backend implementation for the Data Access Facade, described below.

## 2.1.3. Experiment Repository (Application Repository Client)

Application Repository Client is a part of GridSpace Engine responsible for accessing Experiment Repositories where experiment plans are stored and versioned. A sample Experiment Repository based on Subversion (SVN) is hosted by HLRS at and can be accessed through https://svn.gforge.hlrs.de/svn/virolab/trunk/experiments/. The Application Repository Client is used by other ViroLab Virtual Laboratory components whenever they need to access such a repository.

On the current stage of development, the Application Repository Client offers the following functionality:

- API for Experiment Planning Environment (the integration with EPE is more widely discussed in [D2.3]) – including: sharing a new experiment plan, checkout of an existing experiment plan, releasing a new version of an experiment plan;

- API for Experiment Management Interface (the integration with EMI is more widely discussed in [D2.3]) – including: listing released experiment plans and their versions, downloading an experiment plan for execution, its license file and descriptor, downloading and uploading the experiment's feedback file;

- API for the GridSpace Engine experiment interpreter – downloading the experiment plan for execution;

- Enables Shibboleth authentication (please consult [D2.3]) – a test repository is available at https://virolab.gridwisetech.pl/shibsvn/testrepo/;

- Implementation for Experiment Repositories based on Subversion (SVN) – with use of SVNKit [SVNKIT] libraries.

The future work on the component will include new methods facilitating access to Experiment Repositories and the Application Repository Client configuration.

## 2.1.4.Grid Operation Invoker (GOI)

Grid Operation Invoker is a module of the GridSpace Engine that provides a uniform interface to computational resources within an experiment. It implements the Grid Object abstraction [VLINV] thus facilitates development and execution of complex experiments accessing functionality provided by various remote resources. These resources may use Web Service, WSRF (prototype support), MOCCA, WTS and LCG (EGEE) middleware technologies. The resource to be used by an experiment is selected by the GrAppO component at runtime, its description is retrieved from the GRR and it is accessed in its specific protocol.

Since the 0.2 version of the ViroLab virtual laboratory, the GOI has been enhanced with the following features:

- prototype support for the WSRF middleware,
- prototype support for asynchronous invocations of operations,
- reporting invocations of Grid Operations (for provenance purposes),
- ability to invoke "Shibboleth enabled" Web Services.

The future work on this component will involve creating introspection into Grid Object representatives, implementation of an adapter for the AHE middleware as well as enhancing adapters for WSRF and EGEE.

## 2.1.5.GridSpace Application Optimizer (GrAppO)

The GridSpace Application Optimizer [Malawski07] component is responsible for choosing a suitable service which will be used to perform Grid Operations by the Grid Operation Invoker – GOI. To facilitate this choice, GrAppO should use the Grid Resource Registry, monitoring system and provenance system.

The following features have been added since the 0.2 version of the ViroLab Virtual Laboratory:

- Passing the GRR location as an argument from GOI – enhanced configuration
- Prepared integration with monitoring system (see section 3.3)

The future work on the GridSpace Application Optimizer will include performing further integration with monitoring system and starting integration with the provenance system.

## 2.1.6.Grid Resources Registry

Grid Resources Registry is a component where information about accessible computational resources is stored. Thanks to integration with EPE, Ontology Browser and GSEngine it hides the complexity of used technology from the user and allows to focus on functionalities provided by the resources instead of the way of invoking them (for more information see [D3.2] and [D3.3]).

Version 0.2 of the Grid Resources Registry, which was described in [D3.3], has been improved by adding the following functionalities:

- Grid Resources Registry core:
  - o Support for new technology: WSRF
  - o Integration with monitoring and provenance system
  - o mechanisms that allow to manage information stored in the registry
- Grid Resources Registry EPE plug-ins:
  - o Admin plug-in (allows to add/edit/remove Grid Objects, Grid Object Implementations and Grid Object Instances).
  - o Integration of Admin plug-in with Resources Browser plug-in (for more information about admin plug-in see tutorial available on [VIROLAB-VL])

The following enhancements are planned for the last year of the project:

- Resources Registry core:
  - o Further integration with monitoring and provenance systems
  - o Better support for MOCCA components (by providing a list of known H2O kernels where components can be deployed)
- Grid Resources Registry EPE plug-ins:
  - o Support for ctr+space action in the script editor
  - o Allow to create Grid Object, Grid Object Implementations and Instances from WSDL web services description.

## 2.2. Data Access

Most of the Data Access Service interfaces haven't change since the previous release. Nevertheless, some of the existing functionalities have been extended and/or re-implemented in order to enhance the reliability and scalability [Assel07] of specific parts and to fix some minor bugs. Additionally, new service capabilities have been developed and introduced guaranteeing a certain level of security and trustworthiness [Assel06] for both service users and (data) resource owners. The access to the service is now created and granted on-demand based on the current user's context. This means that each requestor who has been initially authenticated and successfully authorized is exactly matched with one unique instance of the service, which is accurately and only reserved for the respective user. Furthermore, to facilitate the interaction with the services, some of the annoying features like the separate instantiation of each service resource have also been passed from the user's site towards the internal service responsibilities. Finally, the authorization interfaces have been adapted to Shibboleth's authentication and attribute exchange framework (as introduced by WP2) to allow Single Sign On and attribute-based access control [D2.3] for all users of ViroLab's virtual laboratory.

Future developments planned for Data Access shall continually improve the reliability and scalability of the individual services' capabilities as well as increase the data submission performance by processing queries in parallel instead of submitting requests one after another. The basic user authorization shall be handled by an external component – a policy decision point [Kipp07] – which has

been already tested but still needs to be integrated with the corresponding service interfaces. The management of appropriate access control policies shall be supported through a nice and user-friendly graphical user interface allowing the fast, easy and dynamic generation, change, and upload of particularly designed policies in order to provide more flexibility in administering distributed data resources within a collaborative working environment such as ViroLab.

## 2.3. Provenance

Since the previous release, most basic PROToS functionality has been in place. Thus, all changes applied thereafter concentrated on technical details. Firstly, we have built a number of tools to quickly build and deploy the PROToS components. Because PROToS architecture is inherently distributed, this was crucial for swift integration and testing. Secondly, all of system components have been prepared for one-file distribution. This enabled us to deploy the PROToS on the main CYFRONET's ViroLab server where all ViroLab virtual laboratory components are integrated. Much work has also been dedicated to unit testing of PROToS code. Our system is fairly complicated (both in terms of code lines and algorithms implemented), so the achievement of a successful, bug-free processing of data from Virtual Laboratory monitoring system was a big challenge.

In parallel, the provenance team has also been developing ontologies used for semantic modeling of experiments, data and specific applications. Since our last report, we have updated our generic experiment and data ontologies. We have also created additional application-specific models: for Drug Ranking System and genotype-to-DRS scenarios, which also use REGA tools. These lie on the base of integration with application running in our GridSpace Engine.
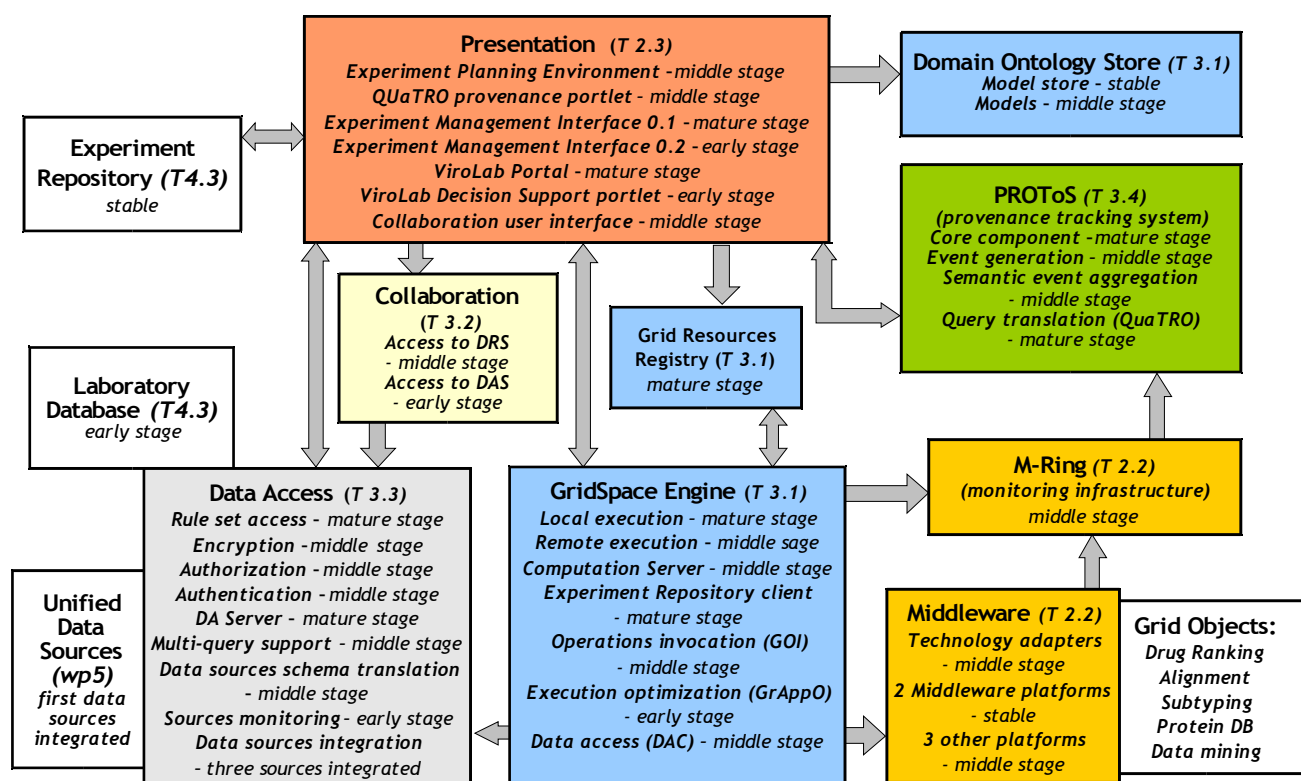
## 2.4. Development status summary

*Figure -3: Current status of the development*

Figure -3 shows an overview of the current state of development of individual packages and the level of integration with their neighboring modules (please compare it with a similar figure in [D3.3]). The development team already reached the phase when the development of all and each of the components of the Virtual Laboratory is started and ongoing. This is the result of the significant shift of the work effort towards the implementation in the second year of the Project. The core components that were started considerably early (with first, technical-preview releases in April 2007) have reached more stable state recently, while a set of newly started components is still in an early, quite unstable phase. Also, some parts of the user interface deployed in Portal are being re-implemented now (due to a decision to drop the previous technological solution) so there are two versions mentioned (this however has no impact on the users of the Virtual Laboratory since the older, more stable version is still supported by the development team). The arrows in Figure -3 show which communication channels are already implemented and used in the latest releases of the ViroLab software.

Since the release of the first prototype the implementation team employed the incremental, open-source based process of development. Each new release is made available for a download for the community (along with its updated documentation) and there is a reporting mechanism where users may report problems or request for new features. The requests are then aligned with the future release plans so the users may see how quickly some issue will be solved. This process is backed by the Virtual Laboratory project website: [VIROLAB-VL].
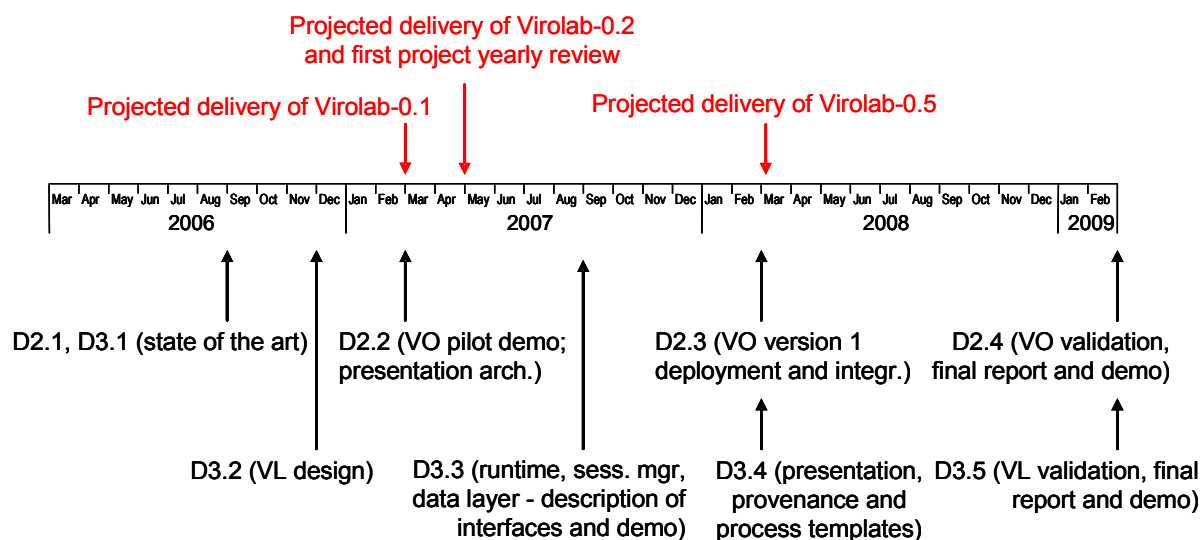
*Figure - 4: Virtual laboratory implementation timeline (from D3.2).*

Figure -4 recalls the development timeline presented in [D3.2] and maintained since then. The implementation process is now approaching the *"Virolab-0.5"* milestone that constitutes a moment in time when all the crucial components of the Virtual Laboratory are finally integrated and start to operate together. The detailed sections that follow explain in much more detail how this works in the latest ViroLab software releases.

# 3. Integration of Runtime System with Middleware and Presentation Layer

## 3.1. Integration with Experiment Planning Environment

As reported in Section 2.1 GSEngine provides Command Line Tools such as 'gsel' and 'gsec' that enables running applications locally or remotely, respectively. These two are accepting several command line arguments such as application URI, main script file name and input arguments, and in the case of gsec additional two arguments, namely, server address and port number of GSEngine Server. These tools may be run both from console and from language-specific libraries for launching external native processes of operational system (OS) e.g. through Java's 'java.lang' package. Having launched external process application can handle input, output and error streams of such a process.
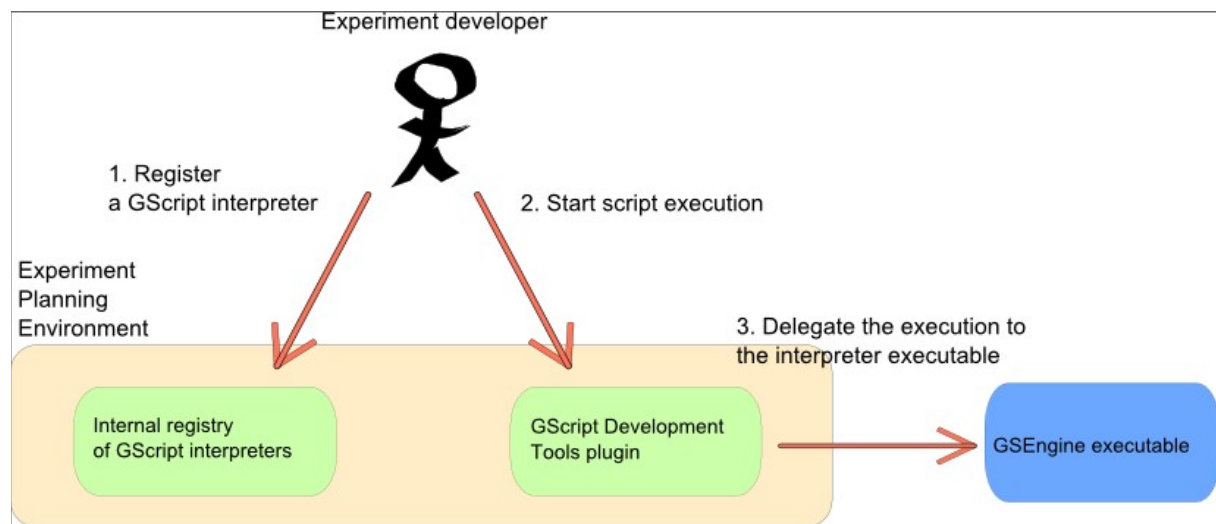


*Figure -5: Procedure of registering GScript interpreter in the EPE*

From the user point of view, the only thing to be done (besides installing the GSEngine correctly) is providing in the EPE information about GSEngine executables location (Figure -5). After this, the developer can execute experiment with the "Run as experiment" operation (for more details see the 4.1.2 section in [D2.3]).

## 3.2. Integration with Experiment Management Interface

As reported in Section 2.1.1 GSEngine provides Java libraries containing GSEngine API and its two implementations that enable evaluation of application both within embedded (in the same Java Virtual Machine) engine or within remotely staged GSEngine Server accessed through GSEngine Protocol.

EMI includes these libraries and directly calls a local facade of either a remote or local implementation of the actual GSEngine server. The deployment is divided into three layers which are a client's web browser, web application server and a remote engine server. Figure -6 presents the integration points between those layers (marked with horizontal red lines).
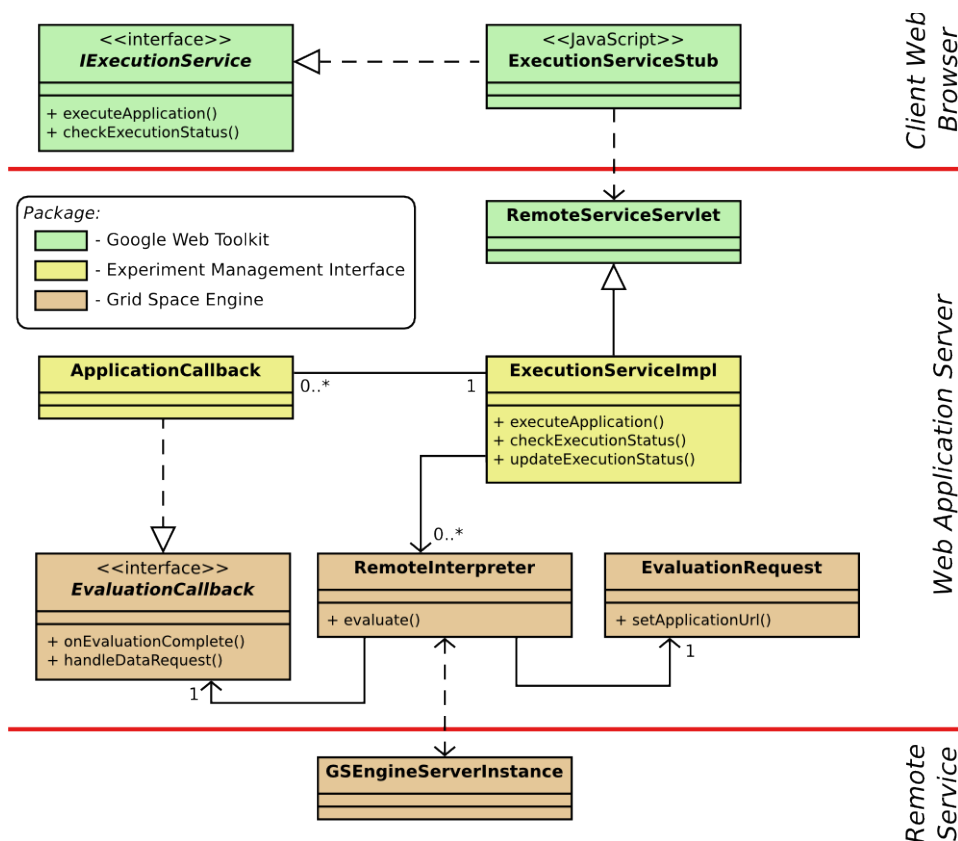
*Figure - 6: Integration points between EMI and Grid Space Engine*

Both integration points are implemented asynchronously. The one between client's web browser and web application server using an RPC mechanism of the google web toolkit [GWT] and the second one using a TCP implementation called Grid Space Engine Protocol (GSEP). GSEP provides a client used in the web application server to implement the communication with the server (marked in brown in Figure -6). On the Experiment Management Interface side (EMI) which is marked with yellow an implementation of the *EvaluationCallback* interface is provided (*ApplicationCallback*) in order for the GSEngine server to asynchronously notify the web application server about the status of application evaluation. In addition EMI uses *RemoteInterpreter* and *EvaluationRequest* classes to request evaluations of specific applications according to the user actions. For each evaluation new instances of those classes are created and separate logical communication channels with GSEngine server are established. This ensures independent application handling by EMI. The main synchronization point is implemented by the *ExecutionServiceImpl* which is thread-safe. From the side of the browser client a standard servlet implementation is used. The base servlet implementation is provided by the GWT package which is extended by the execution service implementation providing the implementation of the business login methods like *executeApplication()* or *checkExecutionStatus()*. The *IExecutionService* interface exposes those methods on the client side. The marshaling and unmarshaling of the parameters between Java and JavaScript implementations is handled by the GWT toolkit.
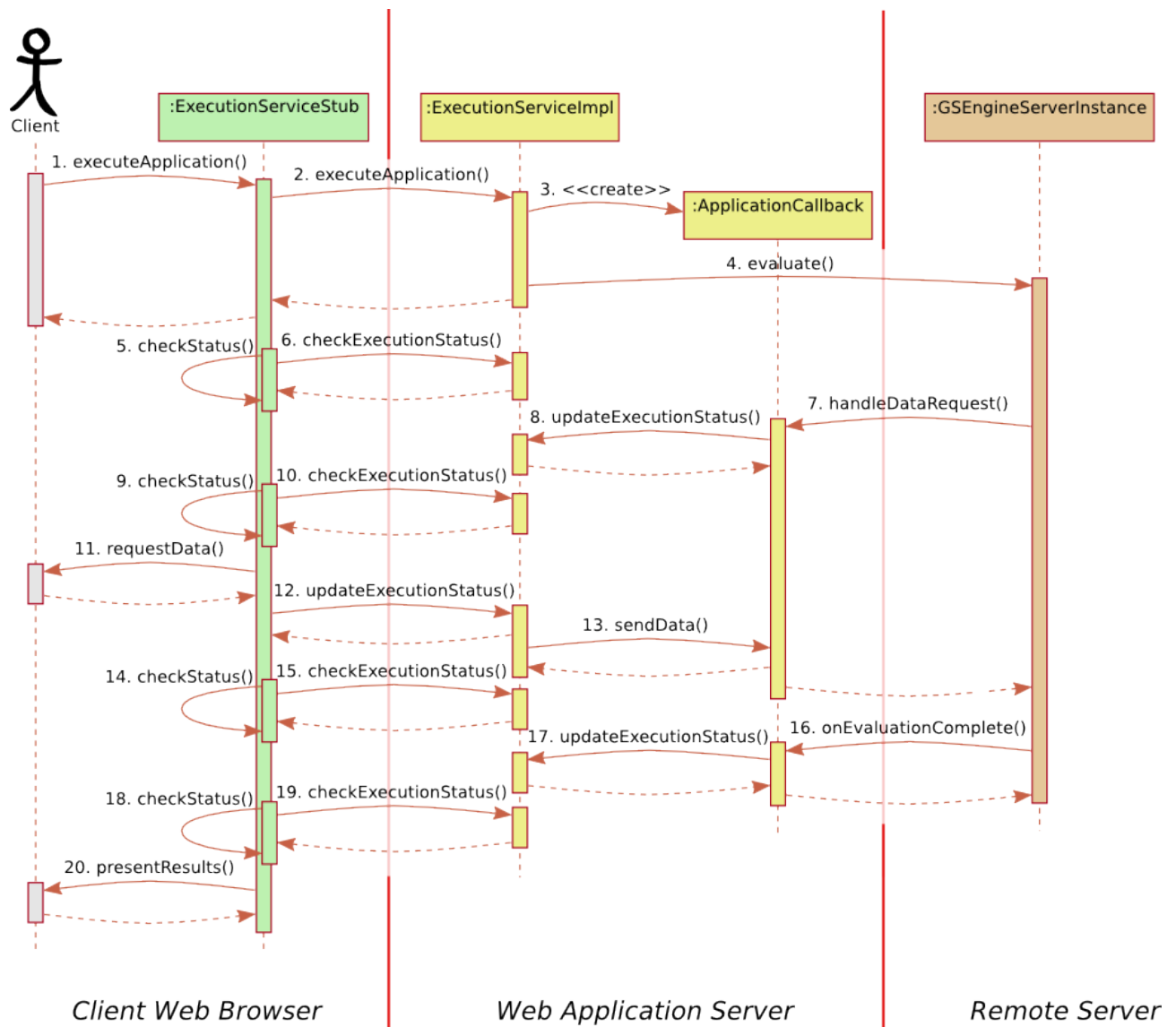
*Figure -7: Generic interaction diagram*

In Figure -7 a generic example of the interaction between a client and a GSEngine server with EMI as an intermediary is presented. This allows a user to execute experiment plans with introspection at the status of the execution. There are three states in which the evaluation may be reported back to the user. These are running, data required and finished states. The described communication model also implements the use case of submitting user data to the GSEngine server when required by the experiment. The diagram in Figure -7 shows that the client is always notified about the changes in the evaluation of the application requested in step 1. To keep accordance with Figure -6 also three deployment layers are depicted. For each evaluation a new *ApplicationCallback* is created in step 3, followed by an asynchronous *evaluate()* call to the GSEngine server instance. In steps 7 and 16 the server requests data from the user and notifies about the evaluation finish accordingly. The state in the execution service implementation is updated in steps 8 and 17, which is done independently of the *checkStatus()* method repeatedly invoked by the browser client. After the new status is passed to the browser, proper views are updated and the client receives new information. It is important to stress that simultaneous execution of applications is supported. Data requests are limited to one single-lined text

strings for each request. During one evaluation several data requests may be performed.

The future integration should extend the data request mechanism to support more complex data definitions. Another improvement will be a complete reimplementation of the way results are presented. This modification will have to include changes in the API delivered to application developers to select result data. A new storage facility with metadata support will be provided to effectively annotate the results. In addition, dedicated viewers on the client browser will be provided to customize the way different data formats of the results are presented.

# 3.3. Integration with Monitoring Infrastructure through Events Propagation

One of the Grid Resources Registry functions is storing the endpoint addresses to every service which is able to be used in the system. The monitoring provides information whether this service is available for use. Integration of this system is based on the Java Message Service (JMS) with two channels (see Figure -8).
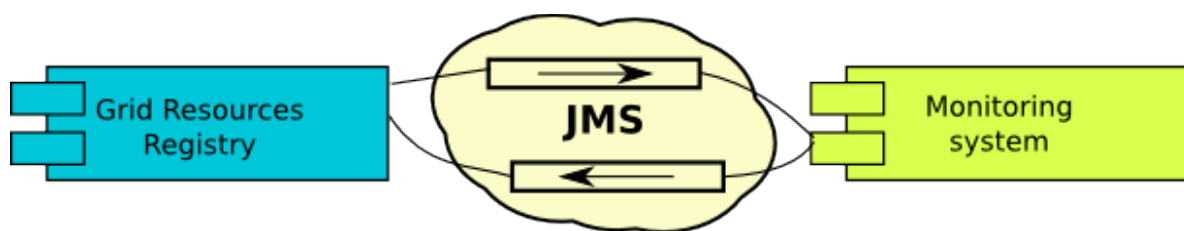


*Figure -8: Communication between Grid Resources Registry and monitoring system.*

The first channel is used by GRR to send events with the endpoint of service to be monitored or events that resource should be stopped being monitored. The second channel is used by monitoring to send the status of the monitoring service to the GRR. The events with information about the availability of service are created periodically with preconfigured time gap. The status of the service passed in these events can be: ALIVE, DEAD or NOT_CHECKED. Thanks to this information GrAppO receives only the information about resources that are accessible and operable.

Integration between GRR and monitoring infrastructure is fully configurable thanks to the registry architecture. As was described in [D3.3] Grid Resources Registry is created using IoC [IOC] design pattern implemented by Spring framework [SPRING] that allows to configure outside the application code what should be invoked at the startup or shutdown of the application. Additionally thanks to support for aspect oriented programming, without hard coding we can configure the method execution chain. The integration between GRR and monitoring system looks as follow (see Figure -9):

- When application starts it registers in the monitoring system all resources instances that should be monitored. If during the runtime new resource is registered, proper event is generated and added to the list of the resources that are monitored.

- Events are generated to the monitoring system to stop monitor concrete resources before registry shutdown. Similar to the previous point if resource is removed from the registry during the runtime, an event is generated to the monitoring system and resource is removed from the list of monitored resources.
- During the runtime monitoring system sends periodically information about resources' status to the registry which updates their status.
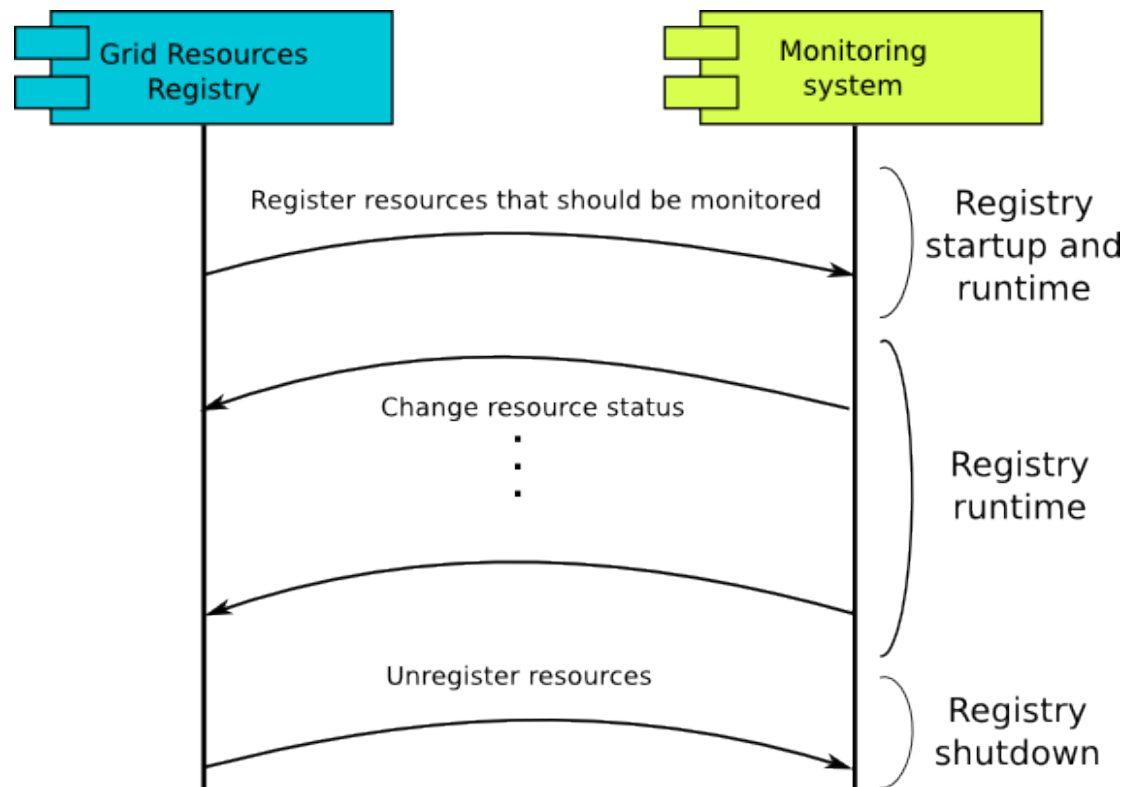


*Figure - 9: Exchange of events between Grid Resources Registry and monitoring system*

In addition to the information about service status (see the previous paragraphs), the monitoring system will pass data concerning capabilities of the resource (node) on which the service is hosted to Grid Resources Registry. This information is node's RAM and CPU, later consumed by GridSpace Application Optimizer (GrAppO). Such data can be stored in the registry since they are relatively static and do not change unless the service status is altered (what is reported to GRR by the monitoring service).

GrAppO needs also information about current (at runtime) resource utilization. This include RAM and CPU usage, which are obtained directly from the monitoring system. The data flow between GridSpace Application Optimizer and Grid Resources Registry and monitoring system is illustrated by Figure -10.
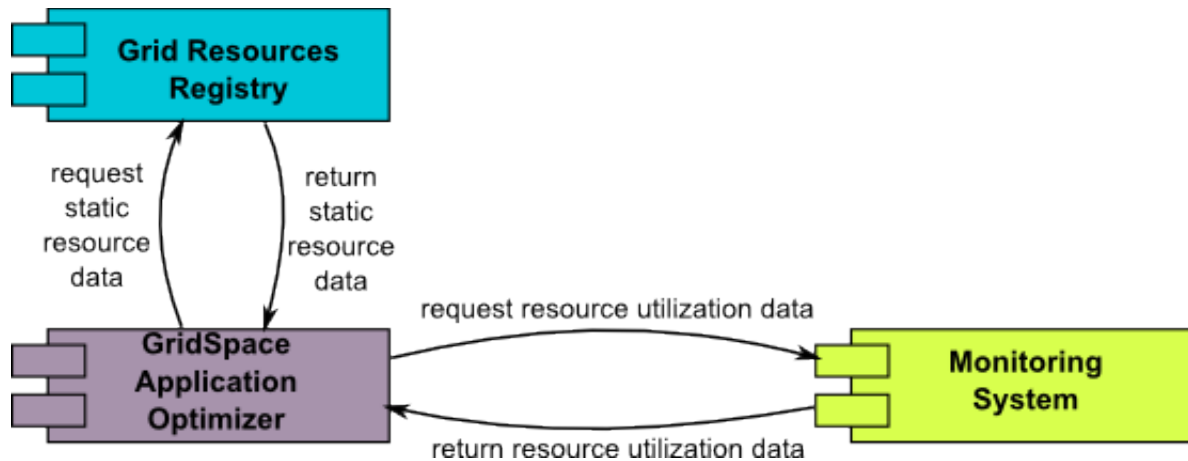
*Figure - 10: Data exchanged between GrAppO and its peers (GRR, monitoring) about resources condition*

GrAppO connects to GRR through a web service interface provided by the registry. The messages are sent using SOAP (see Figure -11).

The integration with monitoring system will be realized, as in case of GRR, using Java Message Service (JMS) with two channels (see Figure -11). The first channel is responsible for carrying the GrAppO's request and the second one carries the response of monitoring system.
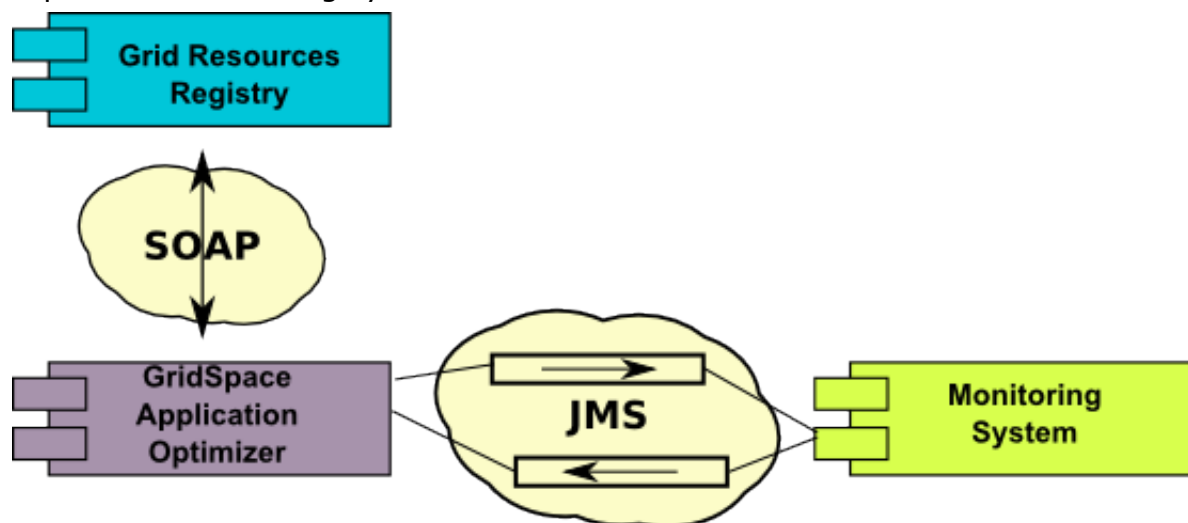


*Figure - 11: Communication protocols between GrAppO and its sources of information*

Current status of the integration enables monitoring of the resources availability and operability. In the future, further integration between Grid Resources Registry, GridSpace Application Optimizer and the monitoring system is planned. When the additional information is gathered by the monitoring system it will be passed to GRR and, further, to GrAppO.

# 4. Integration of Collaboration Tools with Portal, DAS and DRS

The Patient Treatment Support (PTS) tool, initially implemented based on standard mutation list alignment functionality, allows users to define and fine-tune their real collaboration needs, such as exchange of aligned sequence data with their colleagues, seamless access to distributed hospital data, access to consolidated sequence ranking services, and exchange of ranking results and conclusions with other scientists. The integration of the core tool functionalities within the ViroLab Virtual Laboratory are focused on the porting of the medical user interface into the ViroLab Application portal, and its enhancement with collaboration capabilities.

For access to the distributed data providers for communication of data and results via integration with the Virtual Laboratory, the main part of the integration with the postal relies on a few back end services provided by the Virtual Laboratory:

- The Data Access Service backend, which provides libraries for on-the-fly connection to data providers.
- The Grid Object Interface, which provides access to the distributed Decision Support Ranking System.

For instance, we integrate the tool into the Data Access Service backend using the service's org_hlrs_das_stubs.jar and org_hlrs_das.jar. Collaboration within the Virtual Laboratory is provided as a service providing an interface to the GridSphere portal.

ViroLab hospital data repositories provide an interface via the Data Access Service (DAS), using on-the-fly service instances. An interactive Grid service provides connectivity to the meta Decision Support Ranking Service (DRS) which connects to a number of HIV drug-resistance ranking algorithms within specific regions of the genome. The ranking results can be saved and exchanged with colleagues for further discussion and experimentation.

In a typical use case, the tool provides medical users a friendly interface where:

1. A medical user, e.g., a virologist, logs into the ViroLab application portal in order to query drug resistance rankings (https://virolab.gridwisetech.pl/gridsphere/gridsphere), using her Grid credentials or Shibboleth proxy.

2. The user clicks on the Patient Treatment Support tab, where she may:

   o Select data repositories (e.g., ViroLab hospital/institute partners) where input data for new queries can be found.

   o Select one or more HIV algorithms to use (e.g., RetroGram REGA, Stanford, ANRS).

   o Select the regions of the Genome of interest (e.g., Protease, Reverse Transcriptase, Envelope).

   o Enter/paste a mutation or sequence to query, or upload/paste one or various FASTA files.

   o Submit the query to the distributed ranking system.

3. Once the ranking has been performed and the results are shown, the user may:

- o Save the resulting rankings locally and/or e-mail the results, together with comments and conclusions, to other colleagues.
- o Access other colleagues' results and comments for comparison and new experiment runs.

We expect next to work on improving the implementation of the tool's user interface (e.g., the support for multiple FASTA files, automatic alignment of sequences, improving of the results' interface for presentation and sharing), and continue full integration activities within the ViroLab-specific GridSphere portal framework, Data Access and Provenance services.

# 5. Data Access Service

## 5.1. Integration with ViroLab Portal

Querying data from widely dispersed resources constitutes a challenging and difficult task. The connection to all available resources needs to be highly secured and, at the same time, the different underlying technologies need to be carefully integrated. However, apart from the technological circumstances, the complex task of designing and implementing a nice and user-friendly front-end requires the same effort and time rather than building the transparent infrastructure.

To overcome this challenge, a specifically styled data access GUI (DASPortlet, Figure -12) is being developed and foreseen to be integrated with the ViroLab Portal. A first screenshot of this application is shown in the following figure.
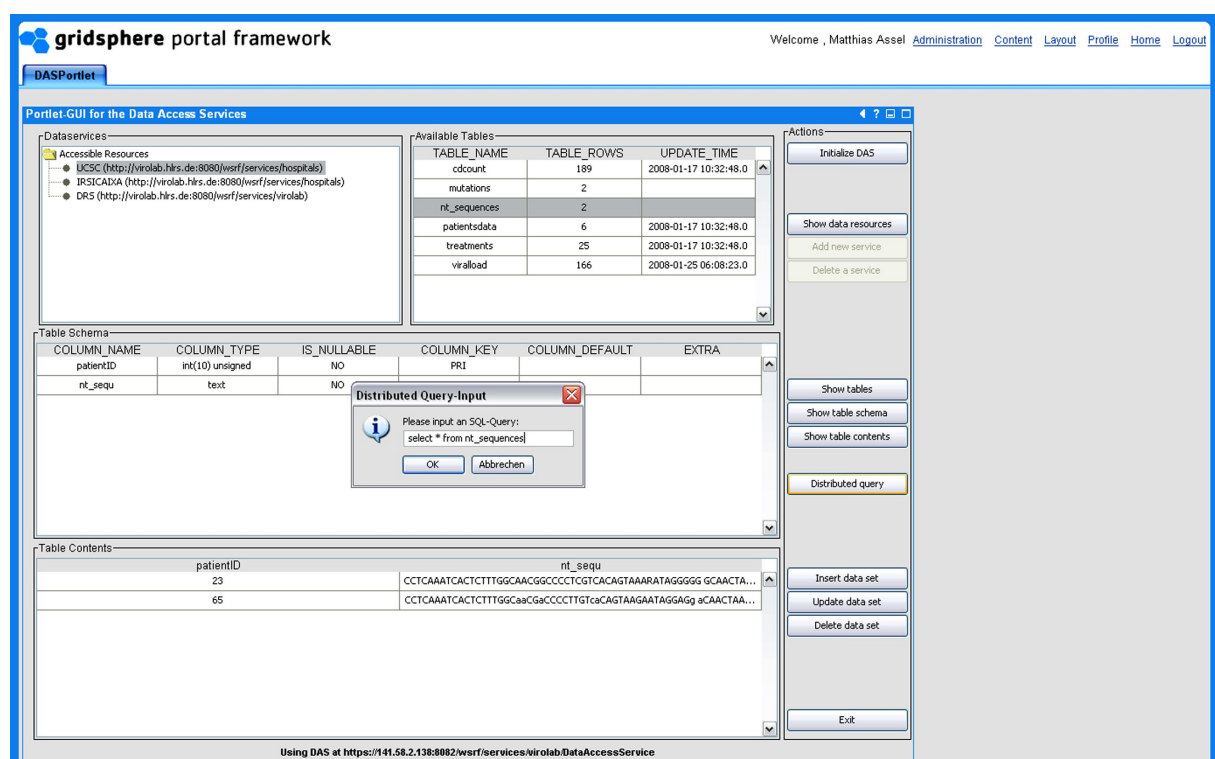


*Figure -12: Screenshot of the DASPortlet*

This preliminary version of the portlet currently running at HLRS's portal instance connects with the DAS in a very common way by simply making use of the well-known web service principles – the services' capabilities are accessible via so-called web service proxies that hide all communication and transformation operations from the users. The interactions between the application and the interfaces are based on SOAP over HTTP or additionally, via HTTPS to increase the message flow security.

All basic functionalities are directly executable from the portlet. That includes the simple viewing of available and/or accessible resources, the individual browsing of certain databases but also the submission of distributed queries to respective resources. Particular security features are described in more detail within deliverable [D2.3].

To explain the extensive interplay of the major DAS subsystems (data and security) together with the developed user interface, we briefly demonstrate a common use case, how one can submit a distributed query to all available data resources concurrently and, explain which actions are carried out during the user's request.

Typically, doctors want to use the virtual environment for requesting patient information including genetic data such as nucleotide sequences or mutations, in order to predict any possible drug resistance for their according case. They simply want to retrieve all relevant information without the need of any specific expertise in computer science. Therefore, the way to get to the data must be kept simple and transparent to them but should be as self-explaining as possible.

Figure -13 clearly depicts the aforementioned scenario separated into the four major infrastructure layers of the virtual laboratory. In the following, the individual steps shall be roughly mentioned.
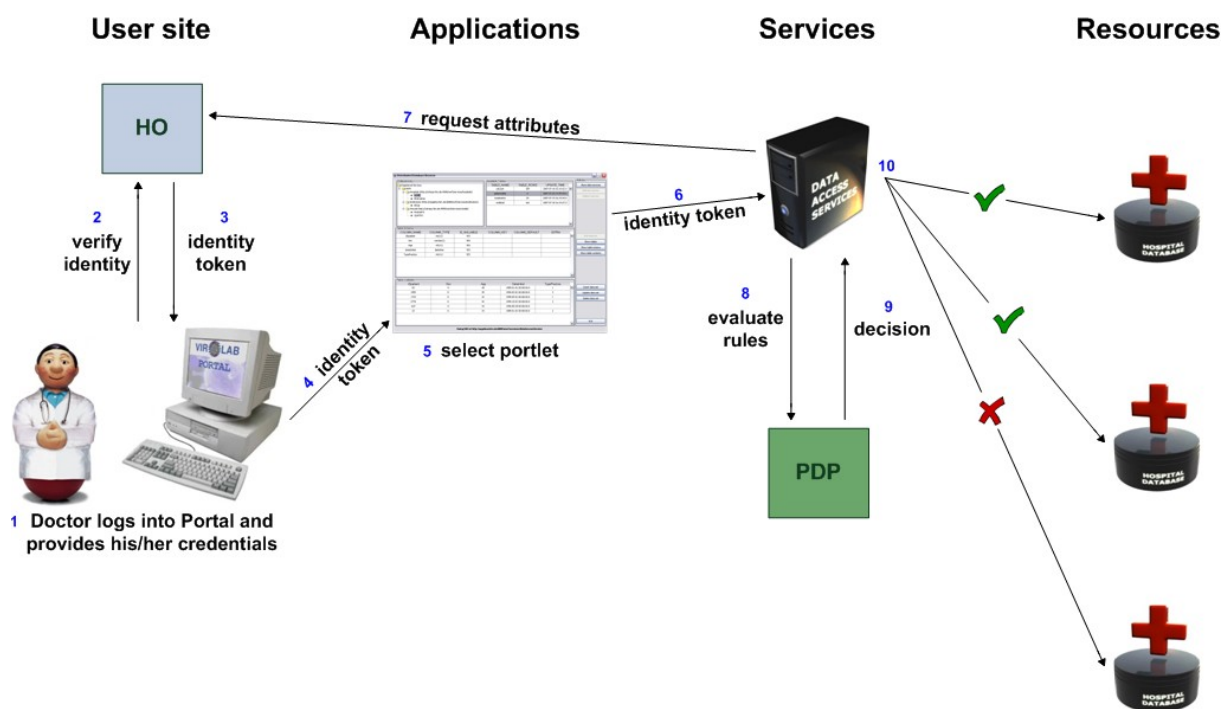


*Figure - 13: Common ViroLab data request*

Steps:

- User (doctor) logs into the Portal. He/she needs to provide his/her credentials (typically username and password).

- The credentials are now transferred to the local identity management system located the corresponding user site (HO). They are verified against the local user credentials stored in the hospital's database.

- In case the user is known to the local system, a digital identity token (fingerprint) is created and sent back to the server that is hosting the portal. If the user's identity is unknown, the portal displays an error message and the process is immediately stopped.

- Once the user is logged in, he/she can choose between different available applications.

- In our case, the user wants to query some data from the integrated data resources. He selects the DASPortlet and sends a request to the DAS.

- During that request, the initially created identity token is also passed to the DAS that takes the token to request the corresponding user attributes like the user's role, organization, department, e-mail address etc.

- This attribute request is performed by sending the identity token to the user's home organization (HO) firstly proving whether the user is known and the token is valid. In a second step, the released attributes are obtained from the local database and returned to the DAS.

- The DAS takes all these attributes to authorize a user for certain resources. The final decision whether someone is allowed to access a resource is only known to the PDP (Policy Decision Point). Therefore, DAS sends an authorization request to the PDP. This request contains the afore-requested attributes and a list of available resources.

- The PDP checks its stored access control policies for corresponding rules. These rules contain conditions specifying the required set of attributes. If a policy rule matches with the provided attributes, the appropriate resource is cached. Having evaluated each access policy, the PDP return a list of accessible resources to the DAS.

- Finally, the DAS takes the incoming query and tries to connect to each of the accessible resources and performs the request.

To present a more technical view on the abovementioned workflow, Figure -14 displays the same scenario in form of a typical use case diagram showing all relevant message and data flows during the submission of a distributed query. In particularly, the involved DAS components are highlighted here.
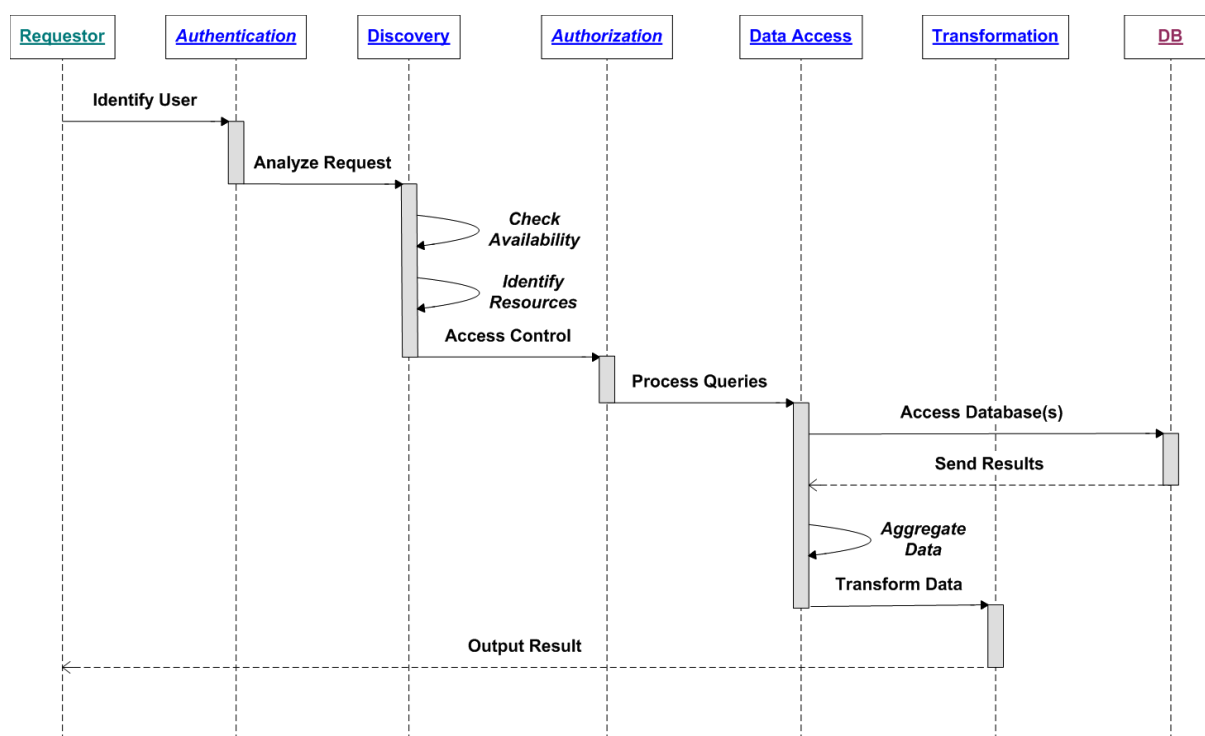
*Figure - 14: Process flow of a distributed data query*

# 5.2. Integration with Experiment Planning Environment

The Data Access Services are an integral part of the ViroLab Virtual Laboratory, but it cannot be directly interfaced by the ViroLab user layer. Hence, a separate module of the ViroLab runtime system, called the Data Access Client (DAC), has been implemented. The DAC has several important functions basically carrying out all communications with the Data Access Services including user authorization, submission of queries, and finally importing results into the context of the ViroLab experiment scripts. All these capabilities are provided in conjunction with the services' interfaces, which can be easily invoked and accessed via so-called web service stubs. The major service functionalities have been integrated with DAC and can be directly used within EPE although some specific WSRF properties have not been considered so far. Regarding secure interactions, all communications between DAC and the services are protected through an encrypted message transfer. Authentication takes place at the user site – experiments can only be executed after a user has been successfully identified by the infrastructure – while authorization is performed by the services [D2.3]. DAC simply forwards the current user identity token once it connects with DAS for the first time. As long as the token is valid, the user can run the experiment(s) again and again.

# 5.3. Outlook on future work

Apart from the current integration status, there are still some open issues, which should be covered within the rest of the project's timeline.

Firstly, all data providing project partners shall be connected to the ViroLab virtual laboratory by integrating their data sources via the data access

infrastructure. Therefore, their databases need to be initially migrated into the RegaDB schema – the common ViroLab data description language decided by all projects members – to guarantee the complete standardization for all available data resources and to ease the formulation of corresponding data requests. Plans for integrating databases from other European projects, e.g. EUresist are also ongoing.

Another task envisages the connection to the Provenance system in order to build a highly scalable data tracking system, and to ensure that data sets are not stored at multiple locations but kept consistent within the overall environment.

To support end-users, mainly doctors with more user-friendly applications that can be used during their daily hospital tasks, a specific portlet enabling the easy execution of on-demand drug resistance interpretations based on up-to-date patient information and the latest available rule sets, is planned to be developed within the scope of the ViroLab project. Therefore, different individual work packages need to cooperate together and integrate their developed services and tools into a real end-user application.

# 6. Provenance Tracking System (PROToS)

## 6.1. Integration with Monitoring Infrastructure

The provenance lifecycle in ViroLab can be divided into the following phases: (1) experiment monitoring, (2) event aggregation, (3) storage of provenance records, (4) provenance querying. The event aggregation phase is where events captured in the course of experiment execution are translated into an ontology representation and published in PROToS [Balis07]. The *Semantic Event Aggregator* component is responsible for the integration of the monitoring infrastructure and PROToS. The *event gathering interface* of PROToS is accessed directly by the Aggregator, responsible for the building of coherent, complete and valuable information. The monitoring events collected from various sources undergo correlation, aggregation and augmentation in order to create the ontology individuals translated directly to the PROToS-specific events. Basically, two types of events are collected:

- **generic events** – describe the execution of experiments in general; represented in XML based on the corresponding XSD schemas;
- **domain events** – describe the semantics of operations and data used in experiments; these events are concrete ontology individuals (RDF/OWL representation) augmented with ACID (Application Correlation Identifier) to correlate domain events with generic experiment events.

The event aggregation process is defined by aggregation rules specified in an **ontology extension**. Every aggregation rule describes:

- what event types should be aggregated,
- how events ACID should correspond with each other,
- what ontological classes should be instantiated when aggregation rule is satisfied.

When a new ontology individual is created, its identifier, functional and ontological properties are established, based on their *derivations* defined in the ontology extension. Two types of derivation are supported:

- **XML derivation** – indicates a concrete XML piece of data, which can be mapped directly to a datatype property.
- **Delegate derivation** – indicates a concrete Java class exposing methods utilized to constitute a datatype or object property, what involves e.g. mathematical operations applied to collected XML data, casting between XSD types or data base querying.

Created individuals are associated with unique IDs obtained as a result of applying a hashing function to the individual's properties or ACID number. In this way, upon creation a new individual, we can discover an already existing individual which must be associated with the one being created.

Unlike the generic events, the ontological events are mapped directly to the individuals, therefore they contain ontological property values defined explicitly. Optionally, in the case when collected information is incomplete, some additional properties can be established in Aggregator.

At the current stage of development, two ontologies are built by Aggregator – **experiment ontology** which is built from the generic events produced by GSEngine and GRR components, and a domain **DRS ontology**, built from the

ontological events. A simplified structure of those ontologies is depicted in Figure -15.
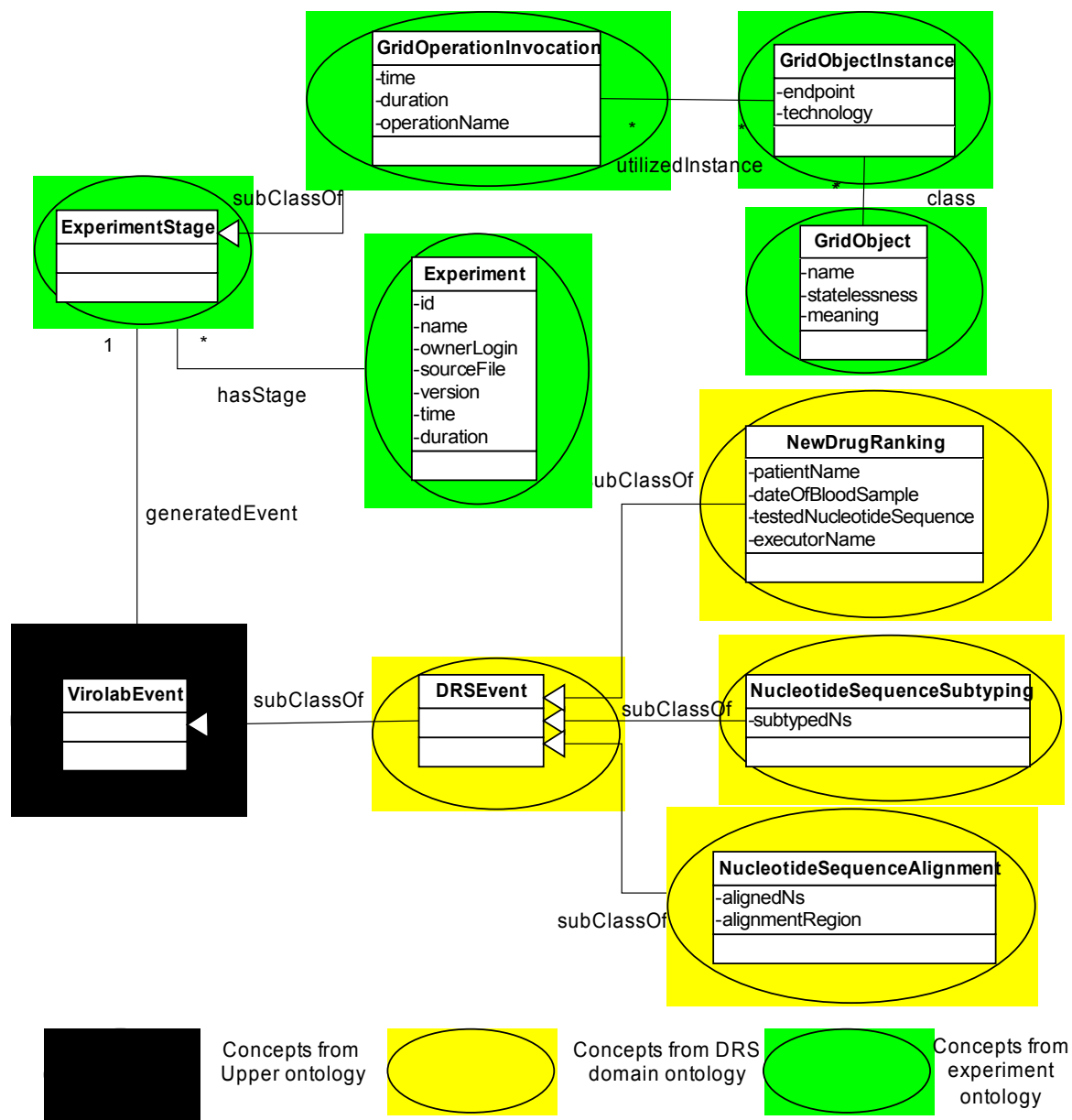


*Figure -15: PROToS ontology model*

The ontologies are semantically connected by a *generatedEvent* property, which is not known at the moment of creation of the corresponding domain and generic events. Therefore, all generic events must be temporarily stored in a buffer managed by Aggregator. When a new domain event occurs, it is associated with a buffered generic event that occurred nearly in time and which has a corresponding ACID number.

## 6.2. Integration with QUaTRO in Presentation Layer

As described [D2.3], QUaTRO is a tool that, by means of AJAX GUI, enables easy, end-user oriented querying to repositories of provenance and experiment data in the ViroLab Virtual Laboratory [Balis07-2].
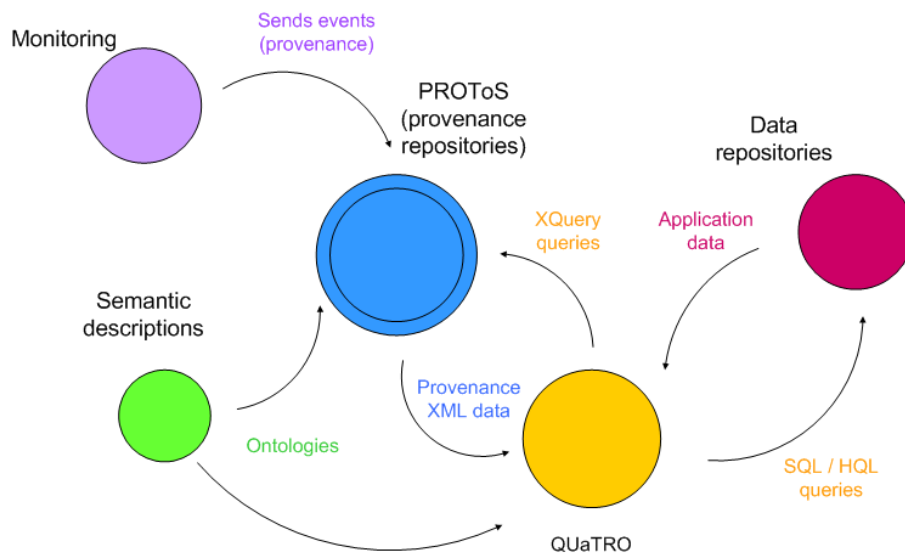
*Figure -16: QUaTRO component integration*

Currently, QUaTRO is fully integrated with the PROToS system, which serves as provenance repository, and with SQL-based experiment data sources, including CYFRONET's test data base (Figure -16). The integration of QUaTRO with PROToS is done by standard PROToS component – Data Retrieval Engine. This consists of data model for preparing queries and external interface, exposed and accessed by stateless Web Services.
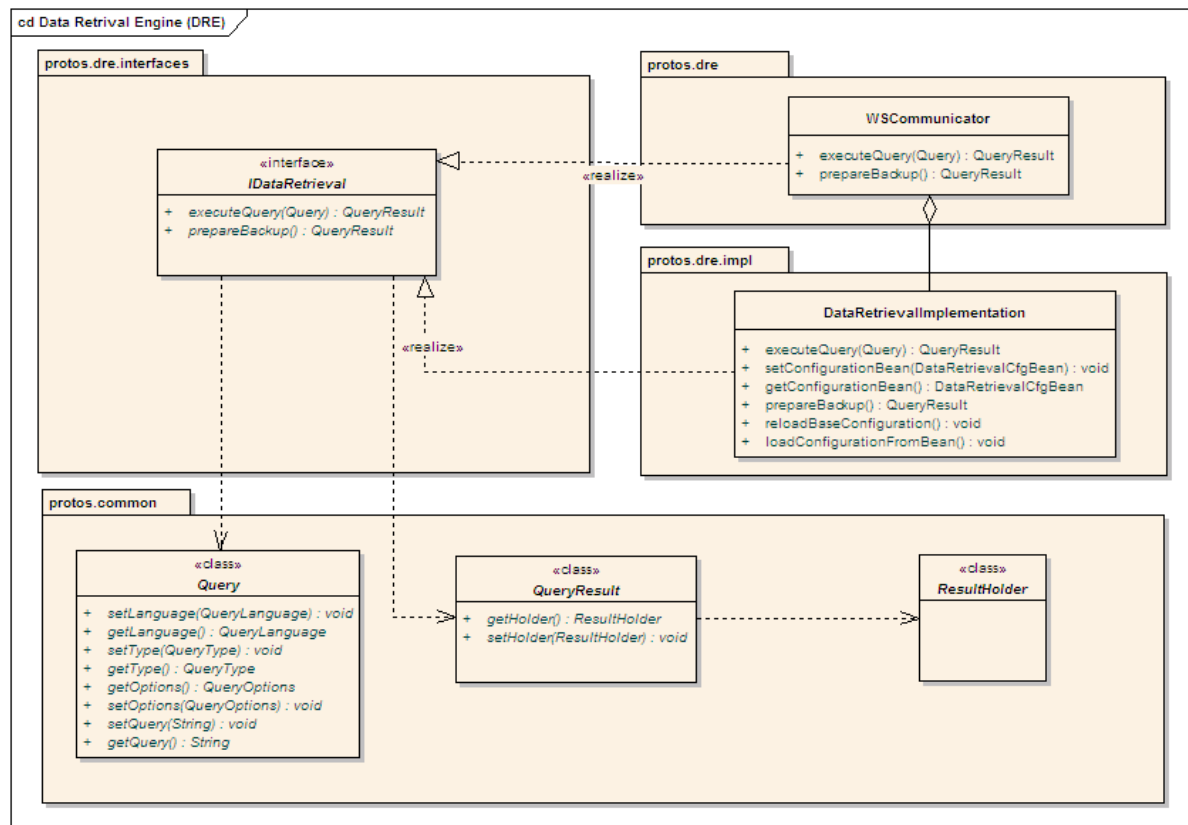
*Figure -17: PROToS DRE interface and data model*

In detail, as presented in Figure -17 a provenance query is submitted by calling the *executeQuery* method from *IDataRetrieval* interface. This should be preceded with preparation of instance of the *Query* class, containing provenance-mining *XQuery*. Thanks to using industry-standard communication technologies based on Web Services and a well-known, previously defined, Java interface, the integration of QUaTRO and PROToS was achieved without problems. In fact, we have been able to move from using mock objects to real PROToS interfaces in few hours.
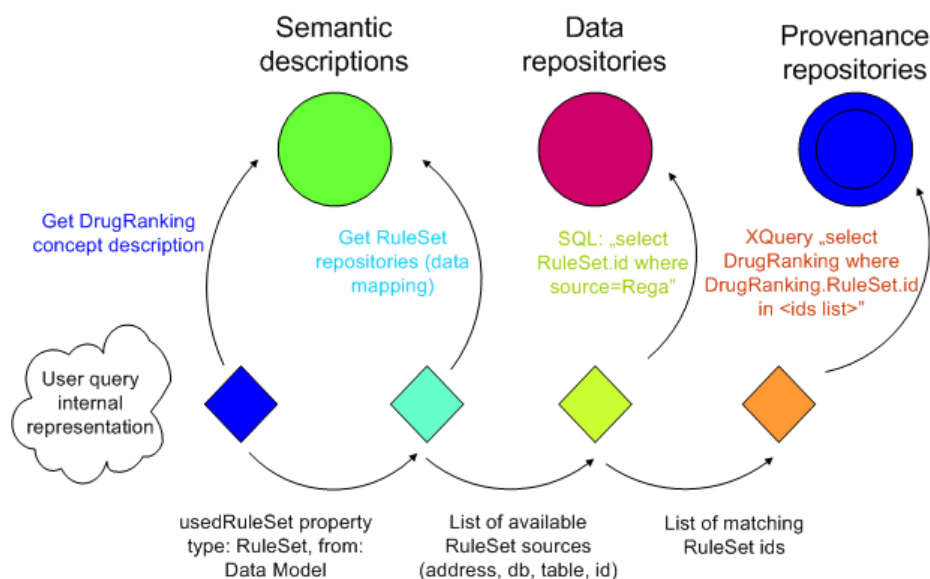
*Figure -18: Query construction process*

Let as examine the interaction of QUaTRO and PROToS by analyzing the evaluation of the following query: *"Select all drug rankings that were performed with some rule sets". Figure -18* shows how QUaTRO interacts with external components, especially the provenance repository (PROToS) in the course of user-defined query evaluation. The process starts when the user has defined a query using QUaTRO GUI. This query is represented in internal data format, which is essentially a tree whose leaves are evaluated as queries to PROToS or experiment databases. Some properties are mapped to underlying data sources and models. These are fetched from the specified data sources (currently CYFRONET's relational database). The query tree is then translated into XQuery language which is used by PROToS. Finally, the query is packed into a *Query-class* instance and sent to the PROToS DRE component through a Web Service-based interface. Query result is returned as XML and presented to the user in GUI.

Future work will concentrate on the integration with all data sources available in ViroLab, especially with those provided by the Data Access Service. This integration will be done by using the GSEngine's standard Data Access Client. We anticipate using a remote instance of GSEngine, running on a ViroLab server, to access the data sources. We are also planning to refactor QUaTRO code to provide a common, extensible repository access layer. This is essential to achieve a semantic integration with other repositories, which is part of our long-term plans.

# 7. List of Publications

The list below contains publications authored by persons involved in the development of the Virtual Laboratory. The list of delivered MSc theses is also provided.

- M. Assel, B. Krammer, A. Loehden. *Data Access and Virtualization within ViroLab*. In Proceedings of the 7th Cracow Grid Workshop 2007, pp. 77-84, Cracow, Poland, October 2007.
- M. Assel, B. Krammer, A. Loehden. *Management and Access of Biomedical Data in a Grid Environment*. In Proceedings of the 6th Cracow Grid Workshop 2006, pp. 263-270, Cracow, Poland, October 2006.
- M. Assel, A. Kipp. *A Secure Infrastructure for Dynamic Collaborative Working Environments*. In Proceedings of the 2007 International Conference on Grid Computing and Applications (GCA'07/ISBN #:1-60132-032-9/CSREA), Editor: Hamid R. Arabnia, pp. 212-216, Las Vegas, USA, June 2007.
- B. Balis, M. Bubak, and M. Pelczar. From Monitoring Data to Experiment Information -- Monitoring of Grid Scientific Workflows. In G. Fox, K. Chiu, and R. Buyya, editors, *Third IEEE International Conference on e-Science and Grid Computing, e-Science 2007, Bangalore, India, 10-13 December 2007*, pages 187-194. IEEE Computer Society, 2007.
- B. Balis, M. Bubak, and J. Wach. User-Oriented Querying over Repositories of Data and Provenance. In G. Fox, K. Chiu, and R. Buyya, editors, *Third IEEE International Conference on e-Science and Grid Computing, e-Science 2007, Bangalore, India, 10-13 December 2007*, pages 77-84. IEEE Computer Society, 2007.
- T. Bartyński, M. Malawski, T. Gubała, M. Bubak: *Universal Grid Client: Grid Operation Invoker*; 7-th International Conference on Parallel Processing and Applied Mathematics PPAM'2007, (LNCS 4967 to appear)
- T. Bartyński, M. Malawski, M. Bubak: *Invocation of Grid Operations in the ViroLab Virtual Laboratory*; in Cracow Grid Workshop 2007 Workshop Proceedings, pp.59-64, ACC CYFRONET AGH 2008
- E. Ciepiela, J. Kocot, T. Gubala, M. Malawski, M. Kasztelnik, M. Bubak: *Virtual Laboratory Engine - GridSpace Engine*; in Cracow Grid Workshop 2007 Workshop Proceedings, pp.53-58, ACC CYFRONET AGH 2008
- M. Bubak, T. Gubala, M. Kasztelnik, M. Malawski, P. Nowakowski, P.M.A. Sloot: *Collaborative Virtual Laboratory for e-Health*; in P. Cunningham and M. Cunningham, editors, Expanding the Knowledge Economy: Issues, Applications, Case Studies, eChallenges e-2007 Conference Proceedings, pp. 537-544. IOS Press, 2007.
- T. Gubała, B. Baliś, M. Malawski, M. Kasztelnik, P. Nowakowski, M. Assel, D. Harężlak, T. Bartyński, J. Kocot, E. Ciepiela, D. Król, J. Wach, M. Pelczar, W. Funika, M. Bubak: *ViroLab Virtual Laboratory* in Cracow Grid Workshop 2007 Workshop Proceedings, pp. 35-40, ACC CYFRONET AGH 2008
- M. Kasztelnik, T. Gubała, M. Malawski, M. Bubak: *Development and Execution of Collaborative Application on the ViroLab Virtual Laboratory;* in

Cracow Grid Workshop 2007 Workshop Proceedings, pp.41-46, ACC CYFRONET AGH 2008

- M. Malawski, J. Kocot, E. Ciepiela, M. Bubak: *Optimization of Application Execution in the ViroLab Virtual Laboratory*; in Cracow Grid Workshop 2007 Workshop Proceedings, pp.65-70, ACC CYFRONET AGH 2008

Finished Master of Science theses related to the research in the frame of ViroLab:

- Joanna Kocot, Iwona Ryszka: *Optimization of Grid Application Execution*; Master of Science Thesis supervised by Marian Bubak; AGH University of Science and Technology, June 2007, Krakow, Poland

- Eryk Ciepiela: *Monitoring of Component-Based Applications*; Master of Science Thesis supervised by Marian Bubak; AGH University of Science and Technology, June 2007, Krakow, Poland

- Tomasz Bartyński: *Remote execution of delegated operations with support for automatic selection among multiple communication protocols*; Master of Science Thesis supervised by Marian Bubak; AGH University of Science and Technology, February 2008, Krakow, Poland

# 8. Summary

This document presents the core components of the ViroLab WP3 and the status of their integration after 24 months of the project. 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 status of the ViroLab Virtual Laboratory and also to present guidelines and plans for the remaining period of ViroLab development.

As described in the design deliverable, we follow a phased approach, progressing from simple functionality to more advanced elements of the architecture as time passes. We expect to be able to report on the final version of the Virtual Laboratory system and its validation in the next scheduled WP3 deliverable, which is due to be released in Month 36 of the Project.

# 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

[Assel06]          M. Assel, B. Krammer, and A. Loehden. *Management and Access of Biomedical Data in a Grid Environment*. In Proceedings of the 6th Cracow Grid Workshop 2006, pp. 263-270, Cracow, Poland, October 2006.

[Assel07]          M. Assel, B. Krammer, and A. Loehden. *Data Access and Virtualization within ViroLab*. In Proceedings of the 7th Cracow Grid Workshop 2007, pp. 77-84, Cracow, Poland, October 2007.

[Balis07]          B. Balis, M. Bubak, and M. Pelczar. From Monitoring Data to Experiment Information -- Monitoring of Grid Scientific Workflows. In G. Fox, K. Chiu, and R. Buyya, editors, *Third IEEE International Conference on e-Science and Grid Computing, e-Science 2007, Bangalore, India, 10-13 December 2007*, pages 187-194. IEEE Computer Society, 2007.

[Balis07-2]        B. Balis, M. Bubak, and J. Wach. User-Oriented Querying over Repositories of Data and Provenance. In G. Fox, K. Chiu, and R. Buyya, editors, *Third IEEE International Conference on e-Science and Grid Computing, e-Science 2007, Bangalore, India, 10-13 December 2007*, pages 77-84. IEEE Computer Society, 2007.

[Bartynski07]      T. Bartyński, M. Malawski, T. Gubała and M. Bubak: *Universal Grid Client: Grid Operation Invoker*; 7-th International Conference on Parallel Processing and Applied Mathematics PPAM'2007, (LNCS 4967 to appear)

[Bubak07]          M. Bubak, T. Gubala, M. Kasztelnik, M. Malawski, P. Nowakowski, P.M.A. Sloot: *Collaborative Virtual Laboratory for e-Health*; in P. Cunningham and M. Cunningham, editors, Expanding the Knowledge Economy: Issues, Applications, Case Studies, eChallenges e-2007 Conference Proceedings, pp. 537-544. IOS Press, Amsterdam, 2007.

[Ciepiela07]       E. Ciepiela, J. Kocot, T. Gubala, M. Malawski, M. Kasztelnik, M. Bubak: *Virtual Laboratory Engine - GridSpace Engine*; in Cracow Grid Workshop 2007 Workshop Proceedings, pp.53-58, ACC CYFRONET AGH 2008

[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

| [D2.3] | ViroLab Project. *D2.3 – ViroLab VO version 1 deployment, integration with WP3, WP4 and WP5 – report and demonstration*. ViroLab Project Consortium, 2008 |
|---|---|
| [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 |
| [D3.3] | ViroLab Project. *D3.3 – Session Manager, runtime system and data layer: installation, integration and usage; description of interfaces to WP2, WP4 and WP5 - report and demonstration.* 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 |
| [Gubala07] | T. Gubała, B. Baliś, M. Malawski, M. Kasztelnik, P. Nowakowski, M. Assel, D. Harężlak, T. Bartyński, J. Kocot, E. Ciepiela, D. Król, J. Wach, M. Pelczar, W. Funika, M. Bubak*: ViroLab Virtual Laboratory* in Cracow Grid Workshop 2007 Workshop Proceedings, pp. 35-40, ACC CYFRONET AGH 2008 |
| [GWT] | Google Web Toolkit, http://code.google.com/webtoolkit |
| [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 |
| [Kipp07] | M. Assel and A. Kipp. *A Secure Infrastructure for Dynamic Collaborative Working Environments*. In Proceedings of the 2007 International Conference on Grid Computing and Applications (GCA'07/ISBN #:1-60132-032-9/CSREA), Editor: Hamid R. Arabnia, pp. 212-216, Las Vegas, USA, June 2007. |
| [Malawski07] | M. Malawski, J. Kocot, E. Ciepiela, M. Bubak: *Optimization of Application Execution in the ViroLab Virtual Laboratory*; in Cracow Grid Workshop 2007 Workshop Proceedings, pp.65-70, ACC CYFRONET AGH 2008 |
| [SPRING] | Spring Framework, www.springframework.org |
| [SVNKIT] | Subversion for Java: http://svnkit.com/ |

[VIROLAB]            The ViroLab Project Website. http://www.virolab.org

[VIROLAB-VL]         The ViroLab Virtual Laboratory Website.
                     http://virolab.cyfronet.pl/

[VLINV]              T. Bartyński, M. Malawski, M. Bubak *Invocation of Grid
                     Operations in the ViroLab Virtual Laboratory*; in Cracow
                     Grid Workshop 2007 Workshop Proceedings, pp.59-64,
                     ACC CYFRONET AGH 2008