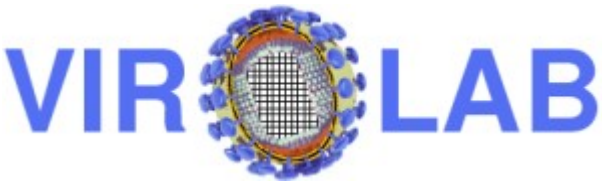


Deliverable 2.3
ViroLab VO version 1 deployment, integration with WP3, WP4 and WP5 - report and demonstration

Project Start:	01-03-2006
Project Duration:	36 Months
Priority area	2.4.11
Contract No.:	INFSO-IST-033446
Website:	http://www.virolab.org

Due-Date:	29-02-2008
Delivery:	14-04-2008
Lead Partner:	CYFRONET
Dissemination Level:	Public
Status:	Final
Approved:	Q Board, Project Steering Group
Version:	1.4

Log of Document

Version	Date	Changes Summary	Authors
0.1	9/01/2008	Initial draft from template	Maciej Malawski, Pawel Jarosz
0.2	24/01/2008	Next draft with names	Maciej Malawski
0.3	27/02/2008	Added figure showing integration between GSEngine and Shibboleth	Eryk Ciepiela
0.4	28/02/2008	Added description of the EPE, including features in version 1, overview of the integration points and planned functionalities part.	Włodzimierz Funika, Dariusz Król
0.5	28/02/2008	Sections on EMI integration with GSEngine and Experiment Repository updated	Daniel Harezlak
0.6	28/02/2008	DOS and OntoBrow sections added	Tomasz Gubala
0.7	28/02/2008	Sections on integration between Security and GSEngine	Eryk Ciepiela
0.8	28/02/2008	Sections: Middleware – features of v.1 - monitoring and monitoring/register integration	Bartłomiej Łabno
0.9	28/02/2008	Section on GRR integration with EPE added	Marek Kasztelnik
0.10	28/02/2008		Kuba Wach
0.11	29/02/2008	Input on middleware	Tomasz Bartyński
0.12	29/02/2008	Detailed description of monitoring features of GSAM, interface between GSAM and GRR and scenario of	Bartłomiej Łabno

Version	Date	Changes Summary	Authors
		monitoring.	
0.13	29/02/2008	Corrections to 4.1	Włodzimierz Funika, Dariusz Król
0.14	29/02/2008	Corrections to images in section 4.1	Włodzimierz Funika, Dariusz Król
0.15	29/02/2008	Input from GridwiseTech Input about DAS Integrating documents, Exec. summary	Pawel Jarosz, Stanisław Kulczycki, Chris Wilk Matthias Assel Maciej Malawski
0.16	29/02/2008	Minor text changes to EPE fragments	Włodzimierz Funika
0.17	29/02/2008	Update of sections 4.2.2.1 (QUaTRO) and 5.2.3 (GRR/Monitoring)	Bartosz Balis
0.18	1/3/2008	Update of section 5.2.2	Michał Pelczar
0.19	3/3/2008	Middleware diagram, formatting	Maciej Malawski
0.20	26/3/2008	Major changes in 2,3,5 sections	Pawel Jarosz, Pawel Plaszczak
0.21	28/3/2008	Input on integration between EPE and Shibboleth	Dariusz Król
0.22	28/3/2008	Input on Shibidpclient – authentication library enabling handle requesting for non-Web clients	Jan Meizner
0.23	29/3/2008	Update of sections 5.2.2 (Provenance/Monitoring) and 5.2.3 (GRR/Monitoring)	Bartosz Balis
0.24	29/3/2008	Cosmetics	Maciej Malawski
1.0	31/3/2008	Producing final draft for QB	Maciej Malawski
1.1	10/04/2008	Update demo	Pawel Jarosz, Pawel Plaszczak
1.2	10/04/2008	Added EPE demo	Maciej Malawski
1.3	10/04/2008	Added EMI description	Daniel Harezlak

Version	Date	Changes Summary	Authors
1.3.1	11/04/2008	Conclusions, polishing	Pawel Plaszczak
1.3.2	13/04/2008	Corrections to Presentation Layer	Włodzimierz Funika
1.4	14/04/2008	Final corrections	Maciej Malawski

Table of Contents

List of Figures

Executive Summary

This document describes the current status of ViroLab virtual organization in Version 1, which was reached after the second year of the project. The system, as presented in this document, has been deployed in a widely distributed installation, covering components and resources located at facilities of several consortium stakeholders.

Section 1 of the document introduces the main concepts of attribute-based Virtual Organization model, together with the presentation and middleware layer, as well as the security infrastructure based on Shibboleth.

The demonstration of ViroLab VO (Section 2) shows how the modules are integrated with the security infrastructure: the user logs in into ViroLab portal and is authenticated using Shibboleth, then is able to browse experiment repository and execute experiments using integrated Experiment Management Interface (EMI). The EMI is integrated with GSEngine (developed in WP3), which is able to contact data sources and invoke remote operations using the middleware layer. It is also possible for experiment developer to create and edit experiment plans using Experiment Planning Environment (EPE).

Current version of presentation layer (Section 3) includes a GridSphere-based portal which is integrated with the security infrastructure and includes Web applications for Data Browsing, Experiment Management and Provenance querying. The EPE integrates plugins for experiment repository browsing, execution using GSEngine, Registry and Domain Ontology Store browsing and editing.

Middleware layer (Section 4) includes access to computing resources and supports such technologies as Web Services, MOCCA components, EGEE jobs and initial support for WSRF services. The middleware was integrated with WP3 (GSEngine) and supports ViroLab applications (WP4, WP5) such as Drug Ranking System (as a Web Service) and RegaDB Tools (alignment, subtyping) using WTS services. Middleware layer includes also support for infrastructure monitoring and the integration with Provenance system is under development.

Security in ViroLab (Section 5) is based on Shibboleth, providing single-sign on for Portal access and attribute-based access control. Attribute request library has been implemented, integrated with Experiment repository and Apache Axis for Web services. GSEngine includes support for delegating Shibboleth credentials and using them to access Shibboleth-secured Data Access Service (DAS) and Experiment Repository.

1 Overview of ViroLab Virtual Organization

The foremost technical challenge in ViroLab is to provide a communication fabric that connects all partners in a secure, efficient and transparent way. In the envisioned environment, there is space for several user roles, different tasks and duties. There are also many distributed resources that require access control, with optimally scalable and flexible security patterns.

To start with, let us elaborate on the concept of a Virtual Organization (VO), which is an idea that is at the foundation of modern Grid architectures. In short, a VO is a security domain extended over a set of entities (individuals, services, resources), allowing them to establish and maintain a mutual trust relationship. VOs are used to interconnect members of different physical institutions and organizations. Membership to the VO is not constant – it can be enabled and disabled at any time. For example, two entities (i.e. Doctor A and hospital B) within the same VO can dynamically establish each other's identity and start secure communication, even if they had no prior knowledge of each others existence.

The example shows that a VO is a very powerful, yet abstract concept. There has been much effort from the Grid computing community, although is often not fully implemented. Over the past decade, most European grid installations have implemented VOs using derivatives of the Grid Security Infrastructure (GSI) package, which originated from the Globus project. GSI-based VOs integrate well with local certificate-based security schemes, but generally miss the dynamic nature that is at the core of the VO concept.

ViroLab implements a VO in a novel and, in our opinion, powerful way. ViroLab VOs are truly dynamic and truly scalable. We consider this approach, and the fact of its successful implementation as an important achievement of WP2.

It should also be noted that a VO is sometimes understood in its broader, common-language sense. Its intuitive meaning, generally refers to all aspects of IT technology, allowing parties from different institutions to communicate remotely and securely. Such should be the understanding of the title of ViroLab's Work Package 2, which is not only about a security layer, but also encompasses work on the layers of presentation and middleware (both closely integrated with security). It is the combination of these three layers that enables users to interact transparently and securely with their work environment.

However, as this document is intended to focus on the technology used in the security layer, we will use the term VO in its primary meaning " a collection of entities united by mutual trust".

The security primitives at ViroLab must cater for three types of users [D3.2]:

Experiment developer – main task is to plan, design, implement and provide scientific experiments conducted on the Grid infrastructure. Requires programming knowledge as well as possessing domain-specific knowledge.

Experiment user – executes the implemented experiments,analyzing and managing the obtained results. This user is commonly a scientist with in-depth knowledge of the domain.

Clinical virologist mainly interested in obtaining drug ranking reports based on selected rule sets and input in the form of lists of virus mutations.

Each of the above mentioned user roles has specific needs in terms of applications, services and data. This brings us to the natural – and very pragmatic – use of the Virtual Organization abstraction:

A ViroLab VO is a group of entities (typically users) that share legitimate need to access similar resources (typically, applications or data). VO membership can be further restricted by other characteristics (attributes) common to the group of entities.

As clinical virologists share a common goal, no matter which physical organization they are from, they could belong to the same "virologist VO". The same applies to other roles in ViroLab. On the other hand, experiment developers and users need to have access to the actual clinical data; therefore, they need to belong to a separate "clinical data VO" that gives them the required privileges.

Here we should stress that the security implementation in Virolab allows for very easy set up of unlimited number of VOs in various configurations. The three VOs (developers, users, virologists) that we have currently set up serve as example groups for system validation, but do not prevent from creating new ones at any point in time.

A typical scenario is as follows:

A Hospital in Rome acts as a resource provider that stores sensitive patients data. A virologist from the University of Amsterdam would like to execute their experiments using the patient data. The Hospital in Rome regulations will only accept professional virologists from associated universities as trusted users.

From this scenario we can view the Hospital in Rome as a Resource Provider and the University of Amsterdam as a Home Organization. Thanks to the *virologist* VO membership, the scientists from Amsterdam can get access to the secured, historical patient data from Rome. Once the

administrators of the Hospital in Rome decide to deny access to that group of users, they ban the Virologist VO in their policies. Alternatively, when the virologist at the University of Amsterdam changes positions, her Home Organization would update her profile so she automatically looses membership in the Virologist VO.

In this model, the key authority responsible for granting users the VO membership is their Home Organization, while the resource access is additionally controlled by the resource owner. The resource owner can apply per-VO access policy, but other, stricter or more complex access policies are possible and equally easy to implement (per user, per user role, or per user's home institution). In summary, the ViroLab architecture provides the following features:

1. access to Virtual Laboratory functionality using the ViroLab Portal;
2. users belong to their Home Organization and these institutions are responsible for user authentication;
3. for all activities in Virtual Laboratory a user is transparently identified by a verifying the users authentication;
4. a resource can be secured by configuring a authorization policy and including an appropriate security module;
5. there is an external tool for experiment management (creation, edition) which is an Eclipse RCP application – Experiment Planning Environment (EPE);
6. there is sophisticated middleware which provides access to computation power and monitoring

The overall software architecture consists of three layers presented as (from top to bottom):

- Presentation Layer - provisions system functionality to the user by means of two graphical interfaces: Portal for users (experiment users as well as clinical virologist), and EPE for experiment developers.
- Security Layer - provides access control, secure identity management and data protection.
- Virtual Laboratory - also called the application layer and is the lowest layer It contains applications, data and simulations together with the runtime library, provenance and monitoring modules.

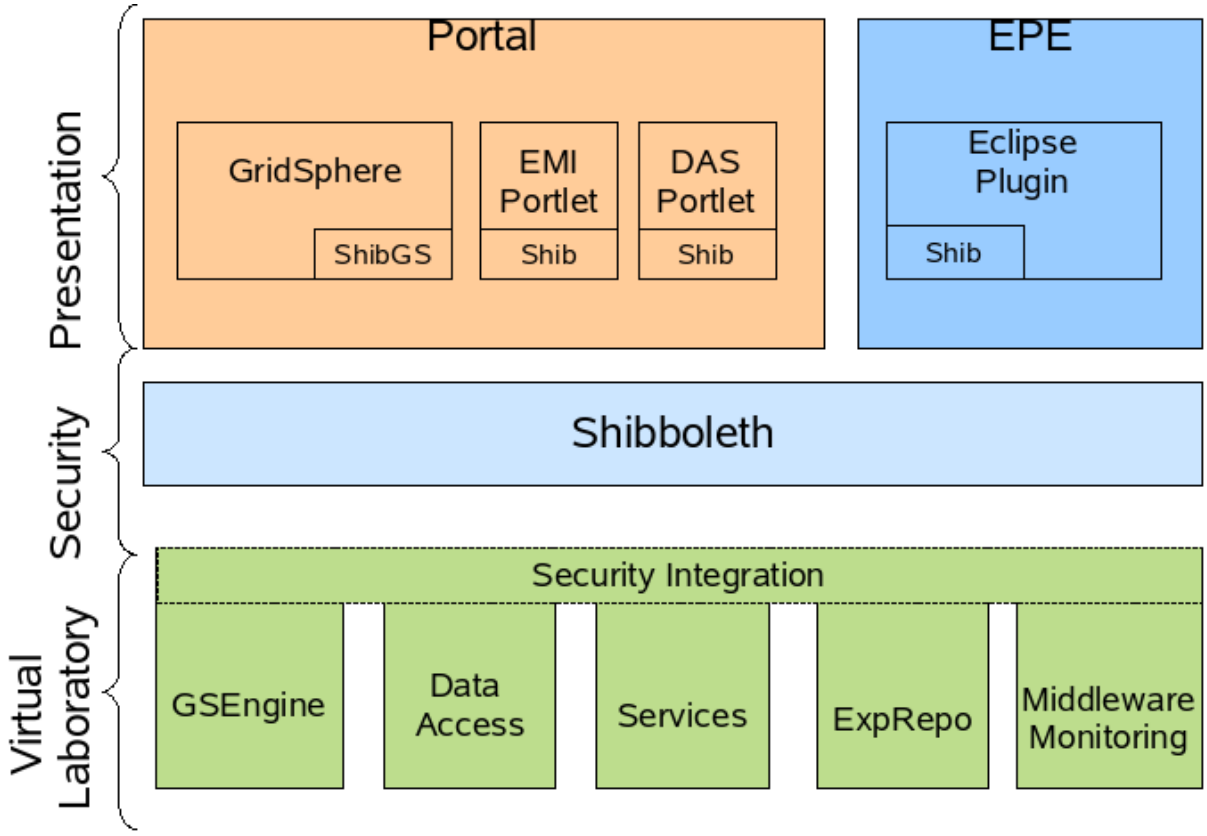


Figure 1: Overall architecture of ViroLab VO

2 Demonstration of the ViroLab VO

To demonstrate how Virtual Organization is created in ViroLab, two typical scenarios is presented.

The first one involves a user such as a virologist working via a portal, and the second one shows the experiment developer working with Experiment Planning Environment.

Scenario 1: Finding out the resistance of the virus based on HIV nucleotide sequence

User: Clinical Virologist

Goal: User would like to learn about resistance of a particular virus mutation.

Scenario:

1. User opens the ViroLab portal (https://virolab.gridwisetech.pl) (Figure 3).
2. User chooses his/her Home Organization (Figure 4).
3. User enters his/her credentials in order to proceed with the authentication (Figure 5).
4. User chooses the Experiment Management Interface (EMI) portlet (Figure 6).
5. User chooses the experiment which he/she would like to run (Figure 7) (*geno2drs* - this experiment takes a HIV nucleotide sequence, aligns it with respect to a reference strain, detects its subtype, then finds mutations in an indicated region and, finally, runs the ViroLab Drug Ranking to learn about resistance of the virus).
6. User runs the experiment and provides the HIV nucleotide sequence (Figure 8).
7. After a while, user gets the results of the experiment.

Alternative scenarios:

If user is not eligible to access to ExpRepo (repository of experiments), he or she would receive the "access denied" message (Figure 9)

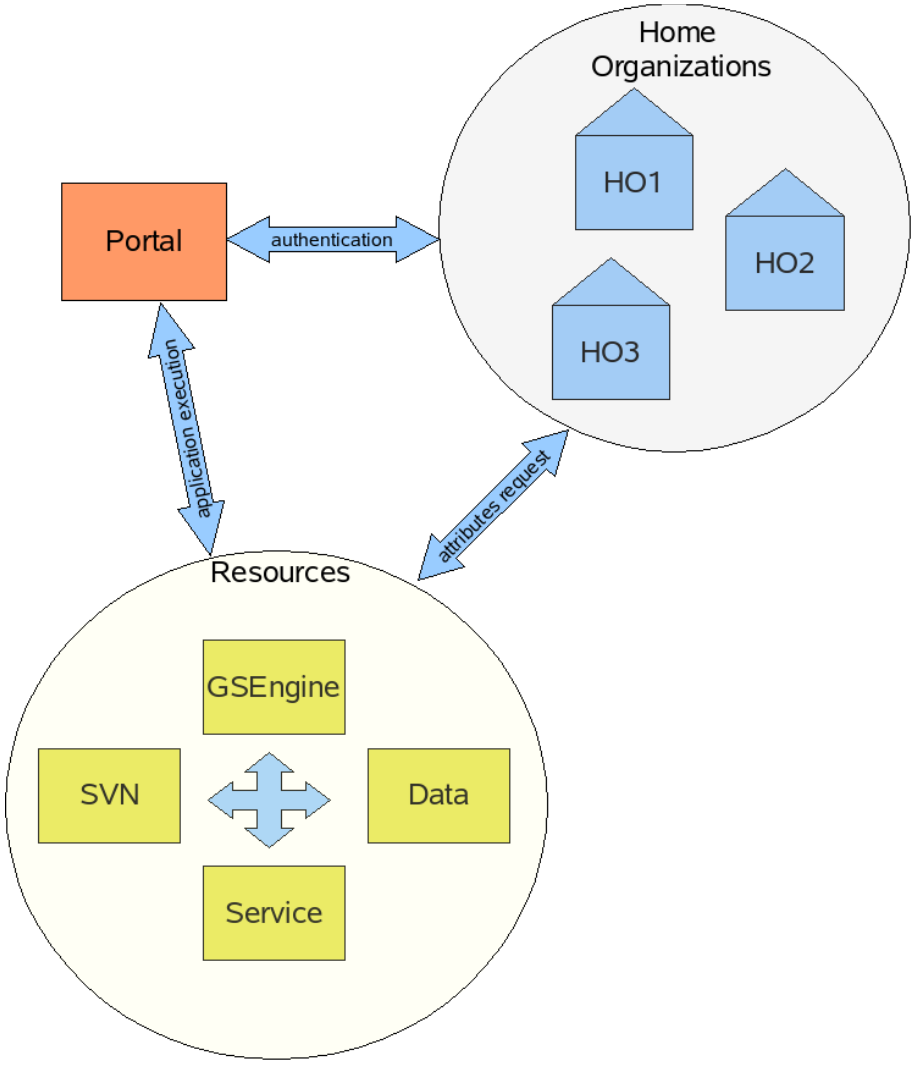


Figure 2 General flow diagram of the scenario

The Figure 2 presents the general flow of the scenario. Users using portal first use their Home Organization in order to authenticate. Next the application is executed and resources invoked. In order to acces each resources attributes are requested from user HO.

Figure 3 Use case – user goes to the ViroLab portal

Figure 4: Use case - user chooses HO

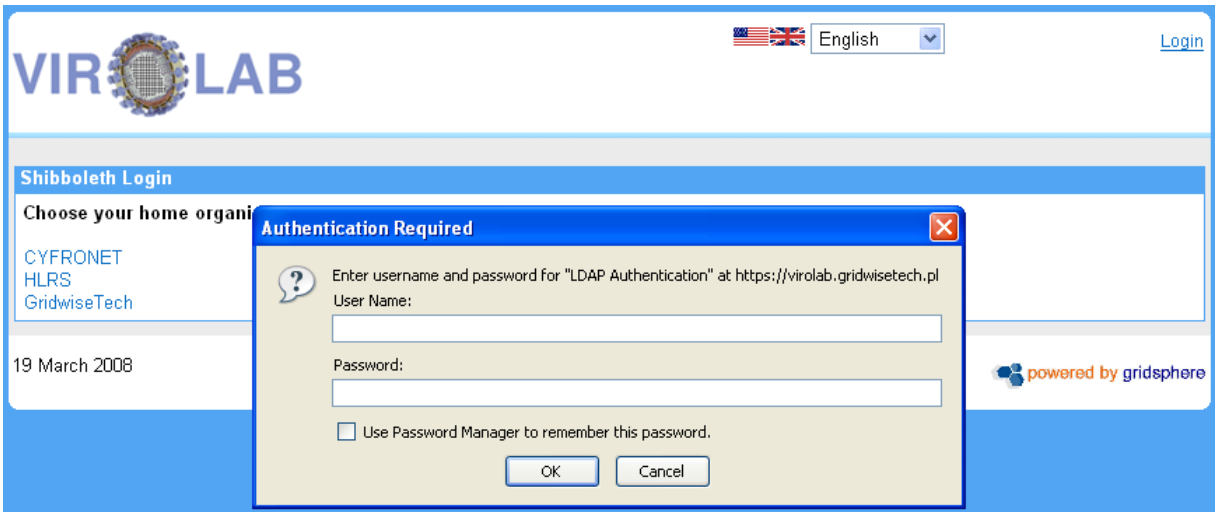


Figure 5: User enters credentials

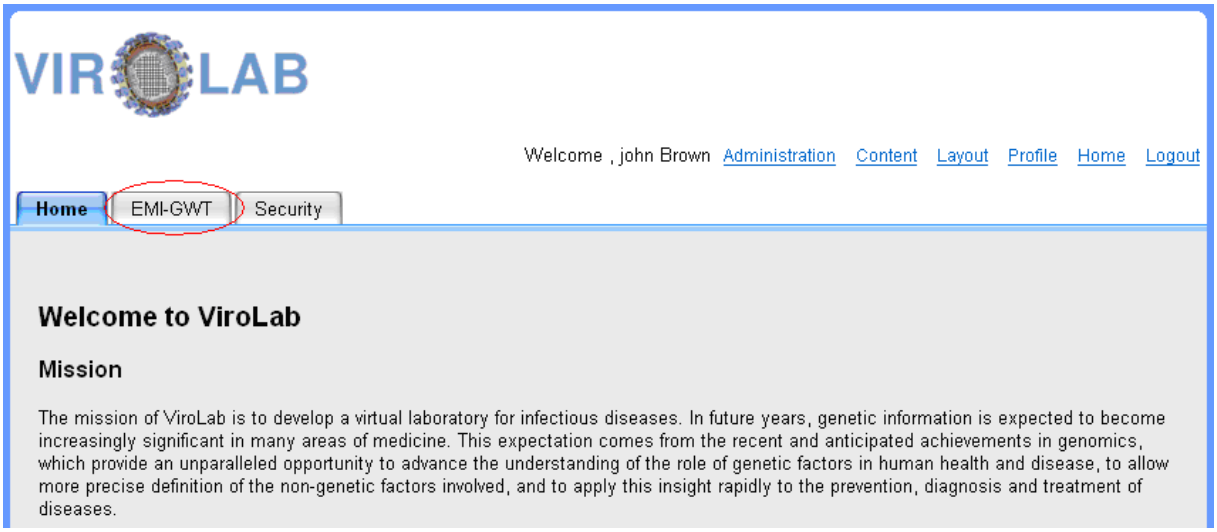


Figure 6: User chooses the EMI application

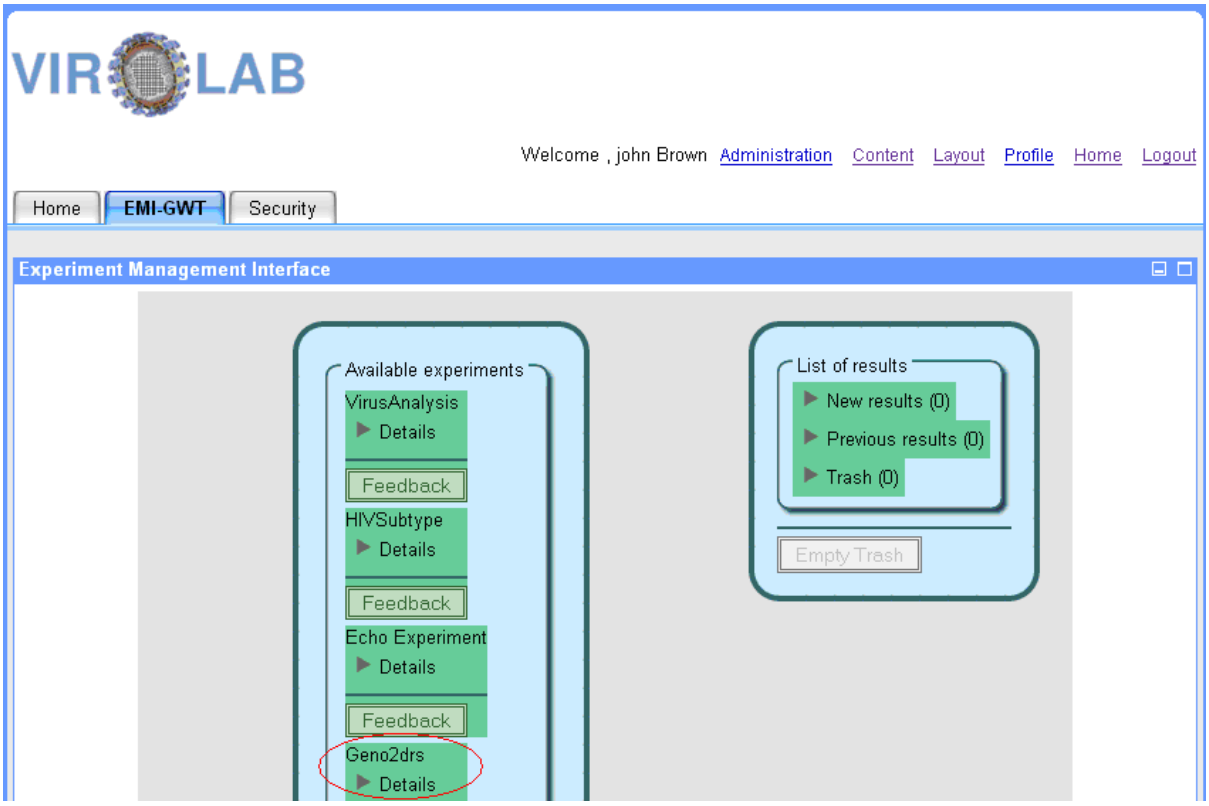


Figure 7: User chooses the experiment

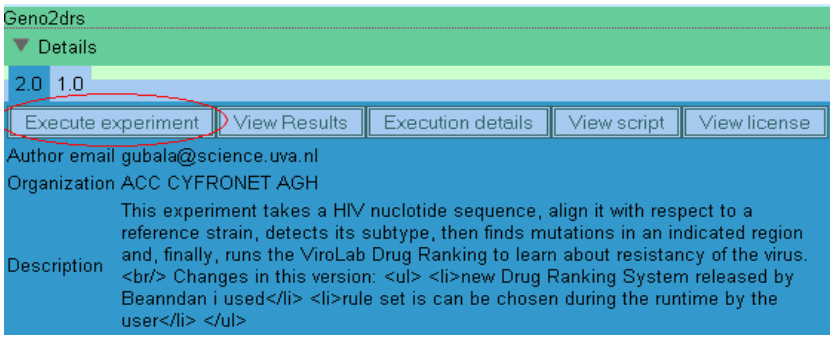


Figure 8: User executes the experiment

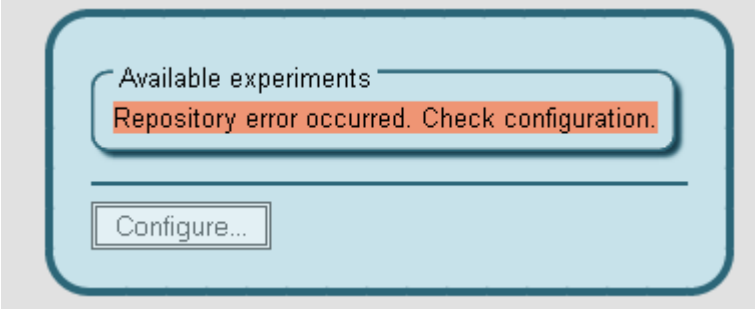


Figure 9: Access to repository denied for unauthorized user

The beauty of a good IT architecture lies in its most complex features being totally transparent to the user. This is also the case here. There are three places in the presented scenario where such advanced operations are performed.

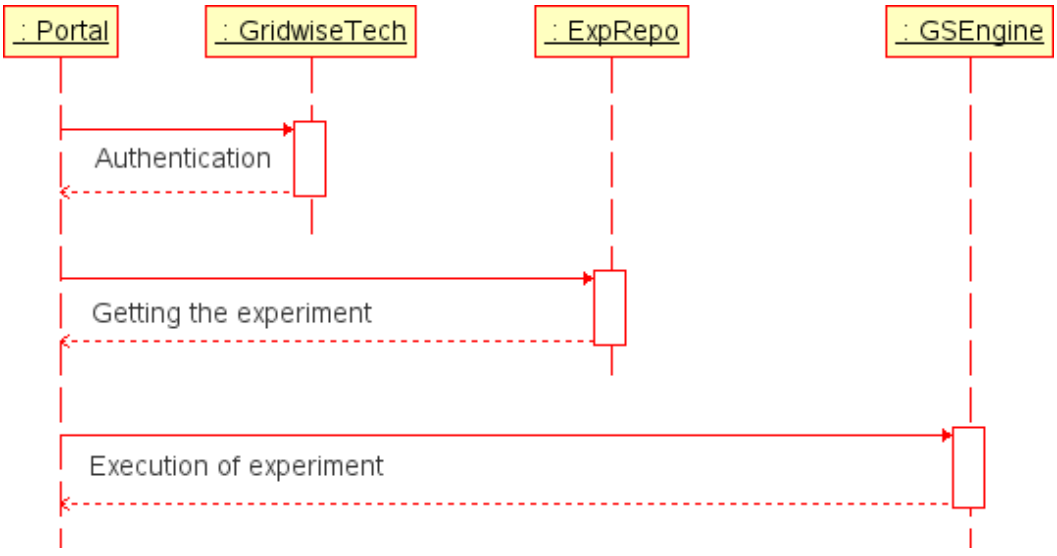


Figure 10: General sequence diagram of presented scenario

The Figure 10 presents the general sequence diagram of the scenario. The first sensitive step is Authentication (steps 1, 2, 3 from the scenario). Then the retrieval experiments from the repository (steps 4, 5) and, finally, the execution of the experiment (steps 6, 7).

The user entering his/her password in step 3 has no idea of the complex process he/she puts in motion. Firstly, the user chooses the Home Organization which he/she belongs to. Then the mechanisms at the site of Home Organization are run. The user credentials are checked with local databases. If the user is authenticated, the *handle* (identifying data) is created and sent back to the portal.

If the user would like to get the experiment, he/she is checked for privileges. If the user's attributes fit with the policy set for the repository, he/she will get access to the experiment. At Figure 10 this process is presented in detail.

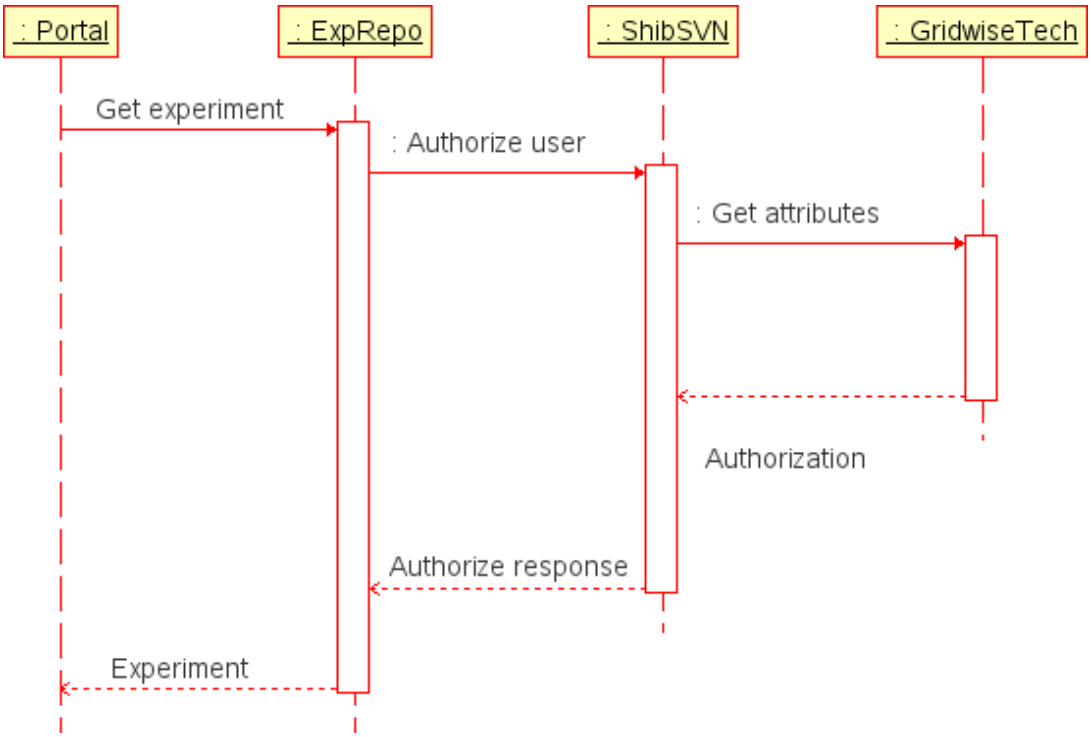


Figure 11: Detailed sequence diagram for getting the experiment from repository (step 4, 5 from scenario)

In order to get the experiment, ExpRepo module (Experiment Repository) has to authorize the user. The ShibSVN is invoked to perform that. The ShibSVN connects with Home Organization of the user (GridwiseTech in particular in this example) to get the attributes describing the user. Based on that attributes and policy set for repository, the authorization decision is made. If the user is authorized, the experiment is downloaded from repository and sent to the portal. The process for the presented scenario enables an unauthorized user to get access to the experiments.

Another point after the retrieval of the experiment is its execution (Figure 11). If the user chooses the option to execute the experiment, the latter is sent to GSEngine module (responsible for applications execution). The experiment may consist of several operations, including those which are invoked remotely. Such remote services can also be secured. In such a case the execution of them is preceded by the authorization process. In this case the module responsible for authorization is ShibAuthAPI, and its

behavior consists in getting the user attributes from his/her HO and then making the authorization decision.

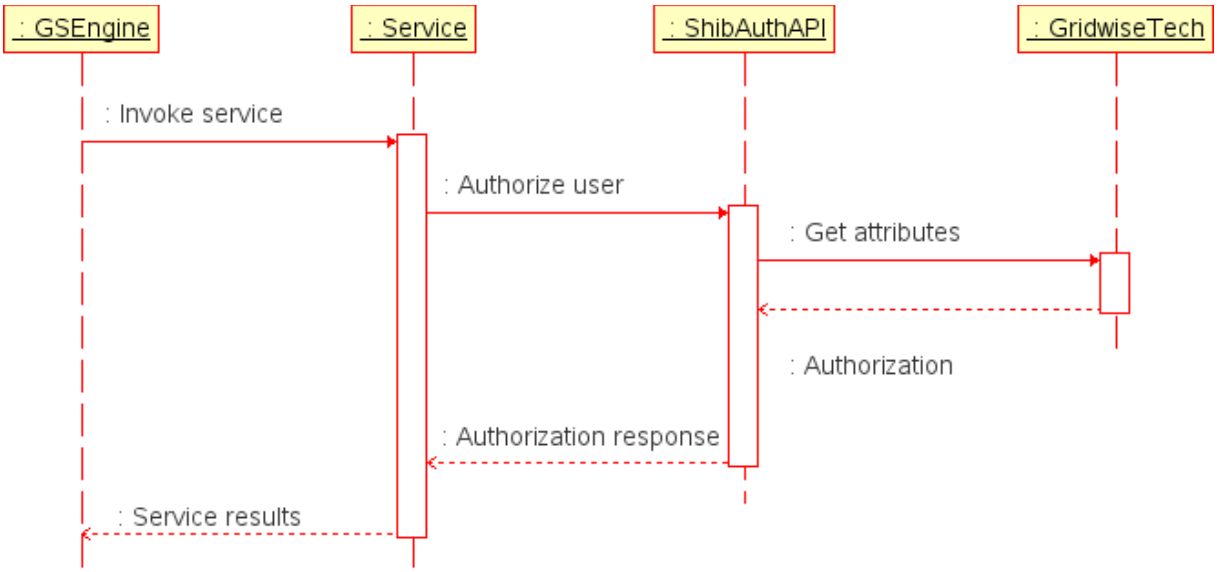


Figure 12: Detailed sequence diagram for the experiment execution (steps 6,7 from the scenario)

During the experiment execution the data is retrieved from the data center. It is important that the data is secured as well. When experiment try to get the data the same process as presented when accessing the service is performed. The data is always received via the Data Access Services (DAS). At Figure 12 the detailed sequence for the data retrieval is presented.

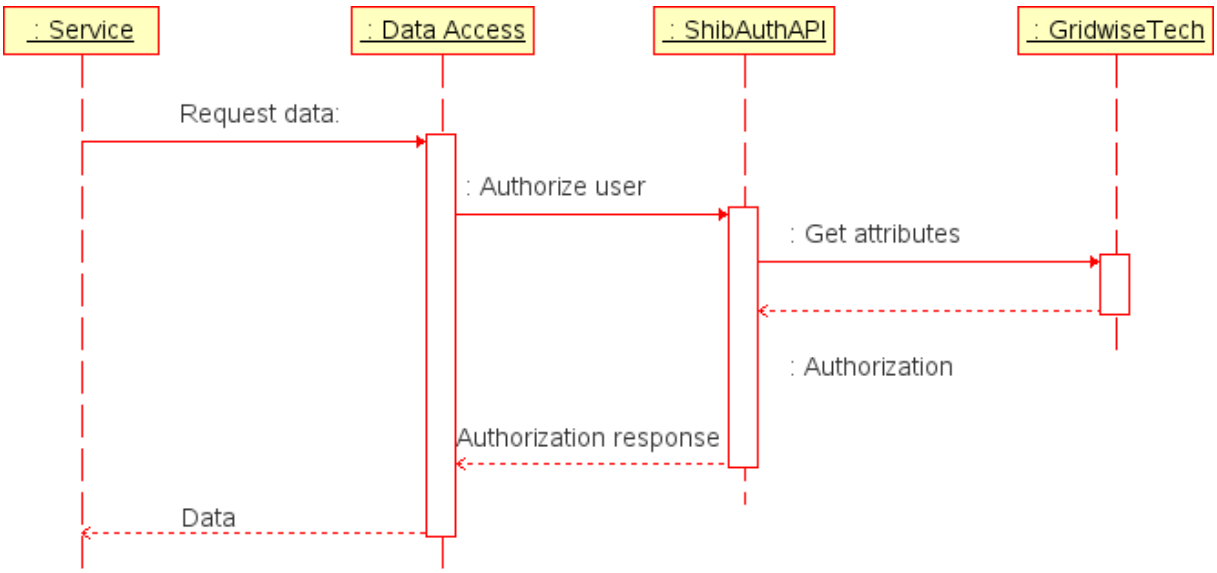


Figure 13 Detailed sequence diagram for the data retrieval

Scenario 2: Development and execution of an experiment

The ViroLab Virtual laboratory provides advanced tools for experiment developers, who are scientific programmers collaborating with domain researchers on preparation and execution of experiments. The Experiment Planning Environment provides and advanced user interface for experiment development, sharing with the use of Experiment Repository and execution using GSEngine on the resources accessible via Middleware layer.

In this scenario, we demonstrate how these subsystems of ViroLab VO work together in a secure way, using Shibboleth-based security infrastructure.

The scenario has the following steps:

1. User launches the EPE and opens an experiment from a local file (Figure 15).
2. User tries to execute the experiment using GSEngine. Since no authentication was performed yet, the user gets a warning message and execution cannot proceed (Figure 16).
3. User authenticates using Authentication plugin (Figure 17).
4. User fetches the experiment from experiment repository (Figure 18 and Figure 19), access is provided using Shibboleth handle obtained earlier from authentication plugin.
5. The experiment can now be executed (Figure 20).

The schematic diagram of the scenario is shown in Figure 14 and the details of the steps are described in the subsequent paragraphs.

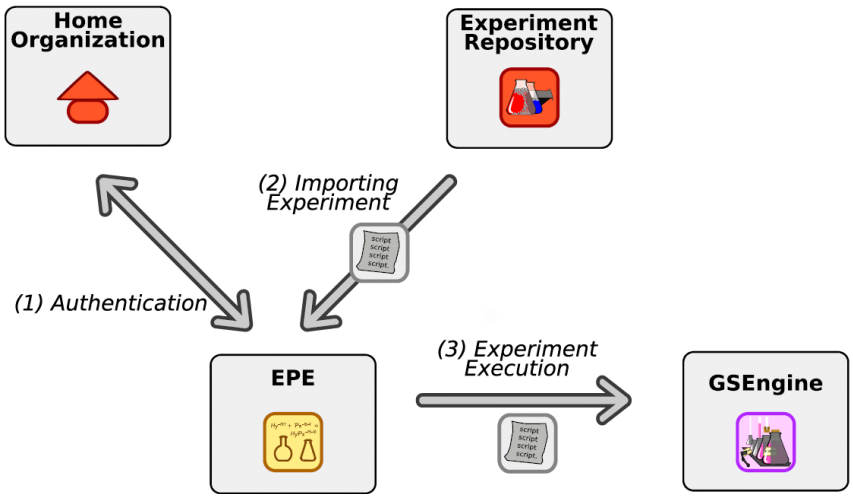


Figure 14 Scenario of the demonstration involving EPE

First, the user can open EPE, where it is possible to open and edit experiment scripts, saving and loading them from the local files. The Figure 15 shows a sample experiment opened in a script editor: the tutorial-like script demonstrates how to access Web services using GridObject library provided by GridSpace engine.

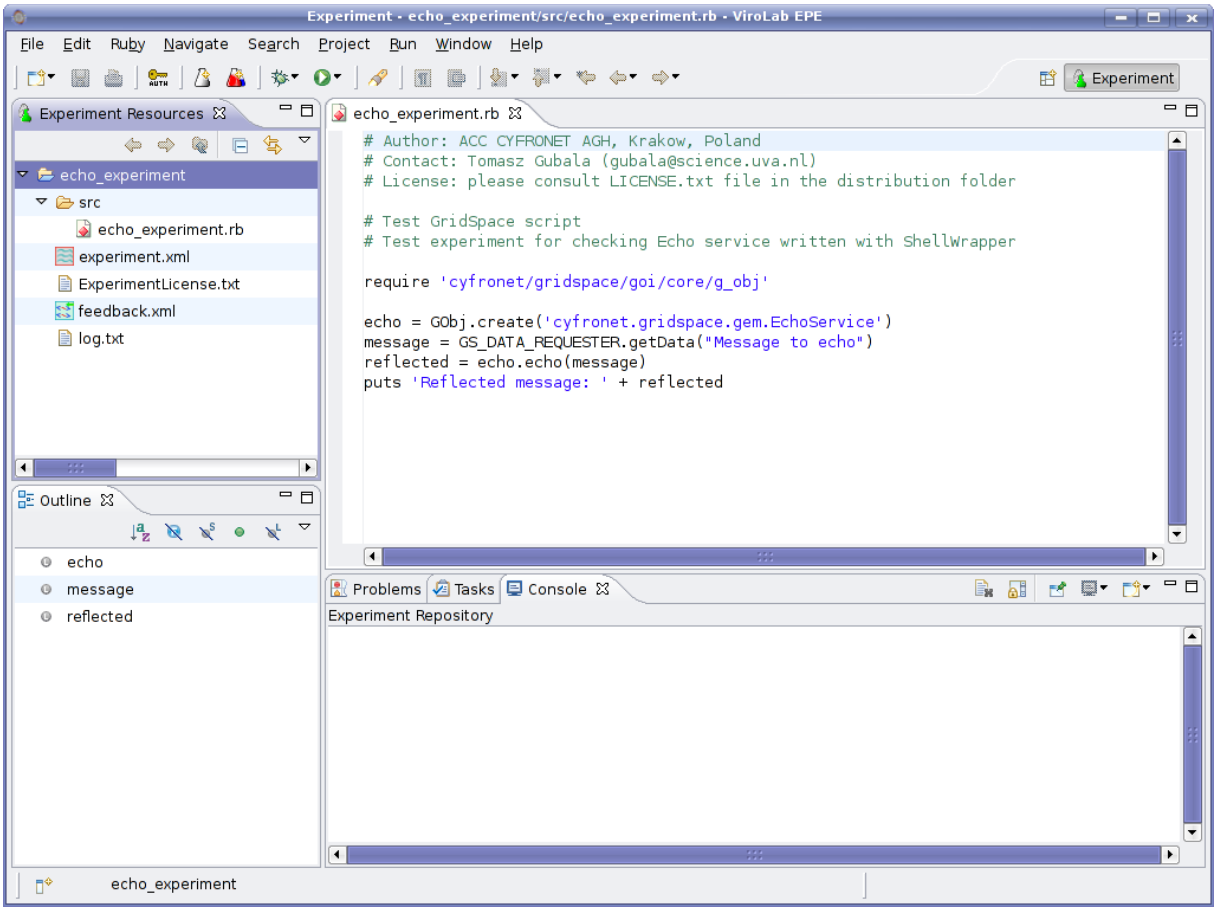


Figure 15 Opening an experiment in EPE from a local file.

When the user clicks on the "Execute experiment" button, the error message is shown (Figure 16). This means that it is not possible to execute experiments via GSEngine, if no proper authentication with the Shibboleth infrastructure was performed.

Figure 16 Error message when no authentication was performed

In order to execute the experiments, the user needs to authenticate using the Shibboleth authentication plugin, initiated by clicking on the "AUTH" button. The login dialog appears (Figure 17), where user has to select the

Home Organization (WAYF – Where Are You From), and provide the login and password to the selected Identity Provider. The password can be optionally saved for convenience. After successful login, the security token (Shibboleth handle) is stored in the EPE and from now on it can be used to access protected resources.

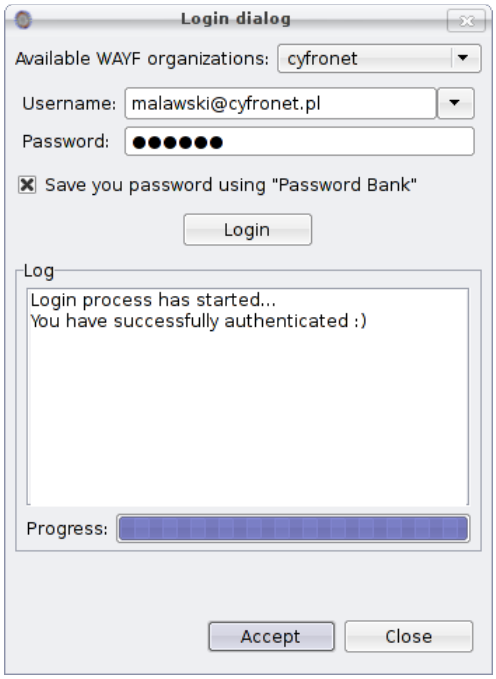


Figure 17 Shibboleth authentication plugin in EPE

One of the resources of ViroLab VO protected by Shibboleth-based security system is the Experiment Repository. It allows sharing the experiment scripts between the users, for both development and publishing purposes: the released experiments are available through the portal.
When launching experiment import wizard (Figure 18), the security credentials are already present, since the user was successfully authenticated before. Therefore the user does not have to insert any additional security information; the authentication is performed automatically by EPE using the security token obtained before.

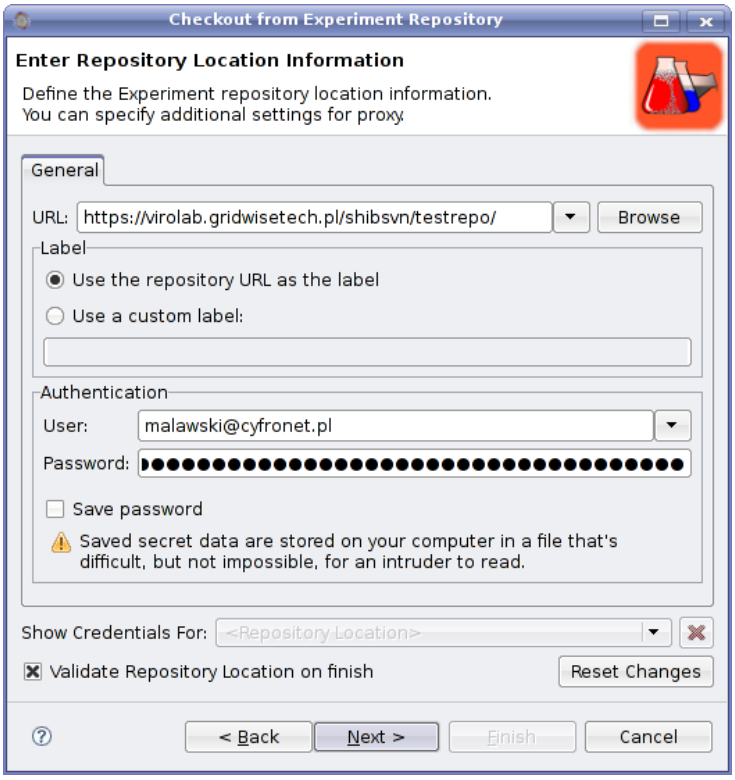


Figure 18 Importing experiment from Experiment Repository

After fetching the experiment from the repository, the user can work on it in the EPE. In Figure 19 we can see a sample experiment which involves comparison of several classification algorithms available from Weka library, wrapped as MOCCA [MOCCA] components.

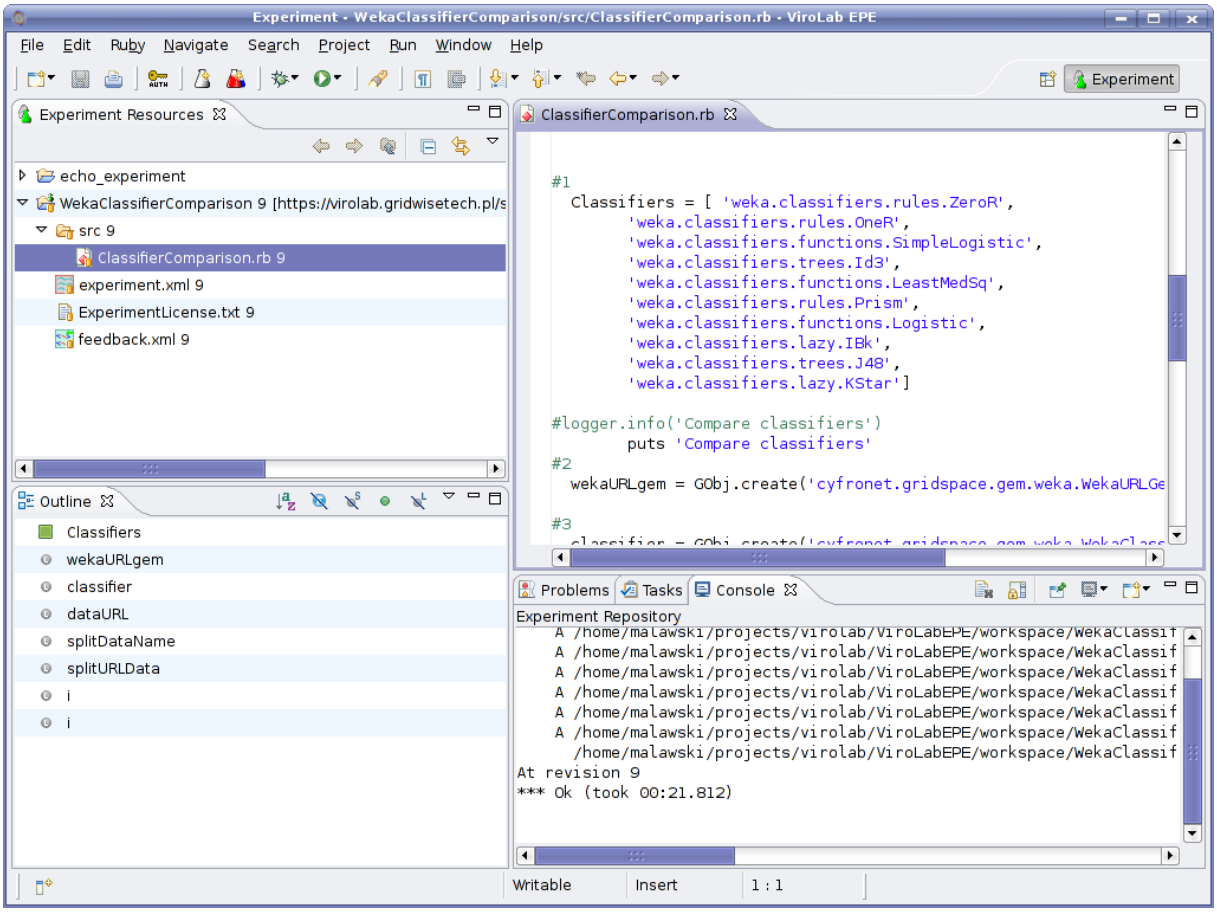


Figure 19 Experiment imported from Shibboleth-protected repository

The EPE enables to execute the experiments directly using GSEngine. This time when the user clicks the "Execute" button, there is no security warning, since the authentication was already done. Therefore the EPE can contact the GSEngine and the execution on the computing infrastructure can proceed using the appropriate middleware. The user can watch the output directly in the EPE (Figure 20).

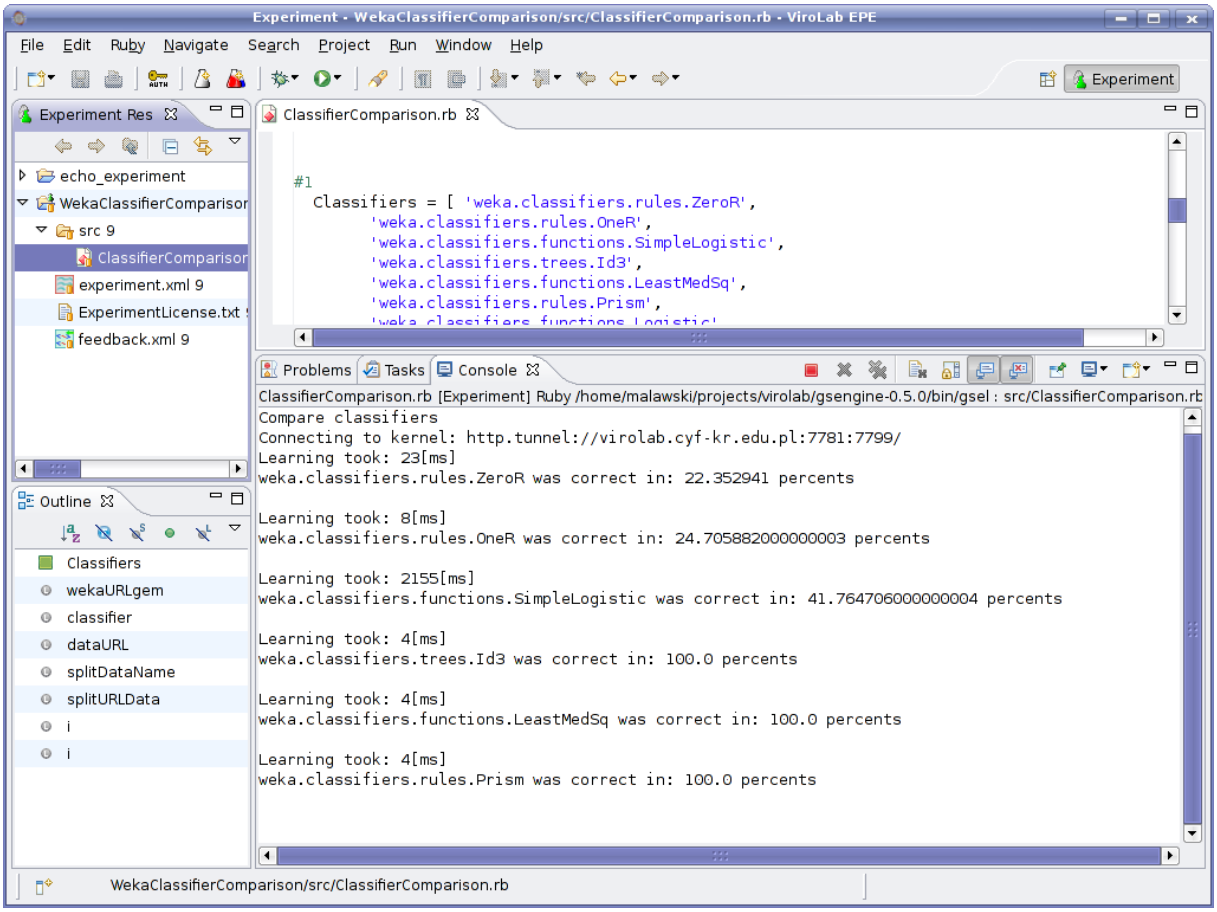


Figure 20 Execution of ClassifierComparison experiment using GSEngine

Demonstration - Summary

The scenarios presented in this section include examples of non-trivial behavior of the ViroLab. In order to make the operations invisible for users, providing the demanded security level at the same time, a significant amount of work had to be performed.

A very important achievement of WP2 is that very complex operations of authentication, authorization, access to resources are hidden for an ordinary user who is only interested in the application level.

All the modules which were referenced in this description are described in detail in the following sections.

3 Presentation Layer

ViroLab provides several useful functionalities for clinicians, virologists and scientists. In order to make them use ViroLab, it is very important to create simple, user-friendly and functional presentation tools [EPEMI].

The presentation layer consists of two main user tools: portal for accessing all applications by experiment users or clinical virologists and Experiment Planning Environment (EPE), which is a developer tool for creating the experiments. The Portal is the main entry to all functionalities provided by Virtual Laboratory (VL) (e.g. retrieving virus genotypic information from remote Dbs or finding out the resistance of the virus based on HIV nucleotide). The EPE is the external application based on Eclipse, which provides flexible and convenient functionalities for experiments creation and edition. Implementation and functionality details of this means of VL access are described in the following sections.

The diagram in Figure 21 presents the integration points of Portal and EPE. There are two modules (GSEngine and Experiment Repository (ExpRepo) which are integrated with Portal as well as with EPE. The integration of other elements is specific to Portal and EPE. DAS (Data Access Services) and Provenance are the elements integrated with Portal and specific applications in there. DOS (Domain Ontology Store) and GRR (Grid Resource Registry) are the integrated modules for EPE. All Home Organizations are integrated with the portal to provide possibility for federated authentication.

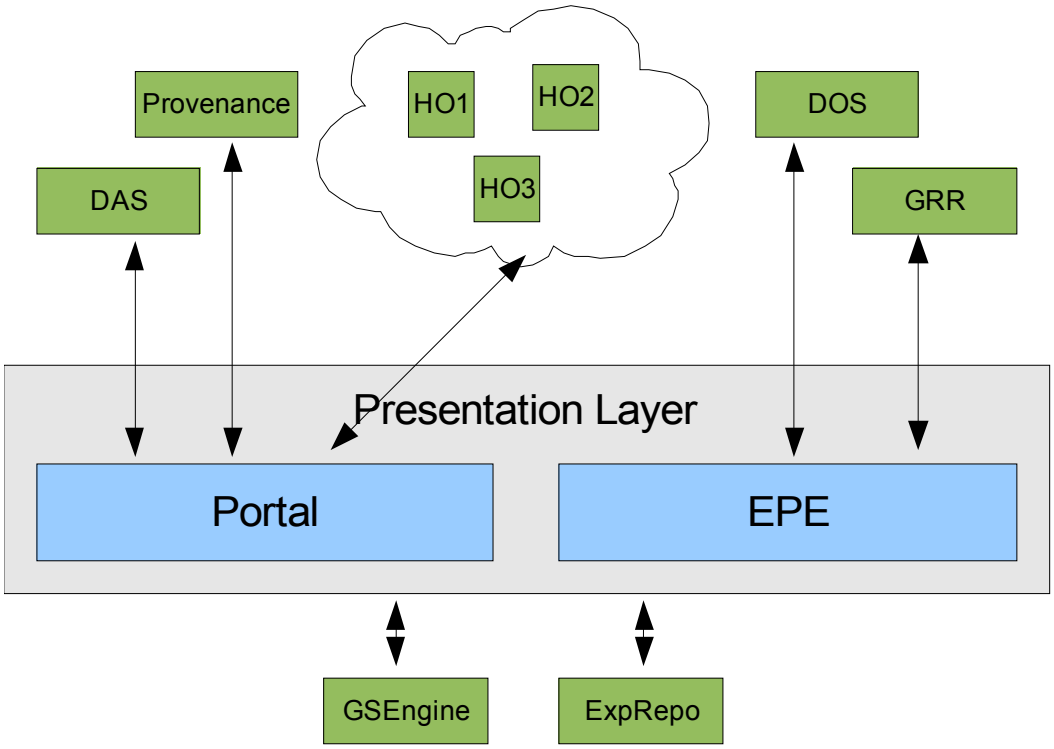


Figure 21: Presentation Layer, and integration point with other elements

Experiment Planning Environment

In the ViroLab project the users of the virtual laboratory are divided, based on different expectations, into two main groups (roles): experiments developers and experiments users. Thus it is crucial to provide both classes with dedicated tools that address their specific needs. In case of the experiment developers Experiment Planning Environment (EPE) is such a tool. Its main goal is to support the whole process of building experiments, from creating new experiments with a default structure, through the development stage up to publishing them into the Experiment Repository, since experiment users are enabled to run them through the ViroLab portal.

In addition to technical aspects of creating new experiments, it is important to remember about relationships between the experiment developer and the experiment user. For that purpose the EPE contains facilities for exchanging feedback information related to a certain experiment, e.g. any defects found.

Security issue has to be taken into consideration, especially when many different organizations do not necessarily want to share their resources with everybody (e.g. data on patients). Being integrated with the Shibboleth security middleware, the EPE provides a user-friendly wizard which eases the authentication process.

EPE Features in Version 1

From the architectural point of view the EPE is divided into small pluggable components called plugins which have individual responsibility. However separation based on plugins is too fine-grained for easy management. Therefore they are gathered into groups, called features, of logically related plugins in order to solve a certain problem.

The current version of the EPE consists of three main modules:

a) The *core* which is used to manage plugins from the environment. From the user's point of view the most usable feature is the update manager (Figure 22). It allows downloading new plugins which will extend the environment with new functionalities, or download new versions of the currently used plugins. The authentication plugin (Figure 23) is also part of the core feature. To authenticate the developer has to select the WAYF organization and provide credentials information such as *login* and *password*. By clicking the "Login" button the authentication process will be performed using the Shibboleth middleware. After successful authentication it will be possible to connect to the Experiment Repository and to use the resources owned by the selected organization.

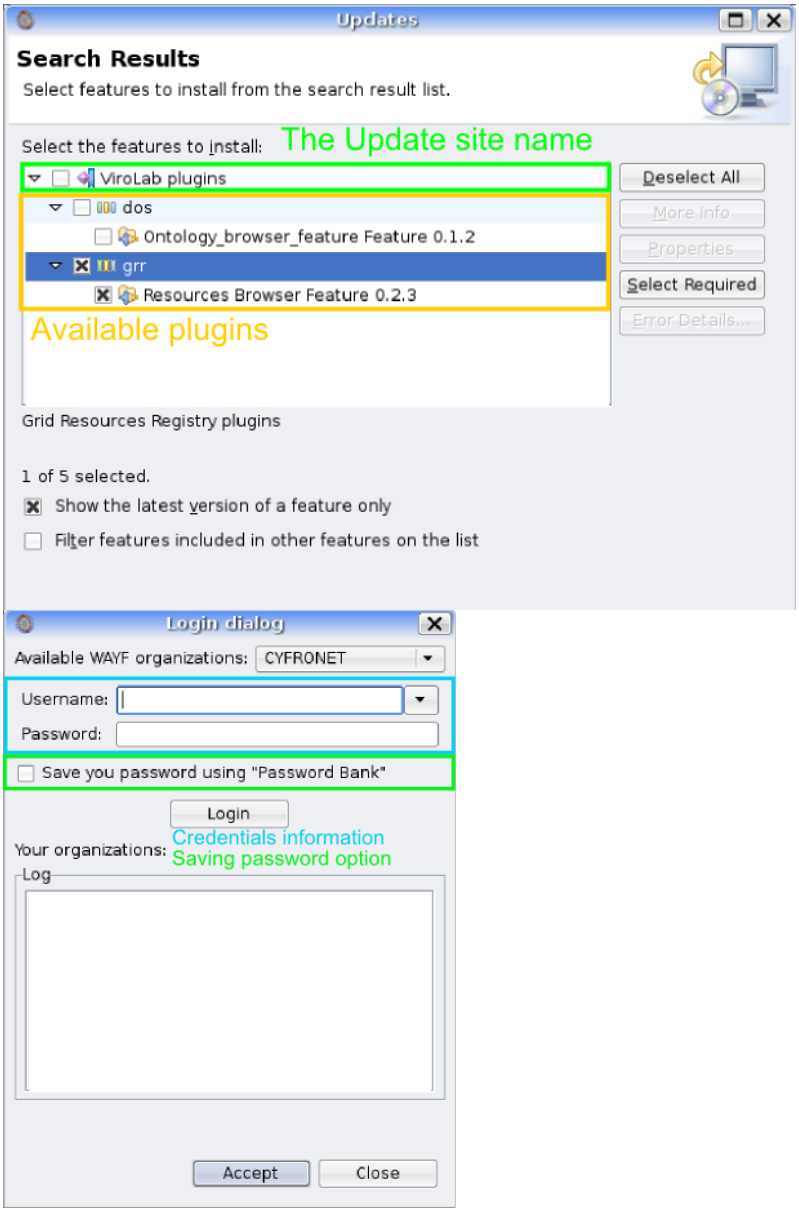


Figure 22 (on the left): Update manager with available ViroLab EPE plugin
Figure 23 (on the right): Authentication dialog

b) The GScript Development Tools feature that provides facilities for writing easily new experiments. To plan a new experiment the developer uses a special script language (GScript) based on the JRuby programming language. Therefore EPE contains a powerful script editor with syntax highlighting and code assistance features (Figure 24). Besides the editor the experiment developer can use an outline view for locating various script objects such as variables, classes or methods. After having developed a part of script it is possible to execute it in order to test it and observe how it works using GSEngine [see the "Integration points" section for more details].

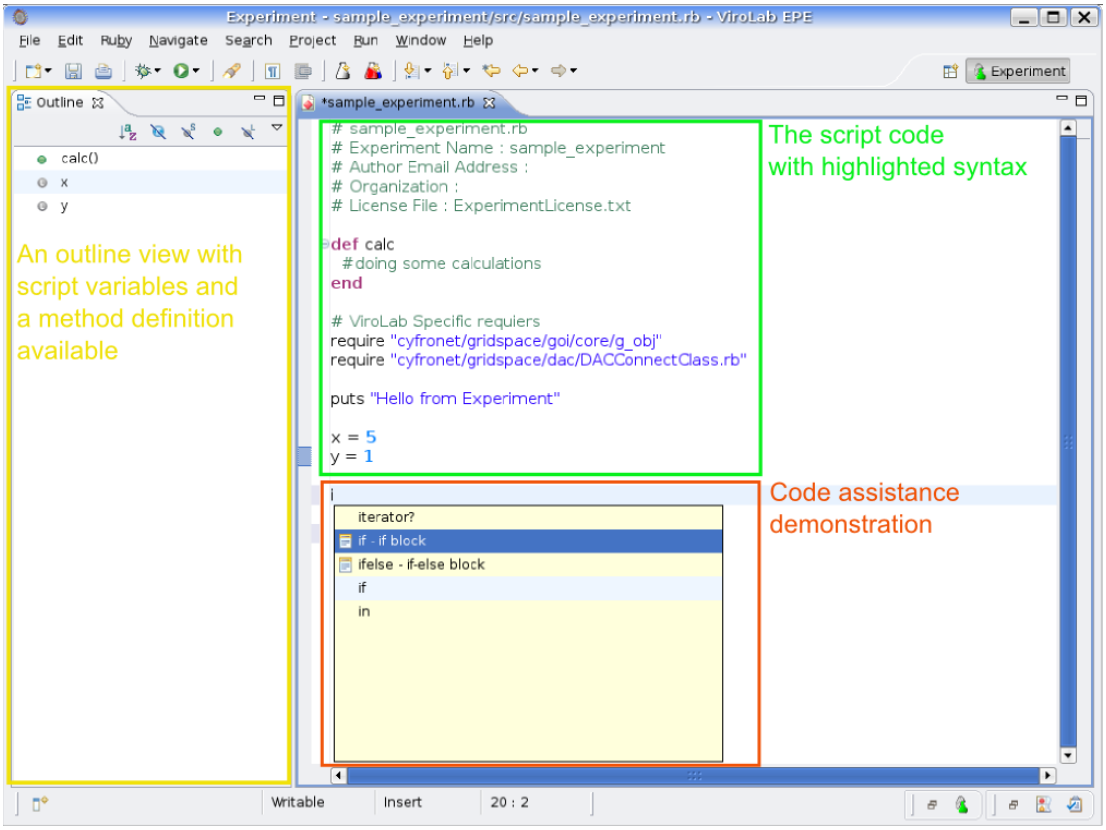


Figure 24: Script editor and an outline view

c) The Experiment Repository feature contains plugins which are responsible for performing operations on the repository. It is a common situation when a project (in the Virtual Laboratory it is an experiment) is built by a team of developers. Therefore it is crucial to share the code between team members in a standard, user-friendly way. This feature addresses the issue by providing facilities for:

- creating a connection (Figure 25) to the repository,
- import (Figure 26) and export experiments,
- releasing new versions of the experiment.

Also exchanging information between the experiment developer and the user is performed through the repository, so it should be transparent to the developer.

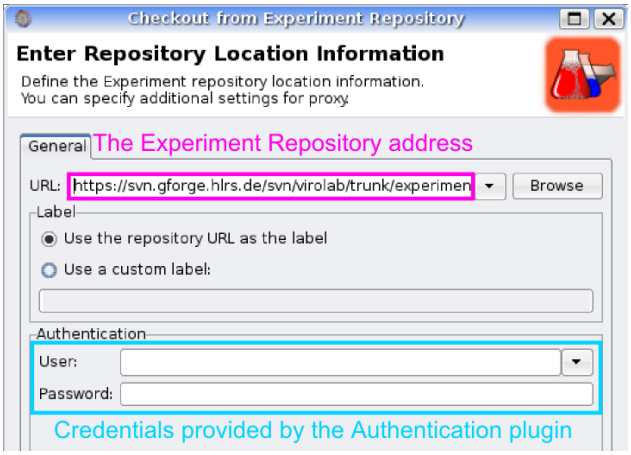


Figure 25 (on the left): Creating a new repository connection
Figure 26 (on the right): Selecting an experiment from the repository to import

EPE Integration Points

As already mentioned the EPE is part of the ViroLab presentation layer. One of its goals is to provide user-friendly interfaces for components that are helpful in the experiment development process but lay in tiers beneath the presentation one. In addition these are often command line tools with complex usage syntax therefore using them can be difficult especially for inexperienced developers. For some of such components the EPE provides 'built in' integration points. In order to use them the developer has to provide some basic information such as location (see GSEngine subsection). Another possibility to integrate EPE with external components is downloading, through EPE update manager, dedicated plugins which will provide Graphical User Interface for a certain component (see the DOS plugin or GRR browser section). Below there is an available list of the most important integration points in the EPE.

Integration of EPE with GSEngine

EPE uses GSEngine [Ciepiela07] as a local script interpreter. The `java.lang` package is the current way how EPE calls GridSpace Engine Command Line Tools. The operating system process which represents a call of the `'gsel'` or `'gsec'` executable, is wrapped into the java Process object. To make this more user-friendly the standard streams from the java process are redirected to the built-in EPE console.

From the user point of view, the only thing to be done (besides installing the GSEngine correctly) is to provide in the EPE information about GSEngine executables location. The user can have many GScript language interpreters reported and use different interpreters for different situations. After deciding on which GScript interpreter should be used, the user can execute experiments with the "Run as experiment" operation. The whole procedure is presented in Figure 27.

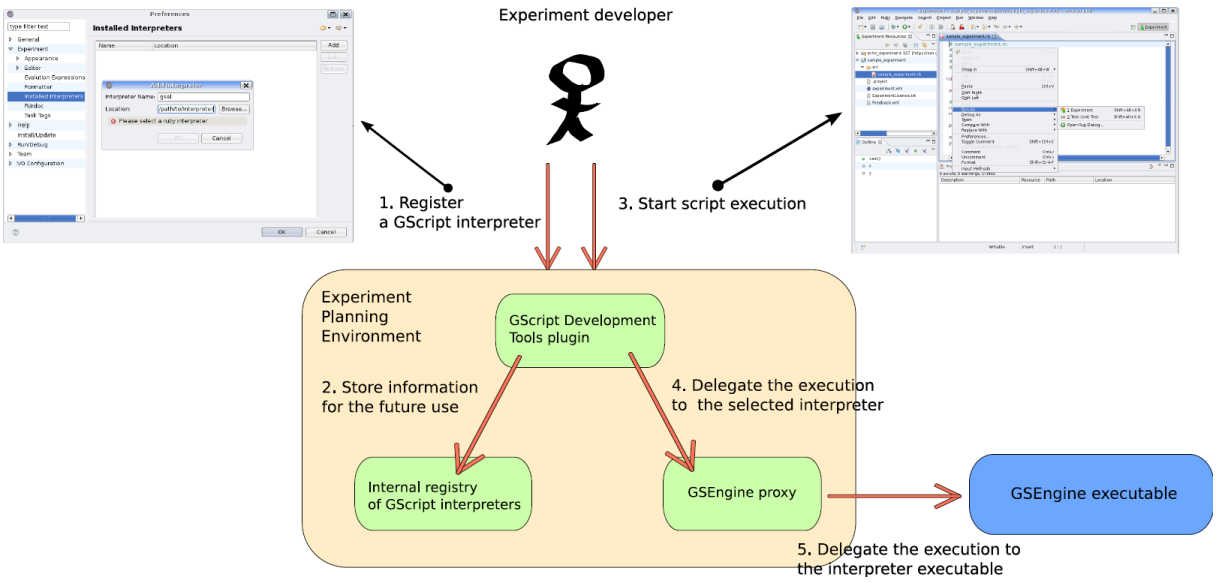


Figure 27: The procedure of adding the GSEngine to the EPE

The next step in the integration procedure between the EPE and the GSEngine will be to provide the GSEngine client plugin for the EPE. Its main goal is to relieve the user from the necessity to provide information about the GScript interpreter prior to its use.

Integration of EPE with Domain Ontology Store (DOS)

The main purpose of the Domain Ontology Store (DOS) is to maintain the common model of the application domain of the ViroLab Virtual Laboratory. The semantic models of stored inside DOS form a natural integration mechanism for modules, services and tools that build the virtual laboratory. The models are divided into submodels taken into consideration both generality and purpose. The domain knowledge is modelled and represented with ontologies and DOS contains these ontological models in a non-volatile storage and provides both read and write access. For the detailed design of DOS, see Section 6.1.3 of [D3.2]. The realization of DOS inside ViroLab is deployed in a Sesame OpenRDF [SESAME] server and the models are stored in a persistent database inside a MySQL DBMS system.

One of the main recipients of the content stored inside DOS is the Ontology Browser Plugin (sometimes called OntoBrow for short). It is a tool deployed inside the Experiment Planning Environment. This plugin is provided for the developers of experiment to help them find appropriate external services and data elements (that could form newly developed experiments) by task-oriented semantic content search (see Figure 28). Please note a slight update of the use cases diagram with regard to the Section 4.3.1 of [D3.3] mainly due to altered data search use case.

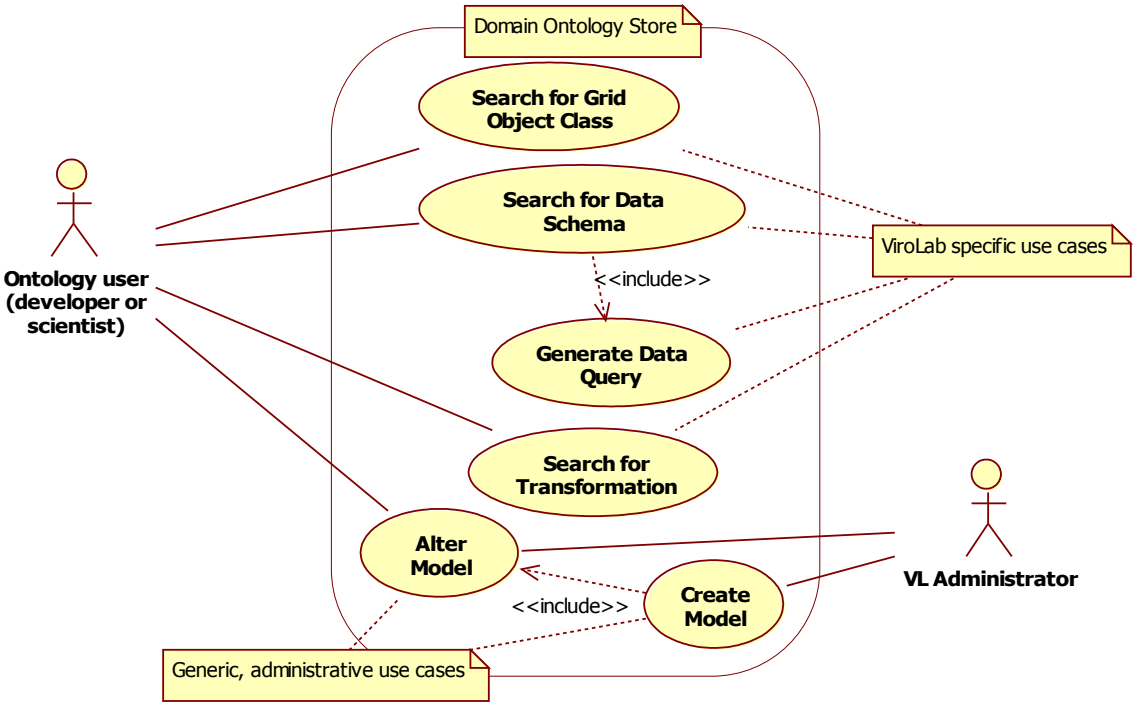


Figure 28: The updated DOS use cases

There are four main integration points for the OntoBrow Plugin:

- connection with DOS as the model taxonomies source
- integration with Resources Browser Plugin inside EPE for Grid Object lookup use case
- integration with the remote Data Access Client façade for data schema retrieval
- integration with the experiment editor within EPE for data query generation.

The paragraphs below explain the status of integration of all these points, staring with the oldest and the most stable ones and finishing with the new additions.

Ontology Browser and Domain Ontology Store are connected with an OpenRDF protocol implemented on top of the HTTP protocol stack. It supports several basic actions like retrieval of available models and querying of a specific model with a chosen querying language (three languages are supported out of which the Ontology Browser uses SeRQL [SeRQL] option as it well suits the submodel retrieval requirements of the plugin. The interesting parts of the model are returned, usually for ontologies, in a triple set form and are subsequently parsed into a graph-like structure on the client side. The structure is then displayed for the EPE user in a graphical widget. This link is implemented since the first prototype of the Virtual Laboratory (as stated in Section 4.3.2 of [D3.3]) and is being maintained (optimized, fixed where necessary) since then.

The Ontology Browser uses internal Eclipse RCP mechanisms (called extension points) for integration with another pieces of EPE, namely with the Resources browser Plugin and the current experiment code editor. The first integration mechanism was already present inside the first prototype of the system (see Section 4.3.2 of [D3.3]). Briefly, it provides two important use cases of the Ontology Browser, namely semantic Grid Object and transformation search (see Section 6.1.3.4 of [D3.2]). It allows the developer to find suitable operations available in remote computation services that would semantically fit the desired input, output or both input and output data. The integration with the experiment editor is explained below, together with the DAC facade link.

The integration with the Data Access Client facade is required to support another crucial Ontology Browser use case of the data schema search scenario (see Section 6.1.3.4 of [D3.2]). In the process of detailed design the scenario presented originally in the Design Deliverable was slightly altered to explicitly distinguish between data schema retrieval and data query creation phases (see Figure 29).

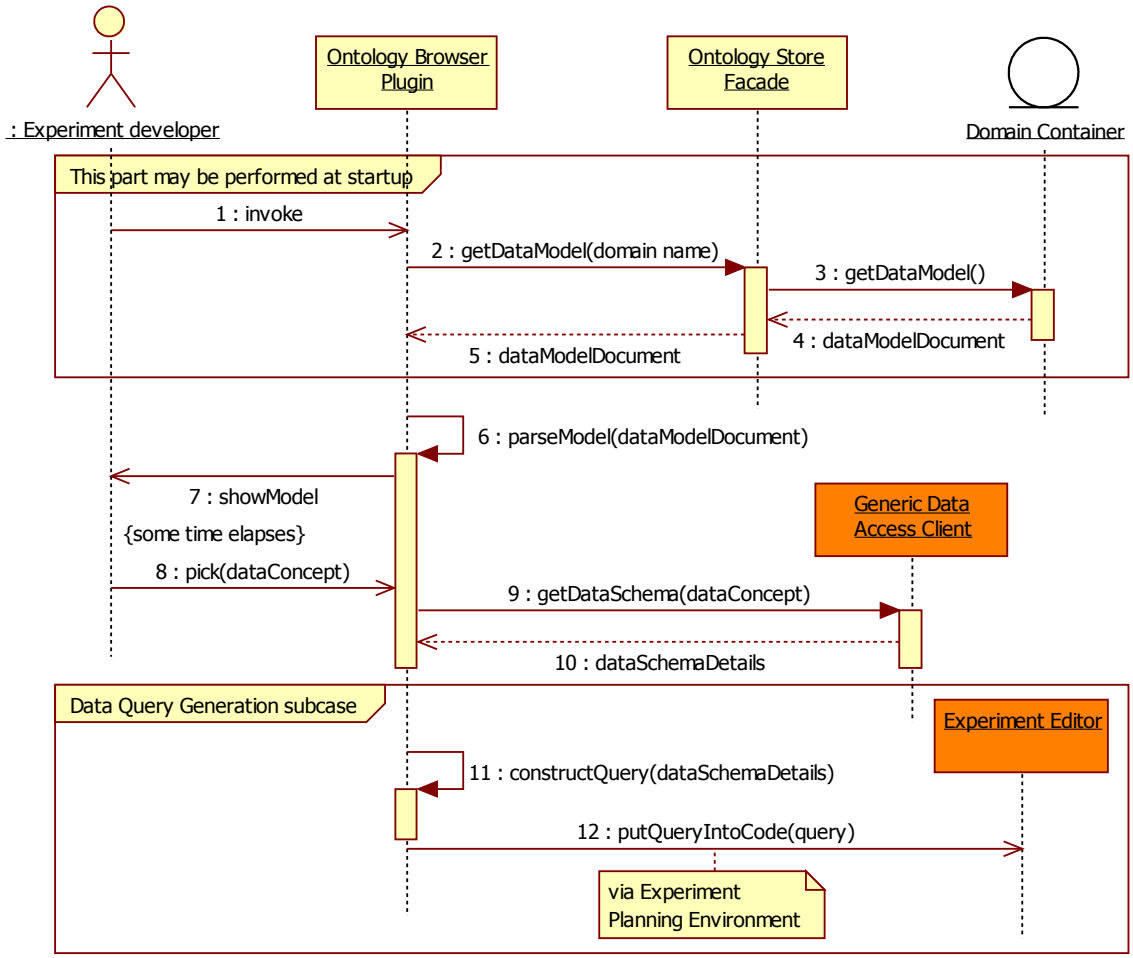


Figure 29: The updated and decomposed use case for data query generation.

Here one may clearly see the three stages of data query creation:

1. the user chooses a semantic concept of the modeled domain that represents the type of data one likes to analyze within an experiment (e.g. a *VirusNucleotideSequence*),
2. the plugin requests the remote DAC to find out how this concept is represented in data sources available to ViroLab experiment developer,
3. with the details obtained of the concept's representation a query that would retrieve the desired data (here, the sequences) is generated and inserted into the code.

To deliver the functionality mentioned in points 2 and 3 two different integration links had to be implemented. First, the Ontology Browser uses a dedicated DAC facade protocol to ask for the current data schema. The input is the mapping of semantic concept into a name in the data sources and the output lists what attributes and of what data type there are present in the available data sources.

Having that information at its disposal the plugin generates a proper query stub that would eventually ask (at the experiment runtime) for the required data. The stub is generic and may be further tuned by the programmer in the experiment code (for instance it is impossible for the tools to guess certain constraints of the query - the developer needs to define those by himself). In order to insert the newly-generated query stub into the code of experiment, the Ontology Browser uses the extension points defined in the GridSpace extension points API (see specific explanations in the Grid Resources Registry subsection 3). The API allows to find the present experiment editor and to inject some source code in the current position.

Integration of EPE with Experiment Repository

As mentioned above, the plugins related to the Experiment Repository constitute one of the core EPE features. The main reason for this is the importance of relationships between the EPE, the Experiment Repository and the ViroLab portal. It can be mapped onto the relationships between an experiment developer and experiment user, as shown in Figure 30. In order to make an experiment visible for the scientists within the portal, the experiment developer has to make a release. It is performed with a user-friendly wizard, which needs from the EPE only a version name of the release (optionally a comment can be provided) (Figure 31).

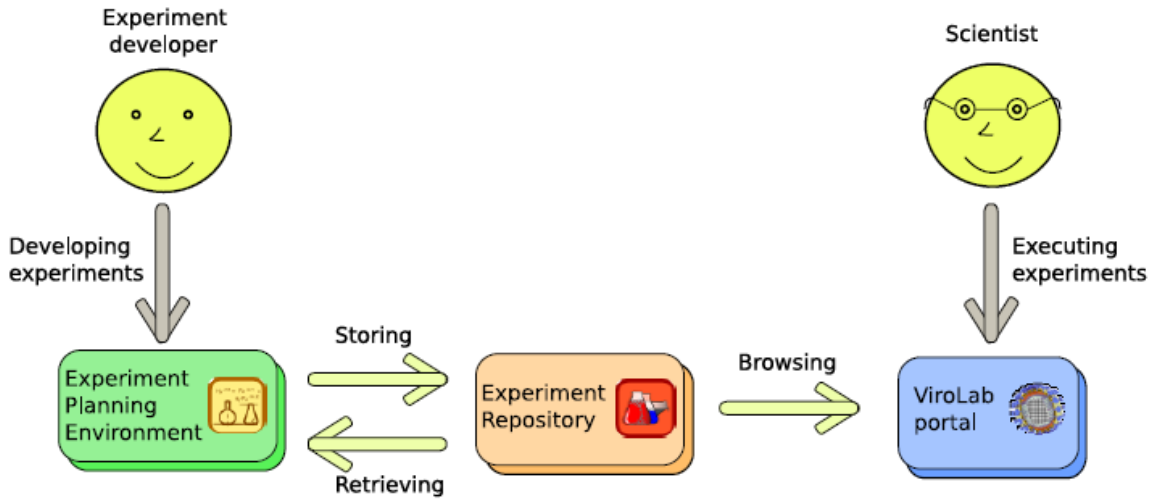


Figure 30: Relationship between the EPE and the ViroLab portal

Sharing an experiment and working on the same project by a group of developers is done in the similar manner as with the SVN repository. First, the experiment has to be exported to repository and imported by each member of the group. This is a mandatory procedure, otherwise a connection with the repository will not be established and no changes will be enabled to be made to the repository. After this stage a copy of the experiment exists locally on each developer machine. Developers can now submit changes to the repository with the `commit` operation or retrieve any changes from the repository with the `update` operation. When the experiment comes to a stage when it is stable enough for using, it can be published as described above.

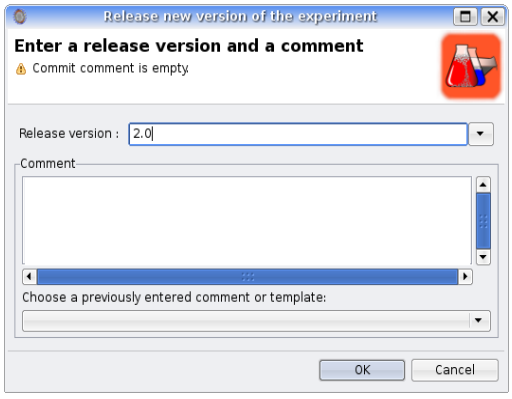


Figure 31: New release dialog

Integration of EPE with Grid Resources Registry (GRR)

Grid Resources Registry stores information about all computational resources that are available in the virtual laboratory. It helps the developer to create an experiment script by hiding the complexity of used technology. The registry is integrated with Experiment Planning

Environment and Ontology Browser plugin thanks to Resources Browser plugin (see Figure 32).

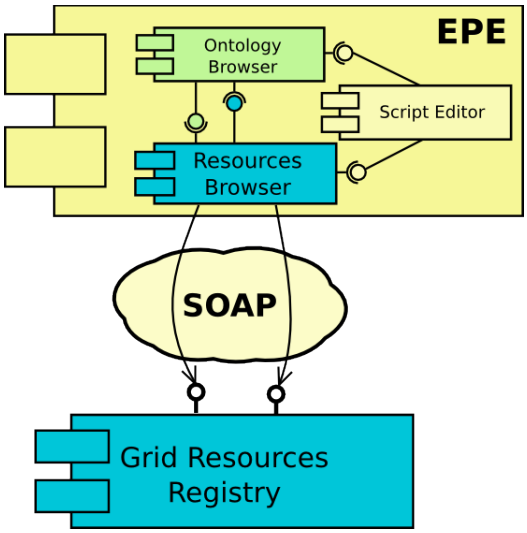


Figure 32: GRR integration points

Resources Browser plug-in is a user interface for the registry. It allows browse and manage the whole content of the registry: packages, Grid Objects, its methods, documentation, implementations, and instances (for more information and a tutorial about Resources plugin please see manuals attached to the [D3.3] and [VIROLAB-VL]). It is integrated with other components through Eclipse extension points:

- **epe.code.inserted** defined by the `cyfronet.gridspace.api.epe.ICodeInserter` interface: it is implemented by the EPE experiment script editor. It allows to insert text into a currently edited experiment file which means that the user can insert a code line responsible for creating a new Grid Object by simply double clicking on the Grid Object in Resources Browser (see figure Figure 33).
- **grrbrowser.contextmenu.items** defined by the `cyfronet.gridspace.api.grr.ICtxMenuItemProvider`: during the runtime when the user performs a right click on the object presented in the Resources Browser all implementations of this extension point is found and asked if there are any additional menu items that should be added to context menu. Thanks to this extension point, the Ontology Browser plugin is able to add additional items to the menu that allow to search for a concrete parameter meaning in a particular domain (see Figure 34).
- **grrbrowser.search** defined by the `cyfronet.gridspace.api.grr.ISearch` interface: it is implemented by the Resources Browser plugin. As a result, it allows other plugs-in to search for and show a specified content from the

registry. This functionality is used by the Ontology Browser plugin (see Figure 35).

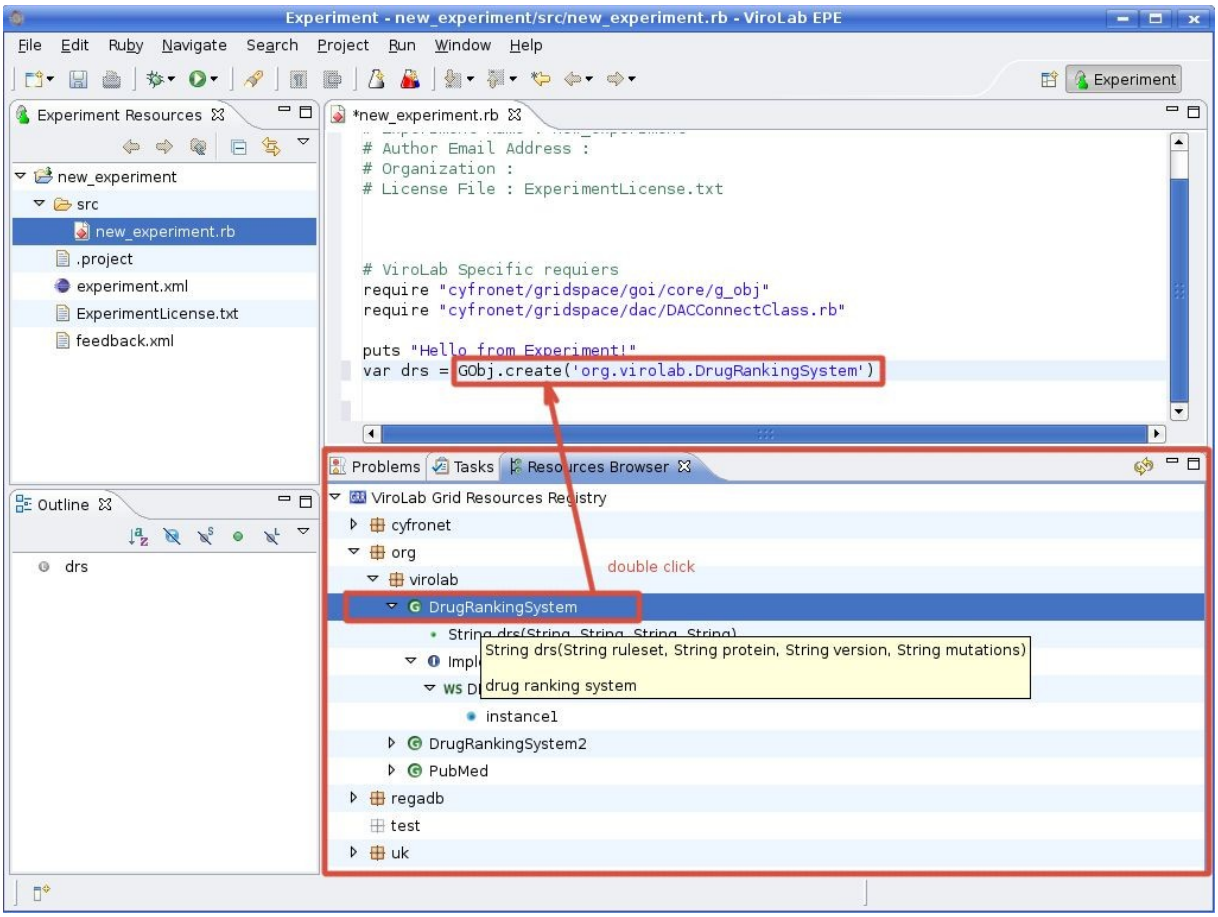


Figure 33: Integration between GRR and EPE – inserting a code line to the editor

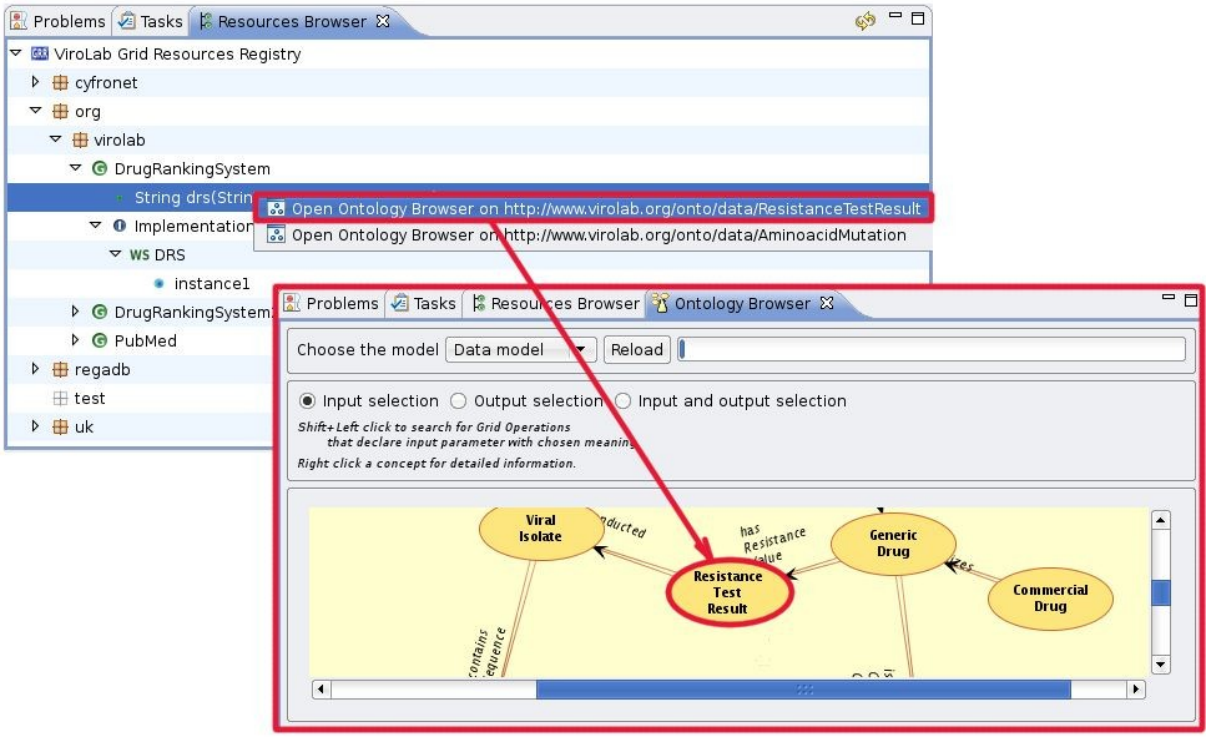


Figure 34: Integration between GRR browser and Ontology browser – semantic search

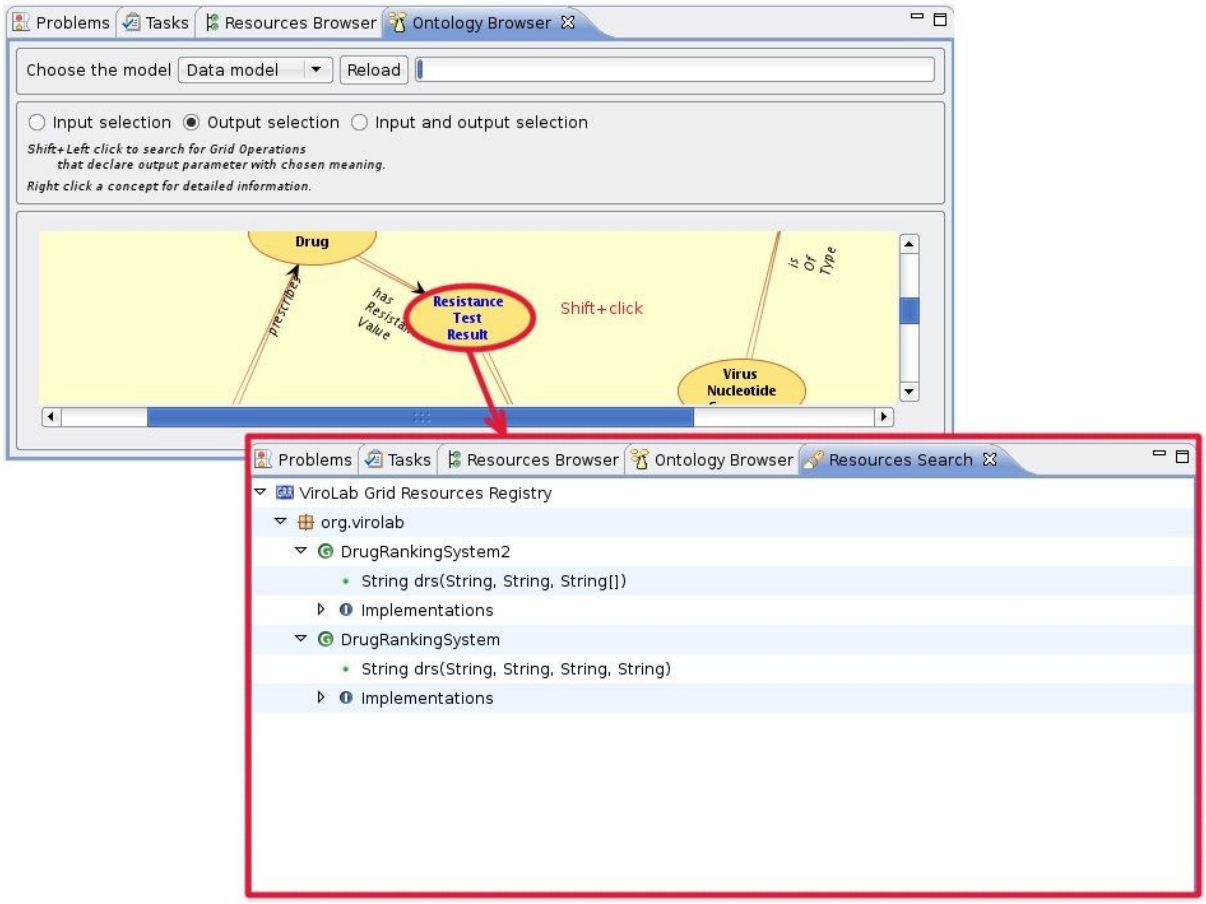


Figure 35: Integration between Ontology Browser and GRR browser – searching registry contents with concrete meaning

Planned Functionalities in EPE

While already providing many useful facilities that support the experiment development process, the EPE is still in its prototype phase. Therefore there is a lot of work to be done in order to complete the environment. The most important tasks are:

a) code assistance – due to being one of the most helpful features of every powerful programming language editor, it should be done with care of every detail. In the current version of the EPE, the code assistance feature is already available however only in a basic form (e.g. language keywords and control structures). In the final version, provided suggestions will also encompass Grid Objects and their methods available from within the Grid Resource Registry.

b) integration with the GSEngine - although it is already possible to use the GSEngine for executing experiments, to make it even more simple, the GSEngine plugin should be integrated into the EPE. Currently, in order to use the GSEngine (either in local or remote mode) the developer needs to point the location of the GSEngine executables. The plugin will enable to avoid even this step by setting-up the default interpreter to the integrated one.

c) improved collaboration between the experiment developer and user - the information exchange process between the EPE and the portal is performed in the well known manner of submitting tasks, to a certain experiment by the experiment users, on the one side and by answering them on the other side. It will be done through a dedicated editor inside the EPE that will present information about available tasks in a user-friendly way.

ViroLab Portal

Portal implementation for ViroLab is based on a popular open-source portlet-based web portal framework - GridSphere. The idea of the portal is to provide a single sign-on (SSO) access to the underlying services. The end user should be able to achieve as much as possible in terms of operating applications, without the knowledge of complex, distributed technologies which underpins the system. The main advantage of GridSphere is the fact that it provides a simple interface for developing new applications, which can be deployed and administered within the portlet container. Every portlet developed for the portal has consistent look-and-feel, so it is relatively easy to learn and use new portlets on the fly, even for a non-experienced user.

The diagram in Figure 36 presents the current architecture and functionality of the portal and integrated elements. As it can be noticed,

the portal is responsible for interactions with the user, access to all services and data visualization. All computations are distributed to resource providers. The access to services is done by passing requests with credentials to underlying services.

The user accesses the portal using his/her web browser. In order to start work, the user is redirected to his/her Home Organization providing authentication (step 3 in the "finding out the resistance" scenario). The authenticated users can take advantage of all the applications provided by the Portal. The Portal is responsible for redirection to the proper Home Organization, display of the application portlets, and then requesting the chosen resources.

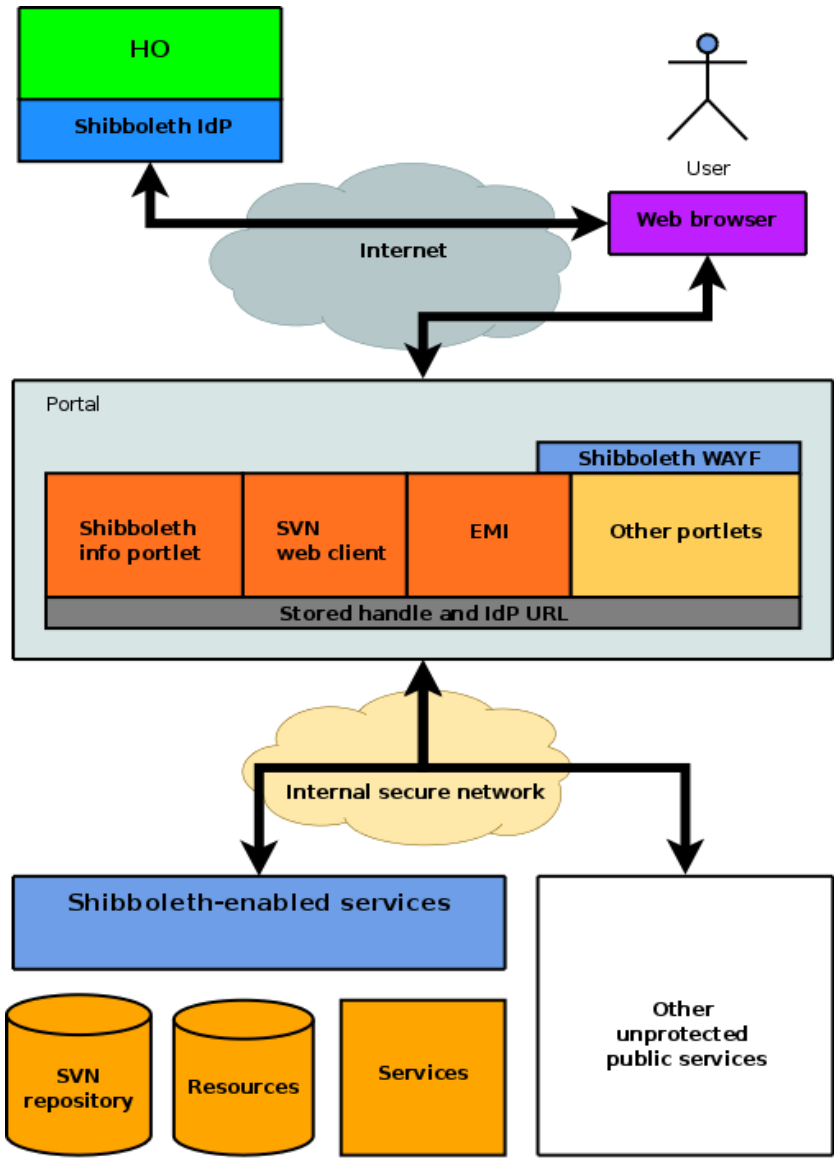


Figure 36 Portal architecture

Portal Features in Version 1

The basis for the ViroLab portal is GridSphere. To provide full functionalities required in the project this framework had to be expanded by several extensions. The need for extensions results from the security requirements as well as needs of GUI behavior. In the version 1, the portal is equipped with the following features:

1. In order to start working with ViroLab portal, the user needs to be authenticated. Authentication is done at the level of the user's Home Organization (HO). There was a need to make it possible to choose the Home Organization to which the user belongs. Such a module is named "Where Are You From" (WAYF). The new portlet designated for that task was created. It is called "Shibboleth Login" and is an acting WAYF function. When the user wants to log in to the portal, he/she has to select his/her HO from the list and then he/she is redirected to his/her HO authentication page [step 2 of the scenario].
2. After having chosen the Home Organization, the portal should redirect them to proper HO Authentication module, and when a positive authentication recognizes the user, he/she is allowed to use the portal applications. In order to perform that, GridSphere was integrated with Shibboleth module ShibGS [step 3 of the scenario].
3. After the successful authentication the user is recognized by a *handle* and IdP name (IdP is other name for Home Organization and stands for Identity Provider). The portal is responsible to keep this data and pass it to requested resources. There is a portlet which shows this information (some resources need to be accessed without using the portal; if they need to be secured, the data from this portlet can be used in order to perform authorization)
4. ViroLab version of the Portal has been integrated with Google Web Toolkit AJAX API. This toolkit provides a simplified dynamic web application creation and decent look of the portlets.
5. In order to make the portal available from within mobile devices such as PDA or cell phone, the portal has been extensively tested on these platforms and required modifications have been made. These implied several configuration changes in order to make the portal adjust to other devices.

Portal Integration Points

WAYF and IdP integration

The WAYF ("Where Are You From") module provides the possibility to choose the Home Organization which the user belongs to. To establish this, integration with several Home Organizations was made. At the moment of creating this document the following organizations were integrated: Cyfronet, HLRS, and GridwiseTech. This integration provides users from within these institutions with their own authentication systems in order to start working with Portal.

Integration of Portal with Provenance (QUaTRO)

Query Translation Tools – QUaTRO, also referred to as the 'provenance GUI' is a tool designed for enabling end user-friendly construction of queries over both data and provenance repositories. In fact, QUaTRO can use any repository for which a semantic description of its data model is provided as in the case of ViroLab's data repositories available through Data Access Service and the provenance repository available through PROToS.

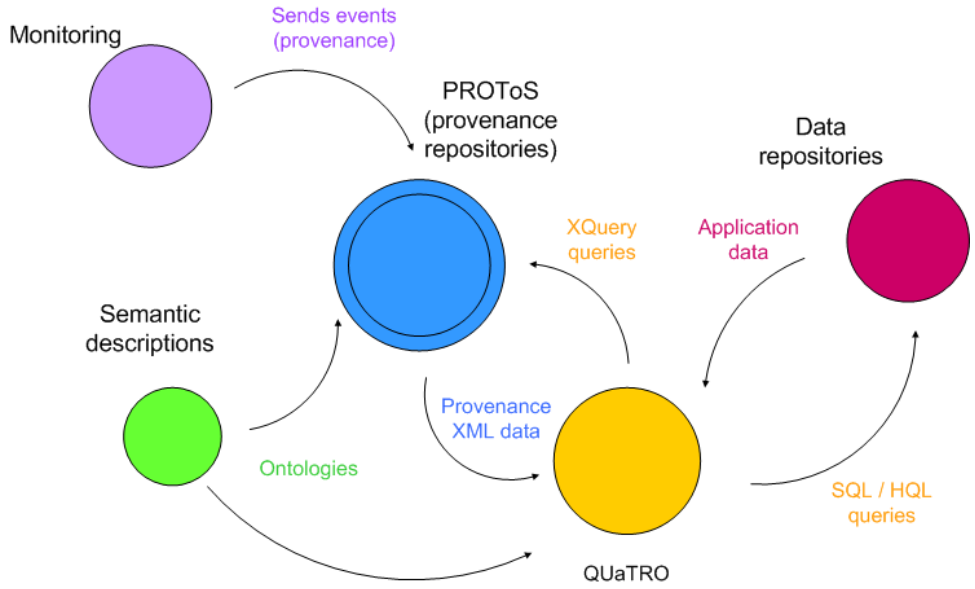


Figure 37 QUaTRO and context architecture

Figure 37 depicts the QUaTRO tool and all external components it interacts with. These are:

- PROToS, which serves as a provenance repository in our Virtual Laboratory. Communication between QUaTRO and PROToS is realized by stateless web services. QUaTRO, using the remote interface, sends queries to PROToS which are XQuery-based queries encapsulated in a *Query-class* instance. Query results are represented in XML and retrieved from a *QueryResult-class* instance.

- Data repositories – currently only relational data bases, whose data models are described as ontological semantic mappings, are supported. These mappings describe how the ontology data model is mapped into particular relational database schemas. For the sake of integration and testing we have set up an SQL database at CYFRONET together with suitable semantic mappings. Below we present part of the mapping ontology.

  The QUaTRO architecture was designed and implemented for quick integration with new data sources in mind. The logic responsible for accessing data has been moved to a separate component, named *quatro-dac*, built around a factory of data access clients. Each client could implement a logic for accessing a separate type of sources, but the differences are hidden other QUaTRO components, so that adding a new client has no impact on component's code. Currently, the implemented SQL client is based around a standard Java JDBC library.

```
<rdf:RDF
  xml:base="http://www.virolab.org/onto/mapping/cyfronet_mysql/">
  <owl:Ontology rdf:about="">
    <rdfs:comment
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Mapping of ViroLab data model into Cyfronet's MySql data server
      schema.
    </rdfs:comment>
    <owl:imports rdf:resource="http://www.virolab.org/onto/data/"/>
  </owl:Ontology>
<!--
    CLASSES MAPPINGS
-->
  <rdf:Description
rdf:about="http://www.virolab.org/onto/data/Patient">
    <vl-upper:mappedTo
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      test:patient
    </vl-upper:mappedTo>
  </rdf:Description>
<!--
    RELATIONS MAPPINGS
-->
  <rdf:Description
```

  In the near future we plan to integrate QUaTRO with all data sources available, especially with those accessible by Data Access Service. This will be implemented by using Data Access Client, running within a remote Grid Space Engine instance, connected by a proprietary protocol.

- Semantic descriptions. Currently our ontologies are stored at a CYFRONET's server, and are available via the HTTP protocol.

Because QUaTRO does not need to query RDF and is capable of processing semantic data itself, we did not assume to use a more sophisticated ontology storage. QUaTRO accesses and downloads required ontologies by using a standard Java HTTP client functionality.

Integration of Portal with Data Access Service (DAS Portlet)

To assist end-users, database administrators and mainly technicians (application developers) to browse and query distributed databases, a nice and user-friendly graphical user interface (DASPortlet) is being developed and foreseen to be integrated within the ViroLab Portal.

The preliminary version of this application is currently installed at HLRS's portal instance, which has been set up for testing purposes only, and interacts with the Data Access Services [Assel07] (DAS) to access the dispersed biomedical databases. The DAS provide standard web service interfaces allowing the smooth invocation of single services' capabilities. In order to connect with these interfaces remotely, so-called web service proxies or "stubs" are available that hide all communication and transformation activities from the users and developers respectively. The basic interactions between the portlet and services are based on SOAP over HTTP or additionally, to increase the level of security by transferring encrypted messages [Kipp07] only, HTTPS could be simply switched on.

All available services' functionalities are supported by the portlet. This includes the listing of available and/or accessible resources, the viewing of certain databases and their released schema definitions, and finally, the submission of simple and complex queries, in particularly, to all accessible resources simultaneously.

Furthermore, while using the remote data services, the already authenticated user has to authorize himself with the DAS at first before he can perform any actions on the aggregated data sources. Details on the security principles and functionalities are described in Section 5.1.1.8.

In the following, some screenshots showing the current version of the DAS Portlet, together with a short description, are presented in order to explain and highlight the major features.

If a user has successfully logged in to the portal and selects the DAS Portlet application, he/she directly obtains an overview of all available resources (Figure 38).

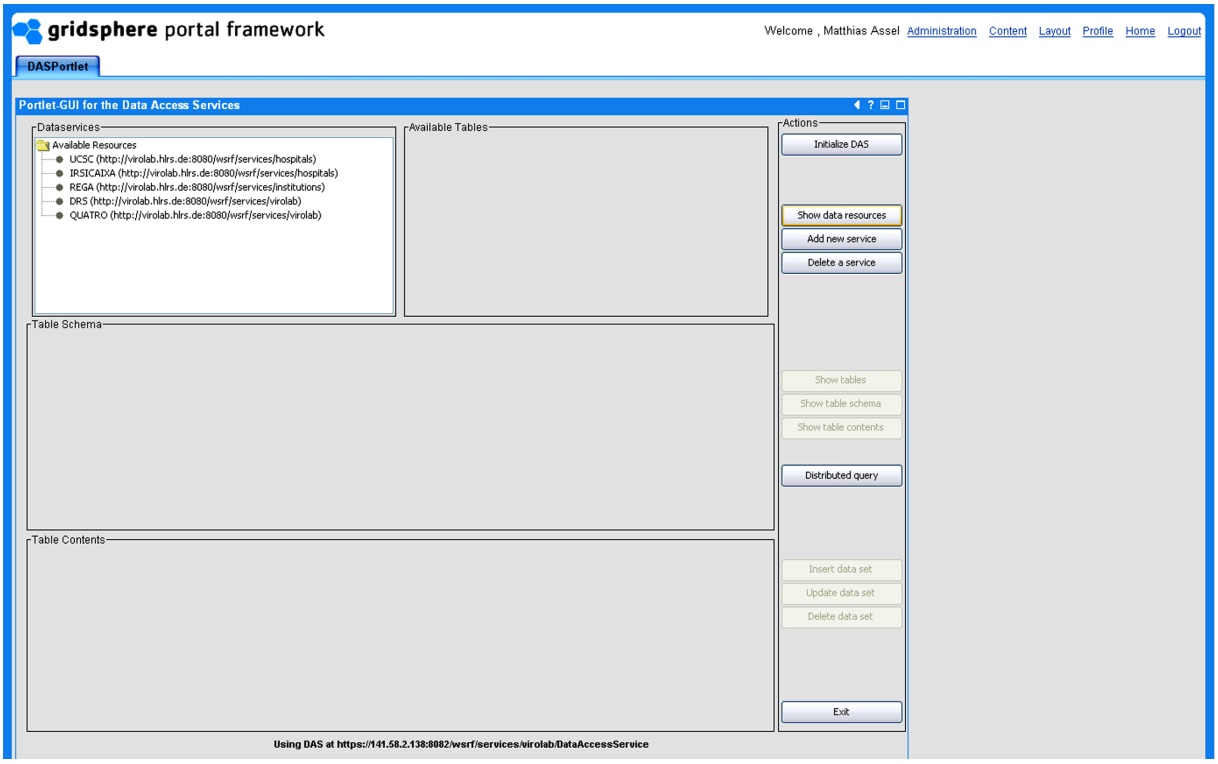


Figure 38 Available resources in DAS portlet

All listed resources are connected via the DAS, and can be used for further processing. If one of the resources has been selected by the user, he might be interested in browsing the content of the corresponding database. To view for example the available tables, he/she simply needs to click on the 'Show Tables' button. But instead of providing him/her with a set of tables, the DAS throw an exception saying that he/she is not allowed to browse the resource's content until he/she has been successfully authorized by the DAS (pls. refer to Section 5.1.1.8, Figure 39).

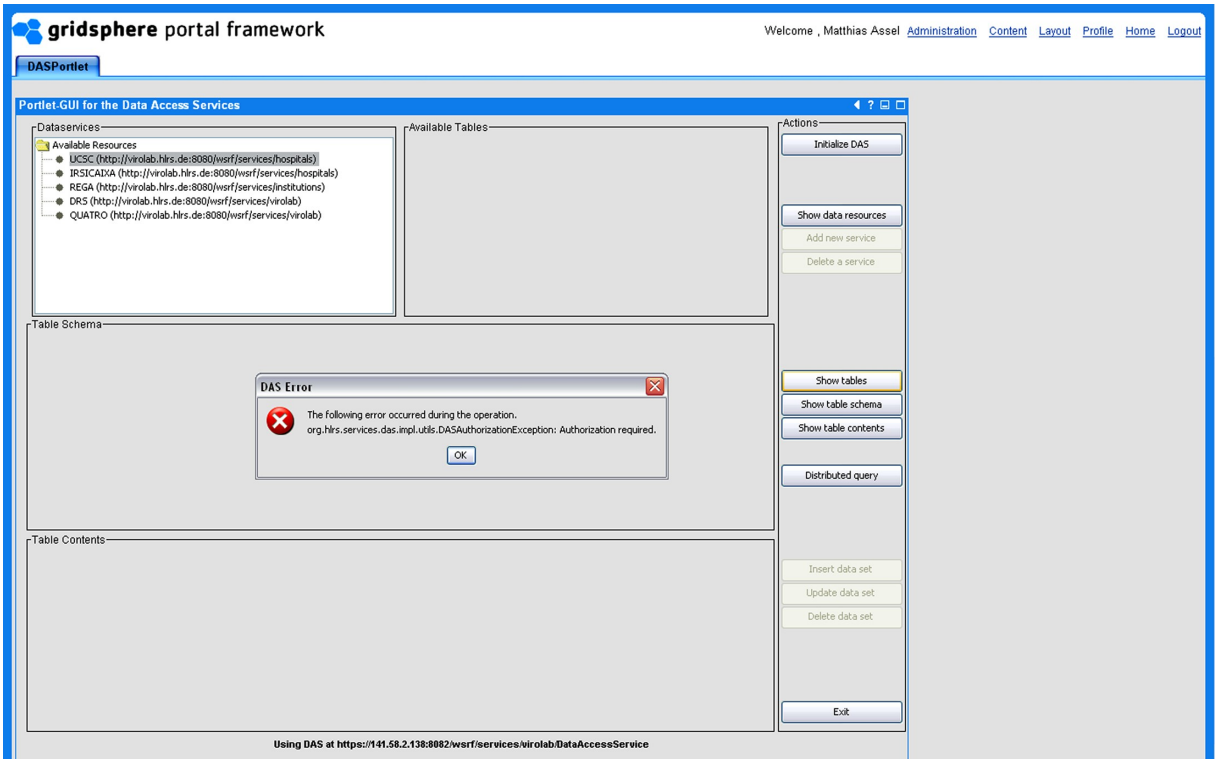


Figure 39 DAS authorization required

In order to deal with the available resources, the user is forced to authorize him/herself with the services. Therefore, he/she has to send his/her identity token and his/her home organization shortcut (basically the address of the corresponding IDP) to the DAS that take this information for requesting relevant attributes used to initialize the authorization process (Figure 40). Both, the identity token or user handle, and the IDP address can be obtained from the incoming HTTP POST request (forwarded by the corresponding IDP). This request contains all relevant parameters in form of hidden HTML form fields.

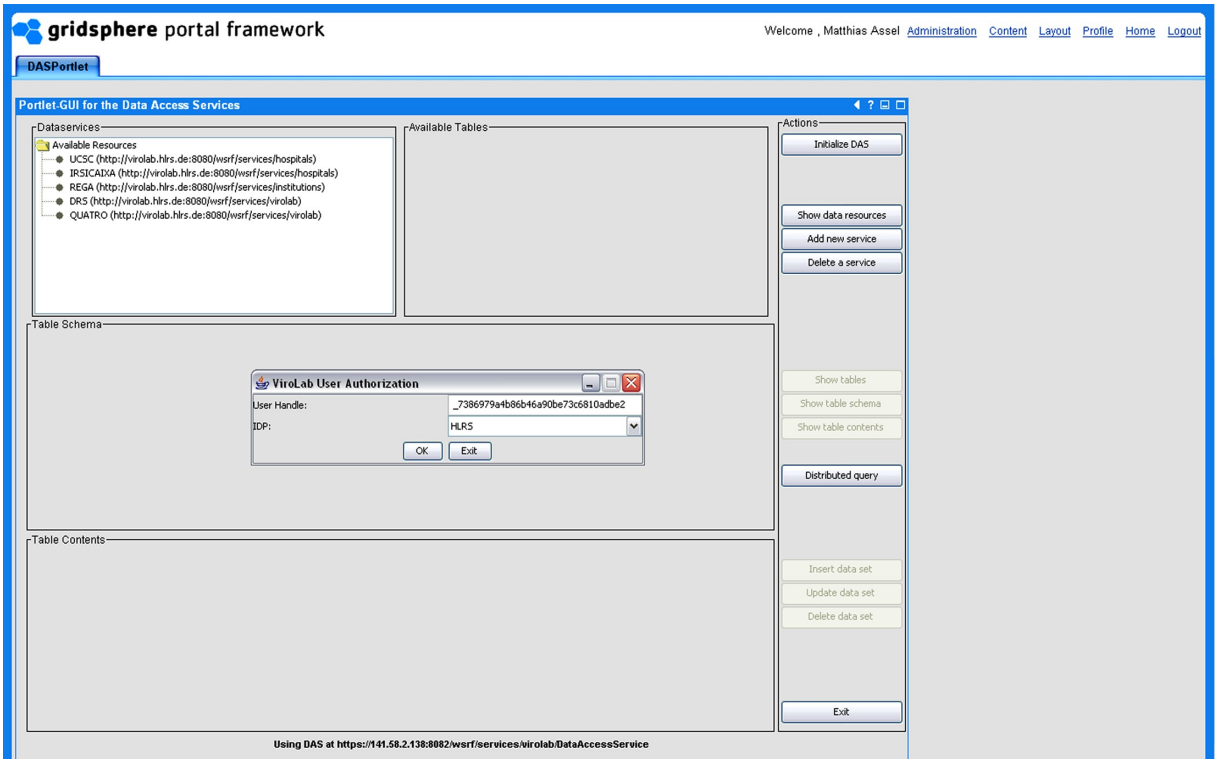


Figure 40 Inserting security token

After the user has been successfully authorized by the DAS, a list of available resources changes into a list of accessible resources is shown, which can now be browsed and queried by the current user.

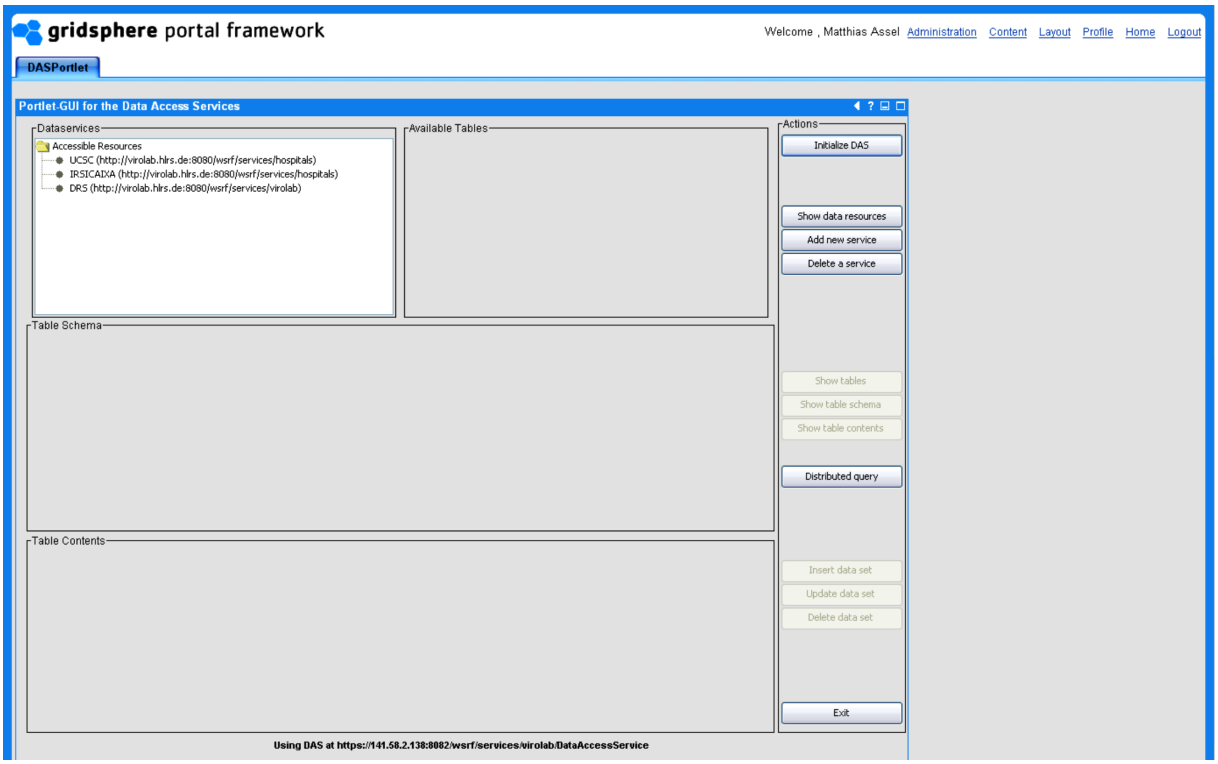


Figure 41 Accessible resources in DAS Portlet

To perform any actions on the individual resources, the user simply needs to select one of the provided resources. Having selected the corresponding resource, he/she can browse for example the content of specific databases or immediately send a distributed query to multiple resources concurrently. The results of this query are shown in the lower part of the application ('Table Contents'), where the user directly sees from which resource the data has been collected (Figure 42).

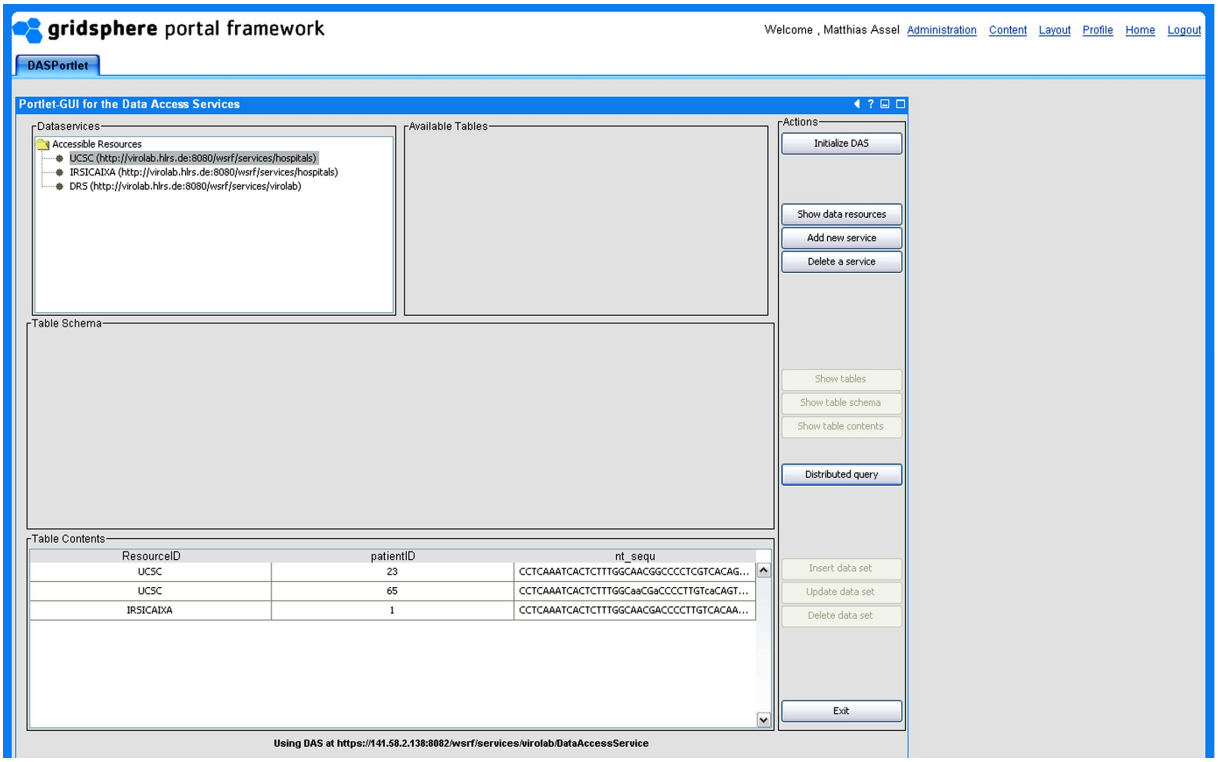


Figure 42 Table of contents in DAS Portlet

All these basic functionalities are principally considered as stable but will be further enhanced with regard to performance, reliability, and scalability. New releases of the DAS and the DASPortlet will facilitate the communication between the application and the underlying resources.

Therefore, a higher language based on common natural terms instead of pure SQL statements shall be used to query data from corresponding databases. Moreover, the possibility of passing queried data sets to other available portlets shall also be considered in the future.

Integration of Portal with GSEngine and Experiment Repository

Portal integration with GSEngine and Experiment Repository is realized by an intermediary component – Experiment Management Interface (EMI). EMI in the environment of the Gridsphere framework is a separate portlet as depicted in Figure 43. It is composed of three views:

- *Repository View* – This view is a direct graphical user interface of the underlying experiment repository. It presents a list of available

experiments with detailed information about each experiment, its authors, release comments, etc. In particular it is possible to view the scenario script, license agreement and submit feedback to the experiment developers by issuing tickets or comments.

- *Execution View* – It is responsible for presenting the execution status of running experiments. When a user input is required during the execution of a particular experiment it is reported back to the user by this view.
- *Result Management View* – When an experiment finishes it usually produces scientific results which are listed and presented by this view.

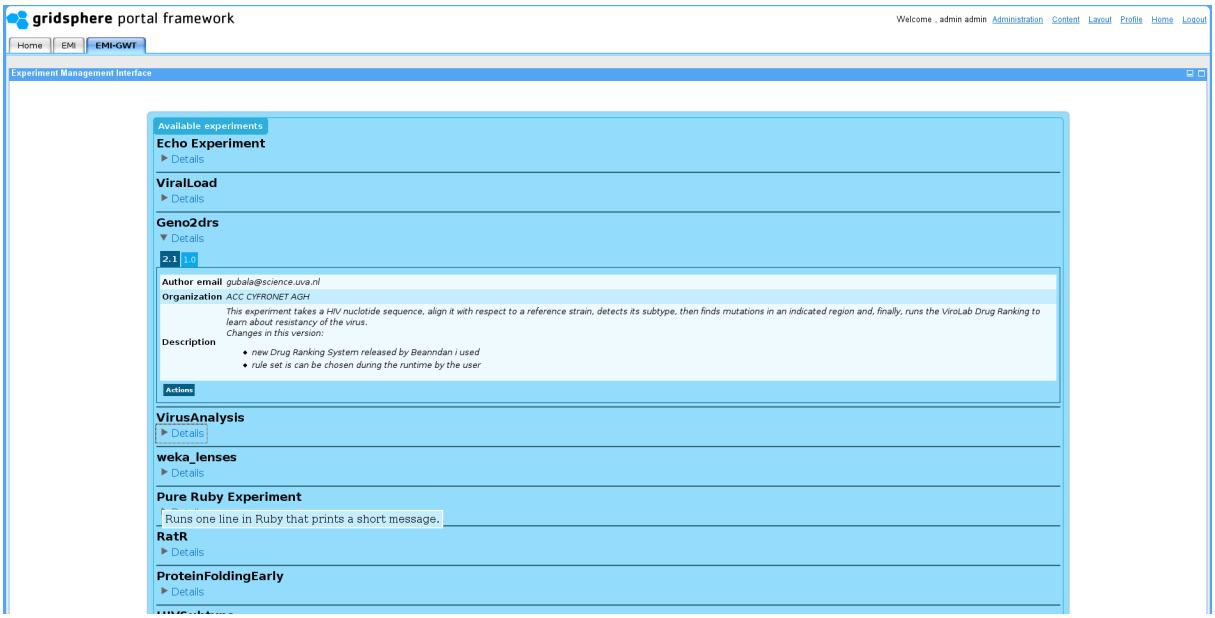


Figure 43 Experiment Management Interface incorporated in the Gridsphere environment presenting part of the Repository View

Integration of the Experiment Management Interface (EMI) with the external Grid Space Engine Server is implemented by including a client package in the EMI release. On the application server side the included library is configured with the engine's endpoint and the connection is established. It is presented in Figure 44 as a component diagram. The type of communication between the EMI and the GSEngine during experiment evaluation is asynchronous and is described in detail in section 3.2 of the [D3.4] deliverable.

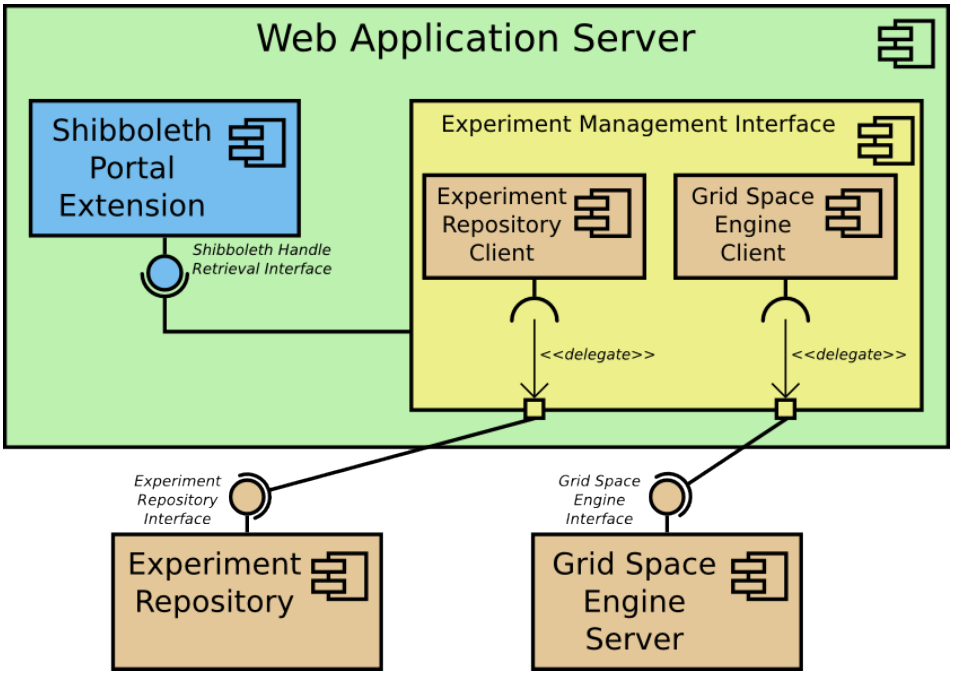


Figure 44 EMI integration points with Grid Space Engine and Experiment Repository

In order to fulfill the security requirements EMI also uses the Shibboleth Handle Retrieval Interface to obtain a Shibboleth user handle in order to delegate it to external nodes (such as GSEngine or Experiment Repository). This integration is accomplished by extending the GridSphere portal with a Shibboleth handle provider which exposes the mentioned interface to retrieve the handle. The interface is implemented by using the GridSphere container's session facility which passes the user credentials from the container to the EMI's JavaScript environment.

As depicted in Figure 44 EMI uses an *Experiment Repository Client* to communicate with the external Experiment Repository. The client library is packaged together with the EMI release and ready to work after configuring the endpoint of the repository and providing the user credentials. This is done by delegating a user Shibboleth handle obtained from the portal Shibboleth extension package. An important requirement here to fulfill by the repository's deployment is to be compatible with Shibboleth authorization mechanism. The extension is critical for the EMI to work properly with the repository.

Next major milestone in the EMI's roadmap is to implement more result management routines so the results may be shared and searched. Another improvement will be the support for annotations and comments of different pieces composing the EMI environment such as experiments, results or repositories.

Planned Functionalities of Portal

There are a few tasks to be performed in the future concerning the Portal. Among other things this task covers: secure elements at the level of portal

(access to portlet only for authorized users), a collaboration tool for currently logged on users, simplification of the Portal and its usage.

4 Middleware

Middleware layer includes access to computing resources and to monitoring infrastructure. Access to computing is provided by the adapters for various underlying middleware technologies, which are plugged in to Grid Operation Invoker of Runtime System (WP3). Currently we support such technologies as Web Services, MOCCA components, EGEE jobs and initial support for WSRF services. The high-level overview of the ViroLab middleware layer is shown in Figure 45.

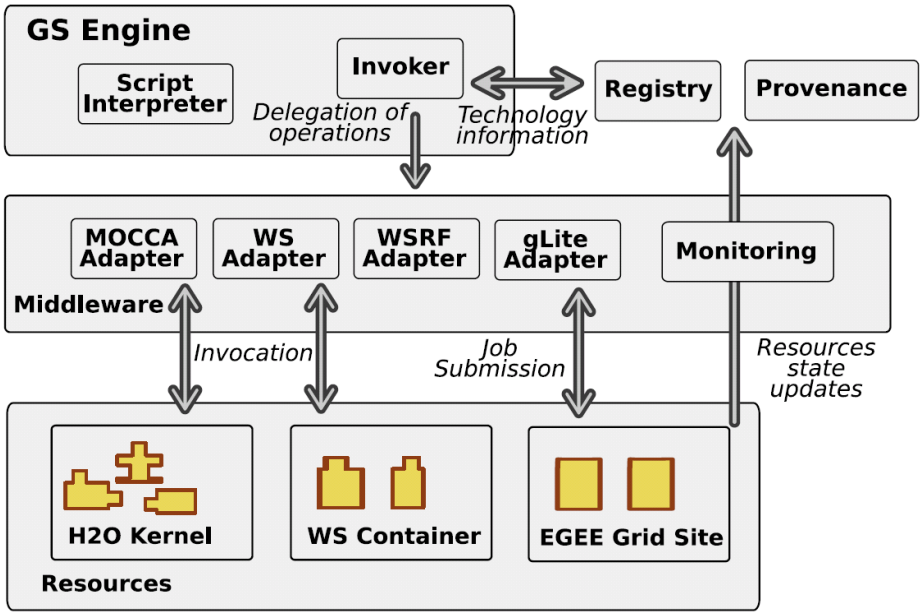


Figure 45 Overview of ViroLab middleware

The middleware was integrated with WP3 (GSEngine) and supports ViroLab applications (WP4) such as Drug Ranking System (as a Web Service) and RegaDB Tools (alignment, subtyping) using WTS services.

The infrastructure monitoring has been integrated with the Grid Resources Registry (GRR) of WP3 and the integration with Provenance system is under development.

Features in Version 1 of Middleware

Within the middleware task a client-side library (named *Grid Operation Invoker*) has been developed. The *GOI* implements the Grid Object abstraction [VLINV] over distributed computational resources and enables to access them remotely within an experiment in a uniform manner. The library is implemented in *JRuby* and relies on *Java* client-side libraries. Additionally, the installation of the *EDG User Interface* is required in order to enable usage of the *EGEE* infrastructure. The *GOI* architecture is modular and every middleware technology is supported by a dedicated

adapter [D2.2]. The *Grid Operation Invoker* provides a simple and uniform API for creating client-side representatives for *Grid Objects* [D3.3-A2]. Representatives interface remote computational resources in their specific communication protocols, but are interfaced like ordinary Ruby objects. At a runtime, based on the description retrieved from the *Grid Resource Registry*, an appropriate adapter is loaded and a representative is created.

On the current stage of development the middleware client-side library allows:

- Invoking *Grid Operations* in a uniform way on computational resources using leading middleware suites. The *GOI* library contains:
  - *WsAdapter* which is based on the Ruby built-in support for Web Service.
  - *WsrfAdapter* (prototype support) that relies on the Java WS Core 4.0.6 provided by the Globus Toolkit.
  - *MoccaAdapter* which uses MOCCA Java client-side libraries.
  - *Witty Services* (*WTS*) that is based on the Java client for the WTS middleware.
  - *LCG* (*EGEE* infrastructure) which uses the *EDG User Interface*.

  WTS and *LCG* are job-oriented technologies therefore a wrapper for software published with these technologies is required in order to expose them in an object-oriented manner [TBCA].
- Calling operations in an asynchronous mode (prototype support), even though asynchronous operations are not supported on the server-side. A future variable is returned each time an asynchronous operation is invoked and a new thread is started on the client-side. Experiment interpretation is not suspended unless a result of an asynchronous operation is required but is not ready.
- Reporting invocations of *Grid Operations*. Each time an operation is invoked *GridOperationInvoking* and *GridOperationInvoked* events are reported to the monitoring system.
- Calling "Shibboleth enabled" web services. At runtime, *GridSpace Engine* assigns a Shibboleth handle to a global constant. If a service is described as being secured with the Shibboleth the *GOI* library transparently includes a Shibboleth handle in requests to a secured service. For information how a *GridSpace Engine* retrieves a handle please refer to Section 5.1.1.8.

Grid Service Availability Monitor (GSAM) has been developed to provide monitoring of the services which are able to be invoked by *GOI.* GSAM task is to provide information about availability of services as well as performance information concerning nodes where services are deployed. For performance information like usage of memory or CPU additional sensor installation on a node which should be monitored will be required.

GSAM uses Java Message Service (JMS) for communication and temporary as an interface. It will be changed for uniform monitoring interface common for provenance and GSAM.

At the present stage of development following functionality is provided:

- Monitoring state of the services whether they are ACTIVE or DEAD (or NOT_CHECKED at start). It is performed by periodic request sending on a registered endpoint address and waiting for a configurable time for response. If any response is given then monitored service gets ACTIVE status and DEAD otherwise.
- GSAM is constantly listening on a JMS topic for messages with list of services to be added to monitoring either to remove from monitoring list.
- GSAM sends status of services to Grid Resource Registry (GRR) periodically with configurable time gap.

Middleware Integration Points

Integration of Middleware with GSEngine

GridSpace Engine is responsible for interpretation of experiments utilizing the middleware library therefore every distribution of GSEngine contains the *Grid Operation Invoker* library as well as all libraries it requires. Both the *GOI* library and *GSEngine* are described with Maven Project Object Model [POM] files in order to ensure that all library dependencies are met and to facilitate releasing of the software.

The *Grid Operation Invoker* library is fully integrated with the GSEngine and its functionality is accessible within an experiment using Ruby language.

Integration of Monitoring with Provenance

Provenance information is an ontology-based record of an experiment's execution. The process of recording provenance is based on monitoring events related to the execution of the experiment, collecting them in a Semantic Aggregator which translates the low-level monitoring events into high-level ontology representation and publishes the ontology records in Provenance Tracking System (PROToS) [Balis07].

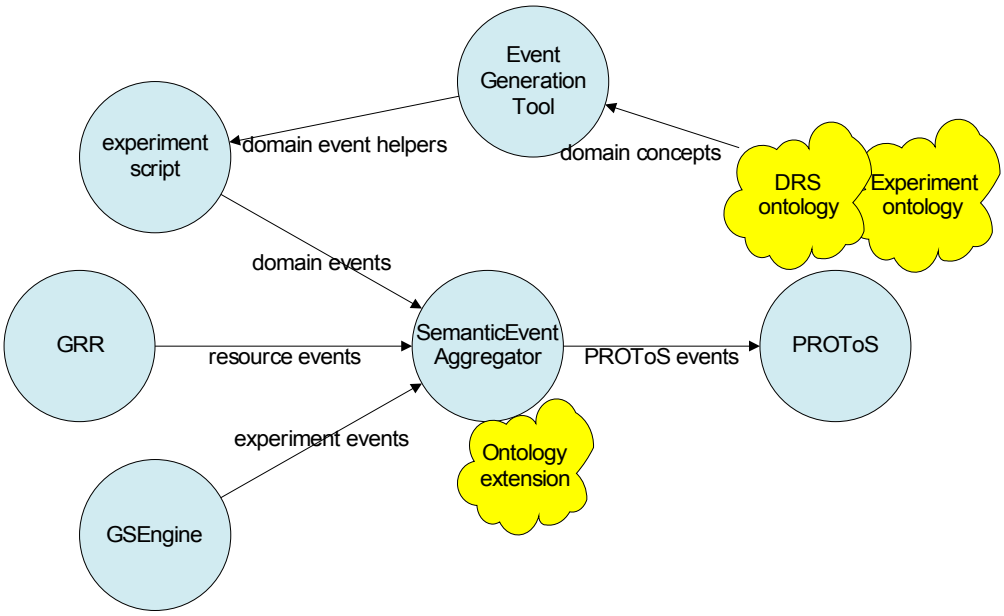


Figure 46 Monitoring data collection process

This process is depicted in Figure 46. The Semantic Event Aggregator is described in [D3.4].

All monitoring events are published through monitoring facade, which is, at the current stage of development, implemented utilizing *log4j* remote appenders. The following type of XML events are passed to Aggregator:

- ExperimentStarted
- ExperimentFinished
- GridOperationInvoking
- GridOperationInvoked
- GridObjectRegistered
- GridObjectInstanceRegistered

The above events are created in GSEngine and GRR components. In order to enable the correlation, events corresponding to experiment course are augmented with an experiment identifier, while events corresponding to Grid Objects calls are additionally augmented with a unique identifier generated by GSEngine. The *Castor* framework provides mapping between events XML schema and Java classes that are used in events creation. In the future, we plan to support JAXB API.

In addition to the generic events, the following domain events are generated for the Drug Ranking System experiment:

- NewDrugRanking
- DRSAlignment
- DRSSubtyping

The domain events are created in the experiment script utilizing helpers that are automatically generated from DRS ontology by EventGenerationTool.

Integration of Monitoring with Registry

GRR registry uses the monitoring system to monitor the availability of services. The availability monitor to which GRR is connected is called GSAM (Grid Service Availability Monitor). GRR and GSAM are connected by two channels implemented in Java Message Service (JMS) (Figure 47):

- The first channel is used by GRR to request availability monitoring of a service, passing its endpoint; or to request to stop the current monitoring of a service.
- The second channel is used by the monitoring system to send monitoring events to GRR which contain the status of the monitored service.

The events with information about the availability of service are created periodically with a preconfigured rate. The status of the service passed in these events can be: ALIVE, DEAD or NOT_CHECKED.

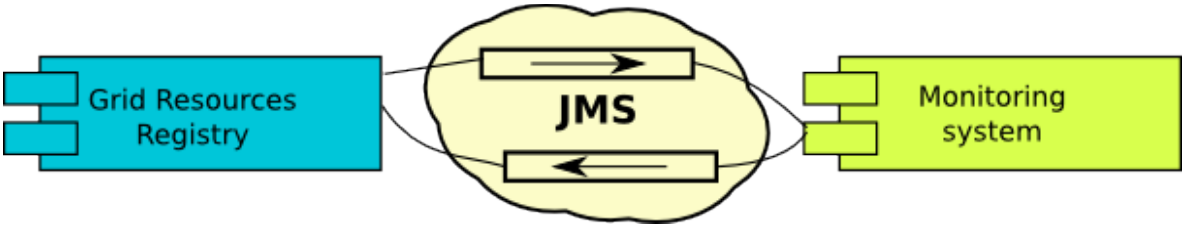


Figure 47: Communication between Grid Resources Registry and monitoring system.

The scenario of service availability monitoring is presented in Fig. 48. It consists of the following steps:

- A new service is assumed to be manually registered in GRR by a Registrator.
- GRR in turn registers the new service (or a list thereof) in GSAM. GSAM initially sets the status of the new services as NOT_CHECKED.
- GSAM periodically monitors the availability of all registered services by sending requests to their endpoint addresses.
- If GSAM receives a response from the requested service within a preconfigured time, it sets the service's status to ACTIVE. The DEAD status is set otherwise.
- GSAM periodically sends update to GRR with status of all services registered for monitoring.
- GRR can also send a request to stop the monitoring of a given service.

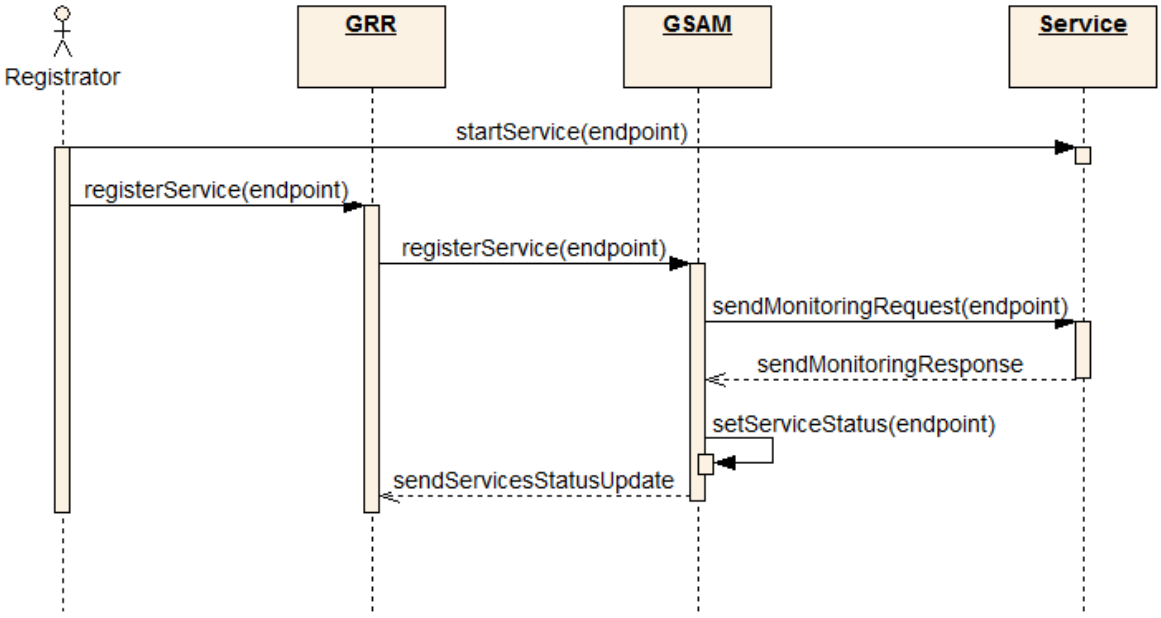


Figure 48: Scenario of GSAM services monitoring and interaction with GRR.

GSAM is configured in an XML file which contains the following parameters:

- URL of the JMS communication broker,
- JMS topic name for adding or removing services to be monitored,
- JMS topic name to publish the service status updates,
- The notification interval,
- The connection timeout for checking the availability of a service,
- The check interval to refresh the list of services.

An example configuration file of GSAM is shown below:

```
<gsam>
    <!-- sets address to the jms provider endpoint -->
    <param name="activeMqUrl" value="tcp://virolab.cyfronet.pl:61616"/>
    <!-- topic name for adding/removing services-->
    <param name="newServiceTopicName" value="newServices"/>
    <!—topic name for service state notification -->
    <param name="notificationTopicName" value="serviceStateNotification"/>
    <!--interval in ms how often should state of services be send-->
    <param name="notificationInterval" value="5000"/>
    <!-- timeout in ms when service is treated as DEAD -->
    <param name="connectionTimeout" value="3000"/>
    <!--time in ms between next checking of services list-->
    <param name="checkInterval" value="3000"/>
</gsam>
```

Currently, the GSAM interface is based on JMS which uses serialized Java classes to represent events (described below). In the near future this will be replaced with a language-neutral XML-based representation.

In addition to the service availability status, GSAM passes information concerning capabilities of the resource (node) on which the service is hosted. This information contains node's RAM and CPU, and is later consumed by GridSpace Application Optimizer (GrAppO).

Integration of Middleware with DRS

Drug Ranking System application is exposed as a Web Service and can be accessed within the ViroLab virtual laboratory. The functionality provided by the DRS is accessed in experiments, such as *drs_experiment* and *geno2drs2drs_experiment* [EXP]. These experiments use the *Grid Operation Invoker* middleware library and can be executed either in the EPE, EMI or using GSEngine command-line tools.

Integration of Middleware with RegaTools

Rega Tools are published using the *Witty Services* (*WTS*) middleware which is supported by the middleware library. Alignment and HIV subtype tools are accessible within the ViroLab virtual laboratory and are used in the *align_wts*, *hiv_subtype* and *geno2drs_experiment* experiments [EXP].

Planned Functionalities of Middleware

The *Grid Operation Invoker* middleware library is already operational and used in many experiments. Nevertheless, the work on GOI will be carried on. Our efforts will be targeted at:

- enhancing adapters for WSRF and LCG technologies,
- enhancing support for asynchronous operation invocation,
- providing more information about experiment execution and results to provenance system,
- implement introspection mechanism to take advantages of the Ruby scripting language,
- developing an adapter for the AHE middleware,
- implementation of a mechanism for registering created *MOCCA* components in the *Grid Resource Registry*.

The future work on monitoring and provenance integration will be focused on supporting XML-based formats of propagated events as well as on improving the stability and performance of the prototypes.

5 Security

In order to develop a reliable security framework that meets the project requirements the following assumptions were made:

- a Virtual Organization is formed from the combination of resources and users
- all participants in a Virtual Organization trust each other;
- no central database of users - system should utilize the existing users' databases of hospitals, universities, scientific institutes;
- policy is the set of attributes which is needed to access a specific resource;
- each service/data provider creates their own policy to ensure that only people with proper attributes can access it;
- users are only allowed to use a resource when their attributes match the policy.

During the first phase of the project it was decided that ViroLab security framework will be based on Shibboleth [Shibboleth]. Key concepts within Shibboleth include:

- Federated Administration (trust fabric exists between Home Organizations, which allows each site to identify the users, and assign trust level; HOs are responsible for user authentication, while Resource Providers are responsible for authorization)
- Authorization is attributes based – each resource ask user's Home Organization about attributes, and make an authorization decision
- Standards Based – this solution uses OpenSAML for the message and assertion formats, security data exchange protocol is based on Security Assertion Markup Language (SAML).

To provide Virtual Organization capabilities a new and innovative approach was taken. It uses proved solutions like Shibboleth and SAML, but in a new and modified way that aims to fulfill the specific ViroLab requirements. A dynamic Virtual Organization is established by using attributes to identify user privileges, which the resource providers can use for authorization purposes. The framework will be developed to secure all resources accessible in the Virtual Laboratory (e.g. Web Services, SVN).

Features in Version 1 of Security Framework

In the current version, the security framework provides several modules and tools that allow to create reliable and dynamic Virtual Organizations. As it was described in [D2.2], the members of Virtual Organization depend on the attributes which they have and the policies of resources. Based on that, the following features are accessible in version 1:

- Possibility to authenticate at one portal instance for users from different Home Organizations (HO). Several organization have already been integrated (GridwiseTech, Cyfronet, HLRS). A separate "Orphan IdP" has also been created and managed by GridwiseTech, as placeholder for participants from member institutions whose IdP has not been installed yet.
- In order to authorize the user, each resource has to get the attributes from users' HOs and compare them with local policies to grant or deny access.
  The module which is responsible for this task was created. It is named ShibAuthAPI and it is Java-based module which performs the attributes request and authorization.
- There is a different possibility to perform the authorization to using the ShibAuthAPI. For each of the resources (or even a group of them) the authorization point can be created. In order to grant or deny access for a particular user, the resource invokes the authorization point with proper arguments. This solution is named ShibRPC and it does not influence the resource as much as ShibAuthAPI, and is not limited to Java language either.
- One of the module which had to be secured is ExpRepo. This is the experiment repository based on SVN. In order to make this resource secured, the Authorization point based on ShibRPC was created.
- As the example and model solution of securing the services the Axis WebService were chosen. The process of granting or denying access to this resources is based on ShibAuthAPI

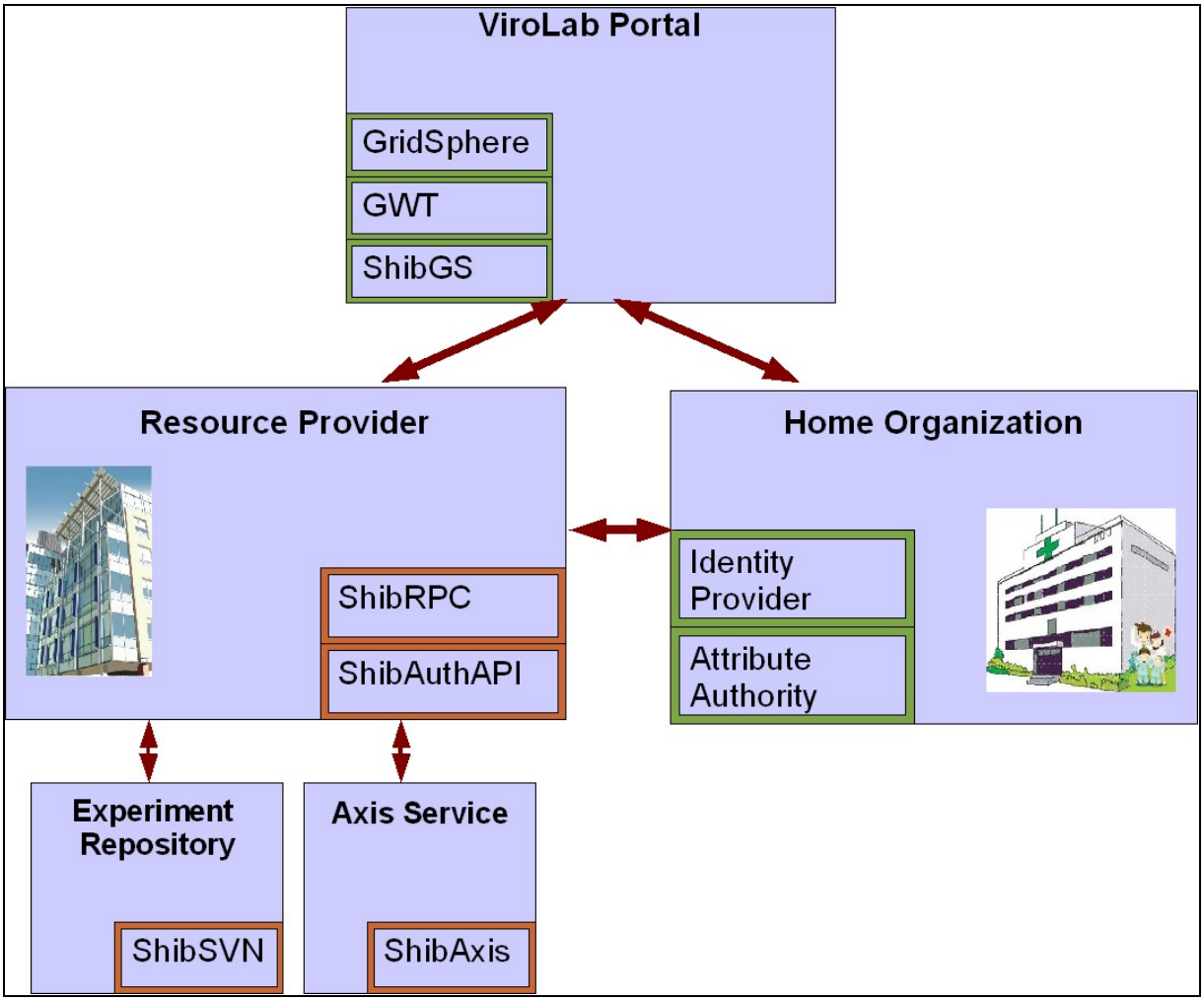


Figure 49: Security integration points

All these elements are presented in Figure 49 and described in the following sections.

Attribute request library (ShibAuthAPI)

The aim of Shibboleth Attribute Authority API, or ShibAuthAPI for short, is to provide a general interface that can be used by other developers to *shibbolize* their modules. The users of these modules will be authorized via Shibboleth. ShibSVN and ShibAxis are example shibbolized modules.

ShibAuthAPI contains two elements:

1. AttributeRequestor
2. PolicyResolver

5.1.1.1AttributeRequestor

AttributeRequestor (`com.gridwisetech.shibboleth.auth.AttributeRequestor` class) performs the attribute request for a given user. Firstly, the user is required to log on to ViroLab portal using his/her login name and password in order to receive Shibboleth handle and Shibboleth Identity

Provider (IdP) URL. These values are later used for querying Shibboleth IdP about the user attributes. If the correct attributes are returned by Shibboleth, PolicyResolver is asked for making an authorization decision.

AttributeRequestor requires the following input arguments:
- Shibboleth handle (e.g. *_133e6bb9458bc857cb1cf031d93e553a*)
- Shibboleth IdP URL (e.g. https://virolab.gridwisetech.pl/shibboleth-idp)

AttributeRequestor parses ShibAuthAPI configuration file (here: `conf.xml`) to find the following information:
1. certificate directory (<*certDir*>)
2. all available Shibboleth IdP certificates, specified as filename and IdP URL pairs (<*certificate*>)
3. Shibboleth SP certificates (assigned to given IdP certificates) as URLs (<*sp*>)
4. Shibboleth IdP/AA certificates, specified as filename and AA URL pairs (<*aa*>)
5. service provider certificate (<*spCert*>)
6. service provider private key (<*spKey*>)
7. Java Key Store server-side file (<*serverJKSFile*>)
8. Java Key Store client-side file (<*clientJKSFile*>)
9. default Java Key Store alias (<*defaultJKSAlias*>)
10. default Java Key Store password (<*defaultJKSPassword*>)
11. access policy file name (<*policy*>)

Example configuration file looks as follows:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<config type="sp">
      <certDir>etc/</certDir>
      <idp_certificates>
              <certificate>
             <filename>idp_gwt.crt</filename>
             <url>https://virolab.gridwisetech.pl/shibboleth-idp</url>
                       <sp>https://virolab.gridwisetech.pl/shibboleth-sp</sp>
                       <aa>
                    <filename>idp_gwt.crt</filename>
                    <url>https://virolab.gridwisetech.pl:8443/shibboleth-idp/AA</url>
             </aa>
     </certificate>
     <certificate>
             <filename>idp_cyfr.crt</filename>
             <url>https://virolab2.cyfronet.pl/shibboleth/idp</url>
                       <sp>https://virolab2.cyfronet.pl/shibboleth-sp</sp>
                       <aa>
                    <filename>idp_aa_cyfr.crt</filename>
                    <url>https://virolab2.cyfronet.pl:8443/shibboleth-idp/AA</url>
             </aa>
     </certificate>
               <certificate>
             <filename>hlrs.crt</filename>
             <url>https://gridmania.hlrs.de/shibboleth/testshib/idp</url>
             <sp>https://csharp.hlrs.de/shibboleth/testshib/sp</sp>
             <aa>
                    <filename>>hlrs.crt</filename>
                    <url>https://gridmania.hlrs.de:8443/shibboleth-idp/AA</url>
             </aa>
      </certificate>
  </idp_certificates>
  <spCert>sp.crt</spCert>
  <spKey>sp.key</spKey>
  <serverJKSFile>shib_server.jks</serverJKSFile>
```

```
      <clientJKSFile>shib_client.jks</clientJKSFile>
      <defaultJKSAlias>globus</defaultJKSAlias>
      <defaultJKSPassword>globus</defaultJKSPassword>
      <policy>policy.xml</policy>
    </config>
```

5.1.1.2PolicyResolver

PolicyResolver (`com.gridwisetech.shibboleth.PolicyResolver` class) checks location of its configuration file (here: `policy.xml`) in ShibAuthAPI configuration file (here: `conf.xml`). Then it loads access policies from its configuration file. In the end, it authorizes users, or denies them authorization, basing on loaded policies

Example access policy configuration file looks as follows:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<policies>
  <policy>
      <publication_date>2008-01-01</publication_date>
         <attribute>
           <name>mail</name>
          <authorized>jb@mail.com</authorized>
    </attribute>
  </policy>
  <policy>
  <publication_date>2008-02-01</publication_date>
    <attribute>
  <name>Organization</name>
  <authorized>GridwiseTech</authorized>
  </attribute>
  <attribute>
    <name>Role</name>
    <authorized>Doctor</authorized>
    <authorized>Developer</authorized>
  </attribute>
  <attribute>
    <name>Name</name>
    <authorized>John Brown</authorized>
  </attribute>
  <attribute>
    <name>Mail</name>
    <authorized>jb@mail.com</authorized>
   </attribute>
   </policy>
</policies>
```

Each policy has its publication date (*publication_date*) and a list of tags set.

Policy attribute tags have the following meanings:

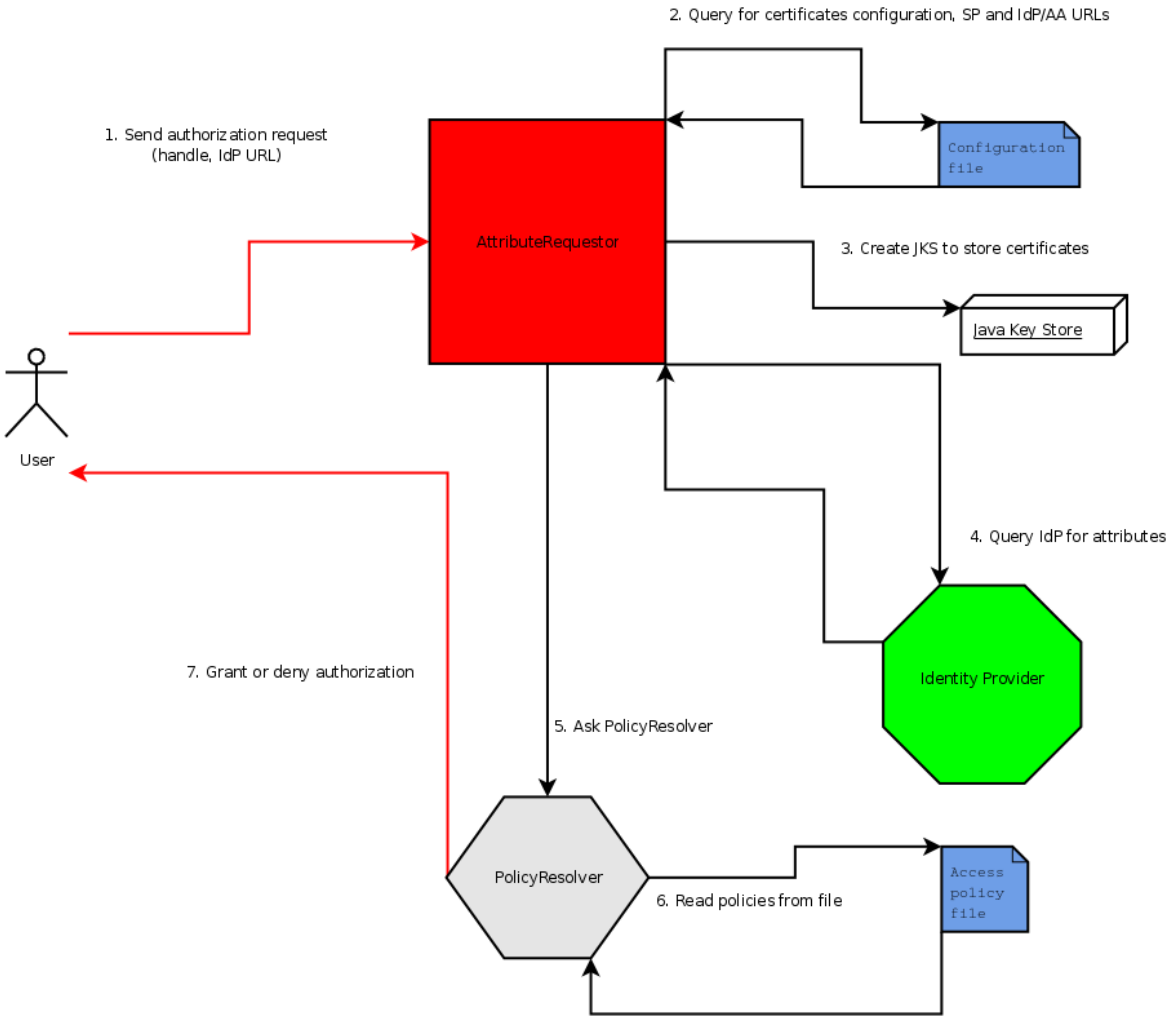
- name: the attribute name of the attribute

- authorized: the authorized value(s) for the attribute

The list will be extended depending on Virtual Organization requirements. Example future enhancements are the following:

- HomeOrganization
- TypeOfOrganization
- Department
- Role

ShibAuthAPI at work



Authentication and authorization process using ShibAuthAPI looks as follows:

1. User sends authorization request using Shibboleth IdP URL and Shibboleth handle previously obtained from ViroLab Portal
   AttributeRequestor (AR) object is created using obtained Shibboleth values
2. AR reads certificates configuration, Shibboleth SP URL and IdP/AA

URL from ShibAuthAPI configuration file
3. AR creates new Java Key Store file for keeping certificates and keys
4. AR queries IdP about user attributes
   AR extracts attributes from SAML response that was received from IdP
5. PolicyResolver (PR) object is created using previously obtained user attributes
6. PR loads ShibAuthAPI configuration file (`conf.xml`) to find location of policy configuration file (`policy.xml`)
7. PR loads all policies from policy configuration files
8. PR matches user attributes with loaded policies. If any policy matches, then user is authorized

5.1.1.3Usage example

Client-side code to authorize the users should be similar to the code presented below:

```
AttributeRequestor client = new
  AttributeRequestor("_133e6bb9458bc857cb1cf031d93e553a",
  "https://virolab.gridwisetech.pl/shibboleth-idp");

HashMap<String, String> attributes = client.queryAttributes();
boolean authorized = new PolicyResolver().isAuthorized(attributes);
client.cleanup();
return authorized;
```

Arguments of `AttributeRequestor` class constructor need to be replaced with appropriate Shibboleth handle and IdP URL.

ShibRPC

ShibRPC is an XML-RPC service which provides a bridge between Shibboleth and client applications. Using ShibRPC it is possible to make application Shibboleth-enabled (*shibbolized*) without doing tight integration with any of Shibboleth libraries. To access ShibRPC service one needs to use an XML-RPC client which meets XML-RPC standard specification. As this protocol is very popular, there are many ready-to-use libraries which are available for C/C++, Java, Ruby, Python, PHP and other programming languages which support TCP/IP sockets.

The advantage of ShibRPC is that it is based on ShibAuthAPI and exposes main functions which this API provides in an easy way for application developers. With ShibRPC the application can easily access user attributes such as: username, email, organization, etc. by passing Shibboleth *handle* and *IdP URL* to the XML-RPC method. Basing on these attributes, the application can decide whether a user can access service or not. As ShibRPC is based on ShibAuthAPI, handle authorization can also be done using `policy.xml` file on ShibRPC side. In this policy file Service Provider

can easily define which user attributes are necessary to access the invoked service.

Currently ShibRPC exposes two methods:

```
int ShibAttr.authorizeHandle(String handle, String idpUrl)
Map<String, String> ShibAttr.getAttributes(String handle, String idp)
```

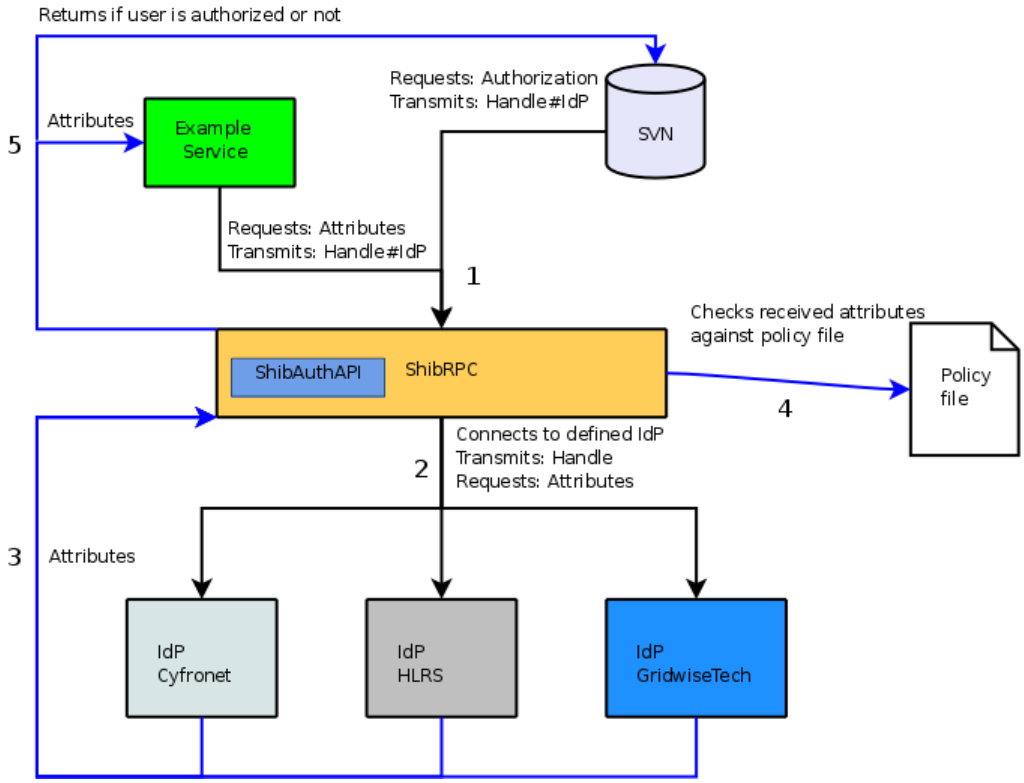


Figure 50 ShibRPC flow diagram

The included flow diagram presents the process of handle authentication. When a service needs to validate a user to Shibboleth basing on the handle, this service calls remote ShibRPC XML-RPC service by passing the *handle* and *IdP URL* as parameters (1). After having invoked XML-RPC method, ShibRPC connects to selected IdP and sends handle with a request for attributes (2). In response IdP sends attributes or information that the handle is no longer valid. After having received a response from IdP (3), ShibRPC checks attributes against policy file to decide whether the user can access the resource or not (4). Then response is sent back to the service which invoked XML-RPC method (5).

ShibSVN

The aim of ShibSVN module is to make access SVN resources from unmodified SVN client with Shibboleth authorization possible. The solution makes use of Java Shibboleth API (ShibAuthAPI), which decides about user authorization basing on Shibboleth handle.

ShibSVN consists of two elements:

1. Authorization module Apache HTTPd together with XML-RPC client

2. Java XML-RPC web service, placed on the separate WWW server which uses ShibAuthAPI in order to authorize the users

Authorization to access SVN repositories is based on "Basic Authentication", standard method included in HTTP 1.0 standard. This method makes logging to SVN repositories from WWW browser or other clients possible. In a standard way Apache HTTPd uses logins and passwords written in text files, LDAP or database.

Apache module *authn_shibattr* provides authorization to any WWW resource inside Apache server, using XML-RPC methods on a remote server. A user providing his/her handle and IdP address as a password is an important codition here. Handle and IdP should be separated by "#" character.

The authorization process is showed at the diagram:

1. User wants to checkout files from repositories https://protectedrepo/svn
2. User logs on to the portal using Shibboleth authentication and obtains the Handle and IdP URL.
3. User wants to connect to SVN and update/commit files
4. User should fill in a user field with e-mail and password with `handle#`
5. SVN client connects to Apache server sending the login and password
6. Apache passes login and password to module authn_shibattr
7. Module authn_shibattr invokes the XML-RPC method (`authorizeHandle`) on the remote Tomcat server with installed ShibRPC and sends handle and IdP URL as parameters.
8. Application ShibRPC obtains the attributes from the chosen IdP basing on the handle and decides if user could access the resources basing on policy file.
9. XML-RPC method sends back the result.
10. If authorization fails the SVN client should ask for password again. Otherwise, user will be granted access to the resources.

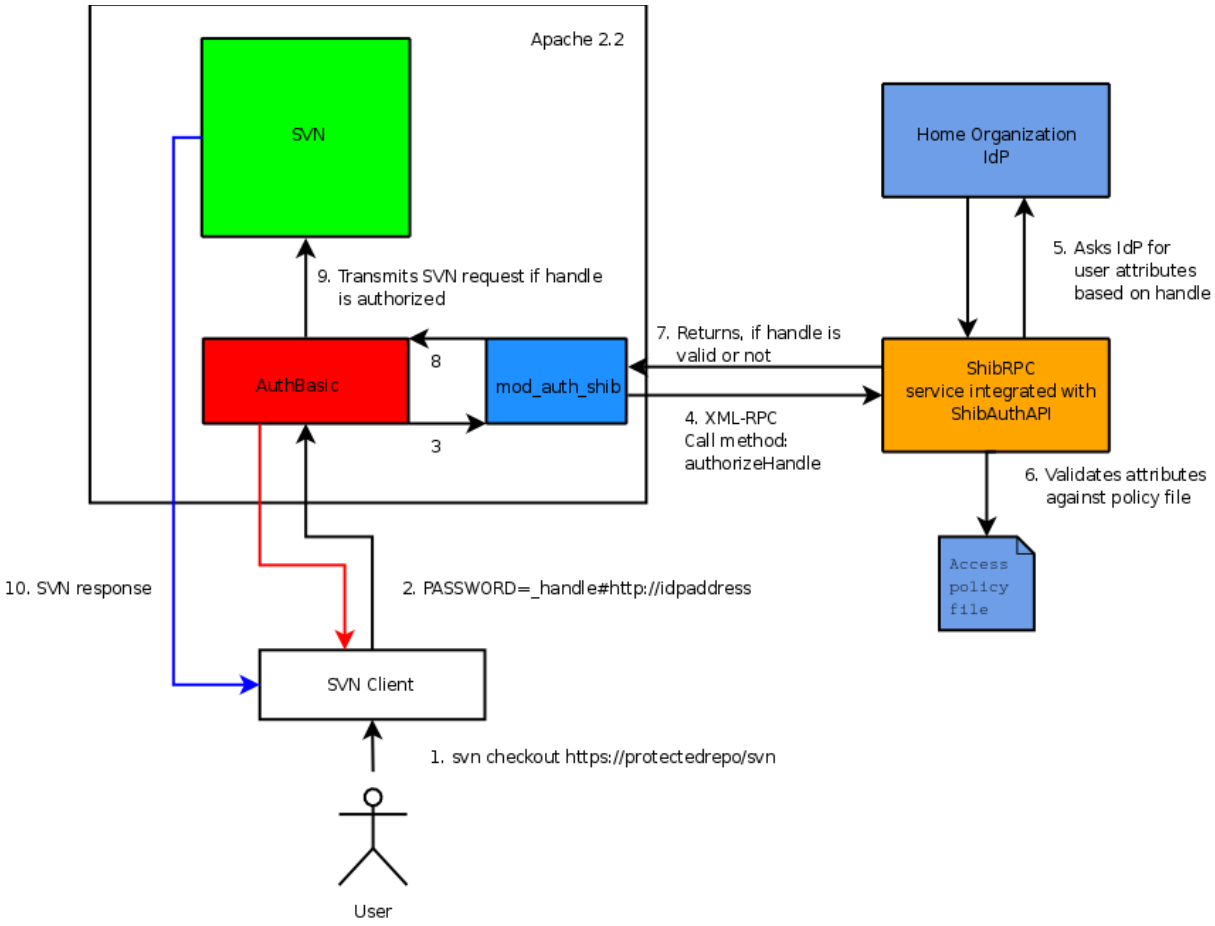


Figure 51: Authorization to Shibboleth-enabled SVN repository diagram

Shibbolized Axis

The aim of ShibAxis is to show a general approach to integration of Shibboleth with Apache Axis2 (Java). It is an Apache Axis module and service that requires the clients to authorize via Shibboleth. It also contains a simple client that connects to the aforementioned web service. In other words, it is an example of shibbolized web service using ShibAuthAPI.
ShibAxis contains three elements:

1. Apache Axis2 module
2. sample Apache Axis2 service
3. sample Apache Axis2 client

5.1.1.4Apache Axis2 module

Apache Axis processing of web service clients requests is organized into phases. We defined a new phase - *authorization phase*. The phase is configured in `module.xml` file (module deployment configuration file) and `$AXIS_HOME/conf/axis2.xml` (global Apache Axis configuration file). Each phase contains one or more Axis handlers. A client requests traverses through all handlers in sequence and needs to be accepted by all of them. Any of the handlers in question may either accept, suspend or abort

further request processing. We created one handler inside authorization phase - *authorization handler* (`com.gridwisetech.shibboleth.auth.AuthorizationHandler` class). As soon as Axis processing flow reaches the handler, its method ''invoke'' is run and returns acceptance decision. In this case we decided that if a given client request is denied at authorization, an exception is thrown.

The handler is wrapped as an Axis module - `shib-authorization`. The module is distributed as a JAR package (`shib-authorization.mar`). It contains the following files:

- module class (`com.gridwisetech.shibboleth.auth.AuthorizationModule`)
- authorization handler class (`com.gridwisetech.shibboleth.auth.AuthorizationHandler`)
- module deployment configuration file (''module.xml'')

Module class allows additional configuration of how the module works. In this case we kept to the standard module behavior.

5.1.1.5Apache Axis2 service

Apache Axis web services, or shortly Apache Axis services, provide a means for running applications by Apache Axis. These services follow web services approach. The service is defined as a single class (here: `userguide.example2.MyService`). It is a modified example service from Apache Axis2 distribution. It acts the similar way as standard TCP echo service (echoes the message it has received as input). The service is configured via `services.xml` file. This file assigns the service to a given module (here: *shib-authorization*).

5.1.1.6Apache Axis2 client

In order to be used the service needs a client. We use a simple client (`userguide.clients.SayHelloClient`). It is a modified example from Apache Axis2 distribution. It sends a SOAP message comprising security data and message for the service.
The above mentioned service and client are described in more detail in *Usage example* section.

5.1.1.7ShibAxis at work

Flow of events while ShibAxis is enabled looks as follows:

- User runs the client (here: `SayHelloClient`) with three input parameters (Shibboleth handle, Shibboleth IdP URL and message)
- Client sends SOAP message to the service (here: `MyService`)
- Apache Axis2 processes subsequent phases and handlers within phases
- Authorization phase processing starts
- Authorization handler processing starts (`invoke` method)

- ShibAuthAPI is used to ask Shibboleth for making an authorization decision
- ShibAuthAPI returns true (*authorized*) or false (*unauthorized*)
- If authorization was denied, an exception is thrown
- Security data (Shibboleth handle and IdP URL) are stripped from SOAP message
- Requested method (here: `sayHello`) on web service is executed
- Method's result (here: echoed input the web service has received) is sent to the user

5.1.1.8Usage example

Sample web service (`userguide.example2.MyService`) uses ShibAxis. Sample client (`userguide.example2.clients.SayHelloClient`) connects to the service that is protected by ShibAxis (here: `MyService`).

On the client side two additional parameters (SOAP tags) are required:

- *handle* (Shibboleth handle)
- *idp* (Shibboleth IdP URL)

Client-side code looks as follows:

```
public class SayHelloClient {

   private static EndpointReference targetEPR = new
EndpointReference("http://localhost:8080/axis2/services/MyService?wsdl");

   public static OMElement getSayHelloElement(String msg) {
       OMFactory factory = OMAbstractFactory.getOMFactory();
       OMNamespace namespace = factory.createOMNamespace("http://example2.userguide",
"example2");

       OMElement sayHelloNode = factory.createOMElement("sayHello", namespace);

       OMElement handle = factory.createOMElement("handle", namespace);
       handle.setText("_3a4854248e6cf243ac833b09755286e7");
       sayHelloNode.addChild(handle);

       OMElement idp = factory.createOMElement("idp", namespace);
       idp.setText("https://virolab.gridwisetech.pl/shibboleth-idp");
       sayHelloNode.addChild(idp);

       OMElement message = factory.createOMElement("message", namespace);
       message.setText(msg);
       sayHelloNode.addChild(message);

       return sayHelloNode;
   }

   public static void main(String[] args) {
       try {
               OMElement payload = SayHelloClient.getSayHelloElement("Hello, there");

               ServiceClient serviceClient = new ServiceClient();

               Options options = new Options();
               options.setTo(targetEPR);
               options.setAction("urn:hello");
               serviceClient.setOptions(options);

               serviceClient.sendReceive(payload);
       } catch (AxisFault axisFault) {
              axisFault.printStackTrace();
       }
```

```
    }
```

The above mentioned code sends the following SOAP message to the relevant web service (''http://localhost:8080/axis2/services/MyService?wsdl''):

```
 <?xml version='1.0' encoding='utf-8'?>
    <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
       <soapenv:Header>
         <wsa:To>http://localhost:8080/axis2/services/MyService?wsdl</wsa:To>
         <wsa:MessageID>urn:uuid:2D7E21A188C3F655CA1204037206987</wsa:MessageID>
         <wsa:Action>urn:hello</wsa:Action>
       </soapenv:Header>
       <soapenv:Body>
         <example2:sayHello xmlns:example2="http://example2.userguide">
           <example2:handle>_eeacb3d5693477491b6c2523c1380f8b</example2:handle>
           <example2:idp>https://virolab.gridwisetech.pl/shibboleth-idp</example2:idp>
           <example2:message>Hello, there</example2:message>
         </example2:sayHello>
       </soapenv:Body>
    </soapenv:Envelope>
```

ShibAxis, Apache Axis2 Shibboleth authorization module to be precise, uses *handle* and *idp* tag content to make an authorization decision (i.e. grant or deny access to a given user). These tags are stripped after use. In other words, web service does not receive any authorization data inside SOAP message.

Service-side code looks as follows:

```
  public class MyService {
    private static final Log log = LogFactory.getLog(MyService.class);

    public OMElement sayHello(OMElement element) throws XMLStreamException
    {
      if (element.getLocalName().equals("message")){
        log.info("MyService message is: '" + element.getText() + "'");
      }
      return element;
    }
  }
```

Web service echoes the message it has received.

Shibidpclient

Shibboleth has been created to protect Web sites, but this functionality does not fulfill all requirements of ViroLab. In some situations we need to authenticate user without accessing any web page directly by the human (e.g. EPE handle requesting plugin), or even without involving any human interaction at all.

For such use, we have created Shibidpclient library, which is able to get Shibboleth handle from SSO part of IdP. It is achieved by accessing SSO part of given IdP pretending to be SP, authenticating using given credential, and then parsing returned HTML to extract SAML response assertion. After getting it library is using OpenSAML to validate its integrity and to extract the handle itself.

Shibboleth 1.3 specification does not regulate any authentication mechanism for SSO protection – it is clearly stated that it is left entirely in hands of person in charge of deployment of Shibboleth infrastructure. In ViroLab it has been decided that we are going to use Basic HTTP authentication, and this type of authentication is currently supported by the library.

Connection to HTTP server is being handled with the use of SSL protocol, so transmission is secured.

Library does not perform authorization which might be performed e.g. by ShibAuthAPI .

User of this library must create instance of `DefaultShibIdProviderClient` class and run its method called `acquireHandle`(), passing as the argument instance of the class being implementation of the following interface:

```
public interface HandleRequesterConfig {
     public String getLogin();
     public char[] getPassword();
     public IdProviderEntry getIdProviderEntry();
     public String getTrustStoreFilename();
     public char[] getTrustStorePassword();
}
```

Methods `getLogin()` and `getPassword()` of the implementation should return credentials for user currently being authenticated, `getIdProviderEntry()` should return the instance of `IdProviderEntry` class (used to configure IdP appropriate for supplied credentials).

Constructor of this class requires that user supply following parameters:

- `ssoUrl` – URL for SSO part of IdP at user's HO
- `shireUrl` – URL of SHIRE at SP we're acting as, it does not have to be real SP but it must be configured as valid SP in IdP's methadata,
- `serviceProviderUri` – name/URI of SP we are acting as, like befor it does not have to be real
- `identityProviderUri` – at present it is not used and should be set to empty string, it might be needed in the future

Methods `getTrustStoreFilename()` and `getTrustStorePassword()` are used to specify location and credentials for standard Java certstore with trusted certificates used as SSL certificates by all HTTP servers hosting IdPs .

The `acquireHandle()`method returns String reference which points to Shibboleth handle if authentication succeeded or is `null` if it fails.

Security Integration Points

The modules available in security framework which was described before, are leveraged in the several components of Virtual Organization.

GSEngine which is the part responsible for running the ViroLab experiments uses modules for securing Web Services and ShibSVN. *ExpRepo* repository of experiments is secured using ShibSVN

Integration of Security with GSEngine

Since GSEngine constitutes an entry point to virtual laboratory it is the first unit that requires authentication and authorization in application execution scenario presented in Figure 52.

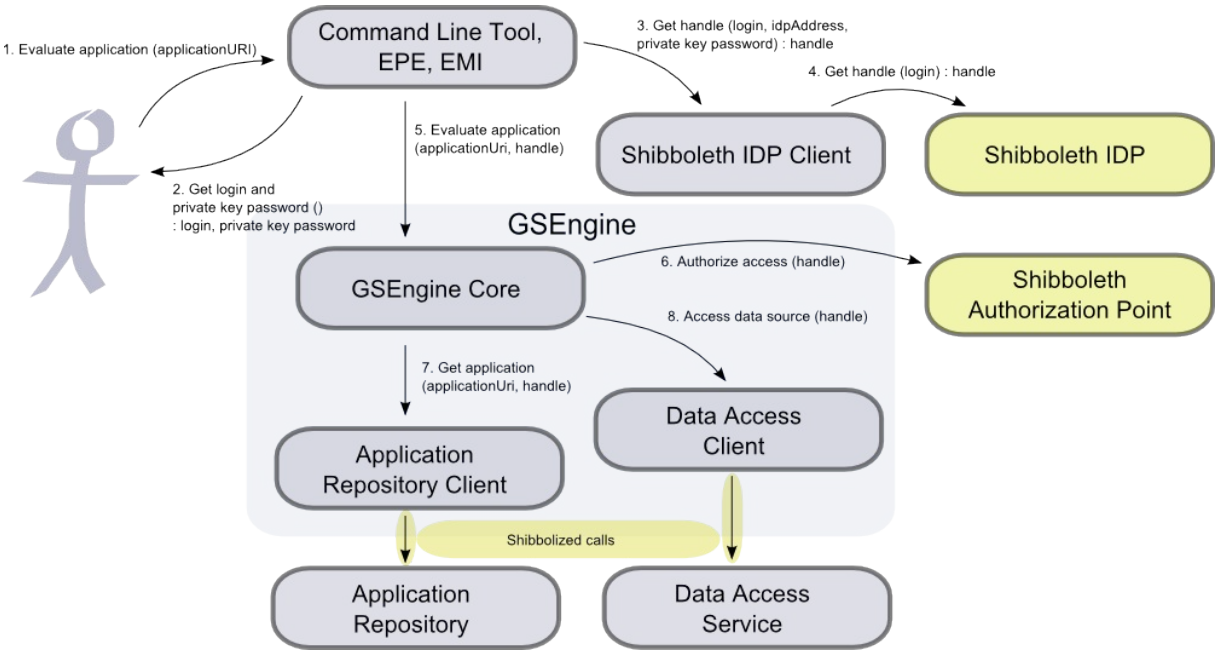


Figure 52 Integration between GSEngine and Shibboleth security infrastructure

GSEngine expects that evaluation requests contain a handle, so first the clients of GSEngine, namely GSEngine Command Line Tool, Experiement Planning Environment or Execution Management Interface have to acquire one with Sibboleth IDP Client. Sibboleth IDP Client, in turn, in order to acquire a handle needs IDP address and user's private key. Therefore, just after receiving application evaluation request (denoted as step 1 in Figure 52) Command Line Tool, EPE or EMI ask user for login and private key password (step 2). Then, Shibboleth IDP Client can be used to acquire a handle from IDP of Home Organization chosen by user (step 3). Shibboleth IDP Client connects with Shibboleth IDP using decrypted user's private key (step 4) and once the user gets authenticated it returns a valid handle.

Given with handle GSEngine is called to execute application (step 5). However, before execution starts the authorization has to take place. In order to do so GSEngine asks Shibboleth Authorization Point for attributes associated with a handle provided (step 6). Then, basing on acquired attributes the access is permitted or denied.

Once the access is permitted a handle is being kept in application context. In order to download application code from Application Repository the Application Repository Client is called with the handle (step 7) so it can access Shibboleth-enabled Application Repository. Afterwards, during application when Data Access Client or any other runtime library of GSEngine can access the handle kept in application context and use it in order to access Shibboleth-enabled services such as Data Access Service (step 8).

Integration of Security with ExpRepo (ShibSVN)

Experiment Repository is a normal Subversion repository which can be accessed by standard Tigris Subversion client or other 3rd party tools such as a web browser. In this repository the experiments are stored as files and access to these files is restricted by Apache authentication - ShibSVN.

When a user logs on to the portal, he/she receives a handle from his/her Home Organization Identity Provider (IdP). If user wants to access Experiment Repository from the portal to get new experiments, the portal sends Handle and IdP URL to the Subversion Experiment Repository provider and this provider checks if this user can access this repository. This process is done internally by Apache server, which hosts Subversion repository. Apache contacts ShibRPC XML-RPC service and sends user handle and IdP URL. This service connects to user IdP and retrieves user attributes and checks them against access policy file.

In that file administrator can define who can access this repository and who cannot. If user attributes match required by policy file attributes, user is allowed to access that repository.

Authorization to the repository can be also done without the GridSphere portal. To access Experiment Repository from the third party client e.g. using Tigris SVN user has to type:

```
svn co https://location.of.experiment/repository/svn/trunk
```

In password field user has to pass handle and the URL of his IdP separated by '#' character.
Example:

```
_7af9bdcad7cf26b0cb6e155eebd4a8ed#https://virolab.gridwisetech.pl/shibboleth-idp
```

This handle and IdP can be easily obtained from GridSphere Shibboleth information portlet.

Integration of Security with DAS

In order to guarantee that only persons who are known to the ViroLab virtual environment can access certain data sets, additional security mechanisms, in particular access control mechanisms for all integrated data resources [Assel06], need to be defined and applied to protect the sensible information from any abuse.

Therefore, the Data Access Services (DAS) have been connected and adapted with the overall ViroLab security infrastructure. The services provide capabilities to perform authorization decisions based on current user attributes obtained from corresponding home organizations (IDPs). These attributes are requested for each unknown DAS user. An unknown user is identified by the DAS for example if a recent identity token has expired or the user has invoked the service for the first time and with it created a new service resource. This unique service instance is created by a so-called service instance factory, a typical design pattern when dealing with stateful web services (GT4 services). This factory creates and instantiates a new service resource by defining a one-to-one relationship through a specific endpoint reference (EPR) assigned with a unique key. Using this EPR returned by the factory, the client can now invoke the service's operations through the proper service.

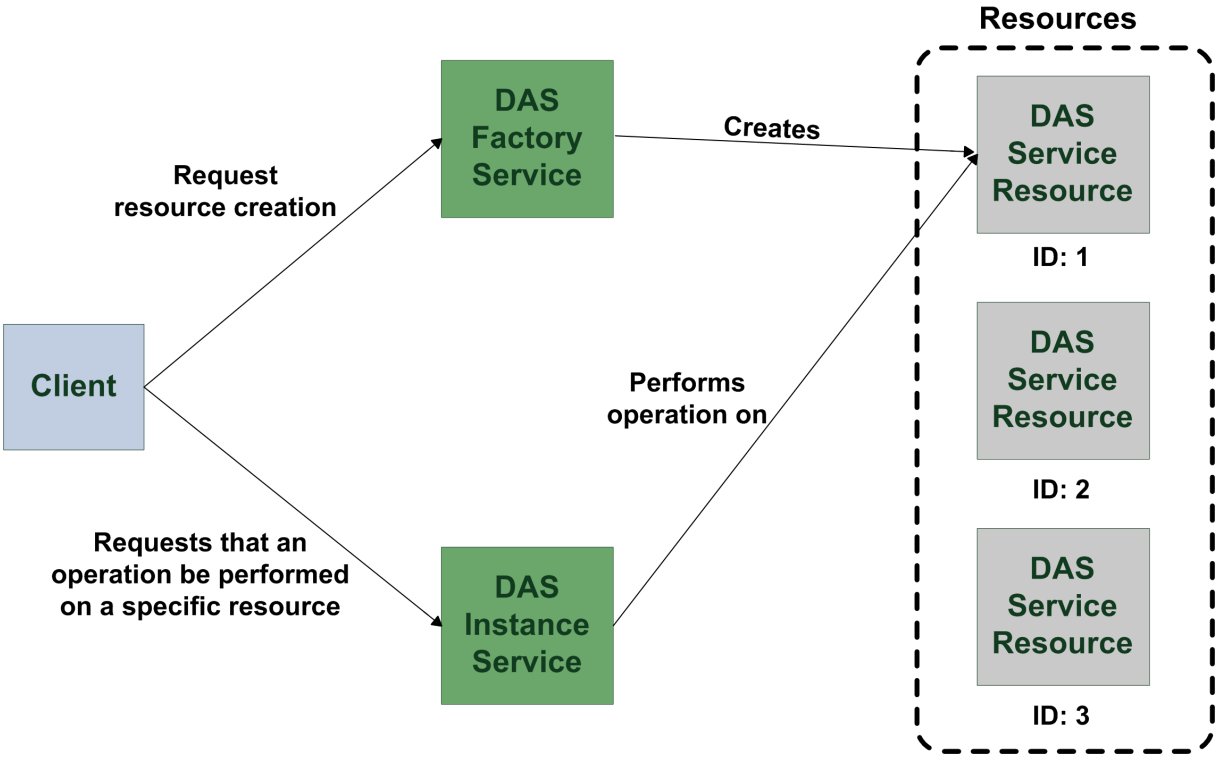


Figure 53 The WS-Resource factory pattern

After this distinct instance has been linked with exact one user, the DAS itself need to be initialized. This means before one can access the integrated resources, the services will perform a user authorization to distinguish between available and accessible resources - Available ones can only be displayed while accessible ones can be queried and/or browsed (compare with figures in section 3). The distinction between these two types (states) of resources specifies the user's rights in terms of having the permission to query data from a particular data source or that the access to the resource is denied. The final decision whether one is allowed or not, is based on the current user attributes that have been released from the user's home organization. These attributes are

requested via a so-called attribute request by sending the user's handle (identity token) to the corresponding IDP which initially checks whether the handle is valid and with it the user is known to the system, and secondly, which attributes are designated to be forwarded to the requesting party. Having received the user attributes, the DAS will use them to authorize the user. The authorization is currently performed by simply comparing these attributes with some pre-defined and hard-coded rules specifying the access to certain resources. The following two examples show some very simple conditions that only consider two specific user attributes (viroLabRole and organization).

```
(hasAttribute(viroLabRole, "doctor") && hasAttribute(organization,
"University of Brescia")) ||

(hasAttribute(viroLabRole, "virologist") && hasAttribute(organization,
"University Medical Center Utrecht"))
```

To guarantee more dynamicity for specifying such access rules, the hard-coded conditions will be replaced with a more flexible solution, by using so-called attribute-based access control policies. Those policies can be individually designed and especially modified by the respective data resource owners themselves. Therefore, a central policy storage and interpretation point is required that supports the dynamic change and analysis of pre-defined access control policies. The concept of Policy Decisions Points (PDP) perfectly meets those requirements for modifying and evaluating policies on-demand and on-the-fly according the sudden circumstances. The usage of a PDP for performing user authorization has been already tested but still needs to be integrated with the corresponding services' interfaces. Apart from that, a nice graphical user interface is being developed in parallel that allows these ad-hoc modifications for existing access control policies and the deployment of newly designed ones.

To explain the dependencies and interactions between several components of the virtual infrastructure, and to highlight the aforementioned principles, the general user authentication and authorization concept, in particularly for ViroLab's Data Access Services, is depicted in the following figure.

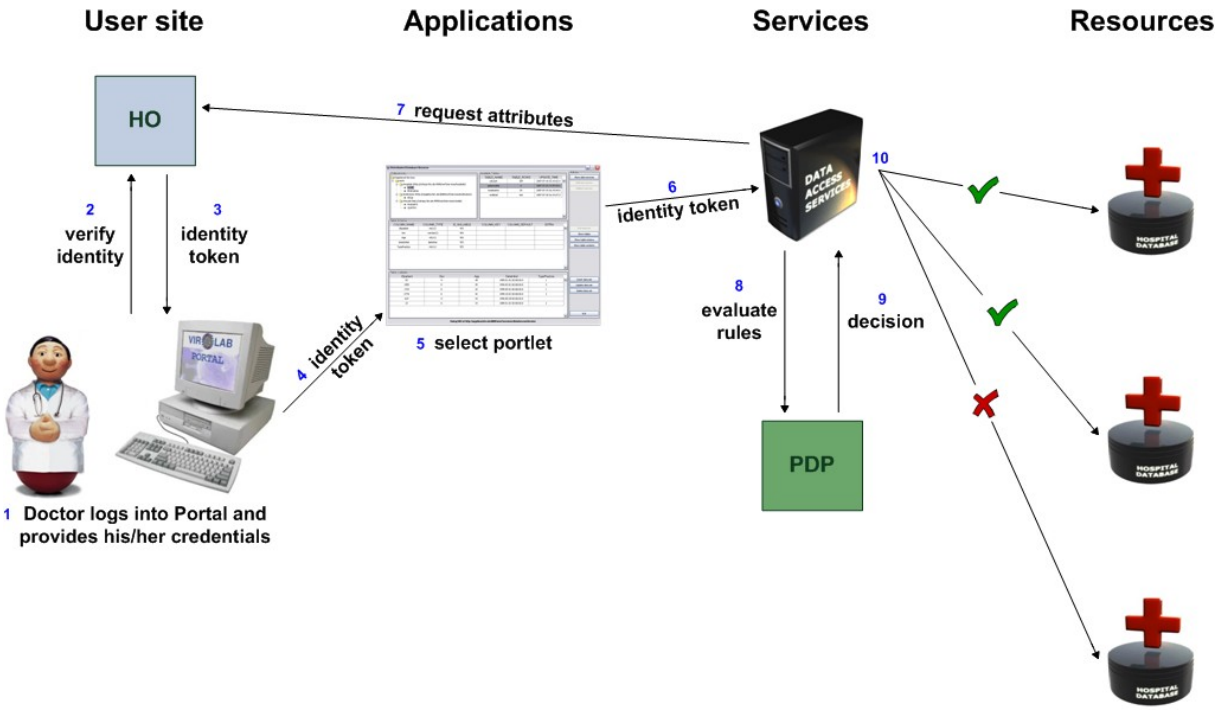


Figure 54 Common ViroLab use case

To present a more technical view on the abovementioned workflow, Figure 55 displays the same scenario in form of a typical sequence diagram showing all relevant message and data flows during the submission of a data query. In particularly, relevant components of the security infrastructure together with respective DAS parts are highlighted here.

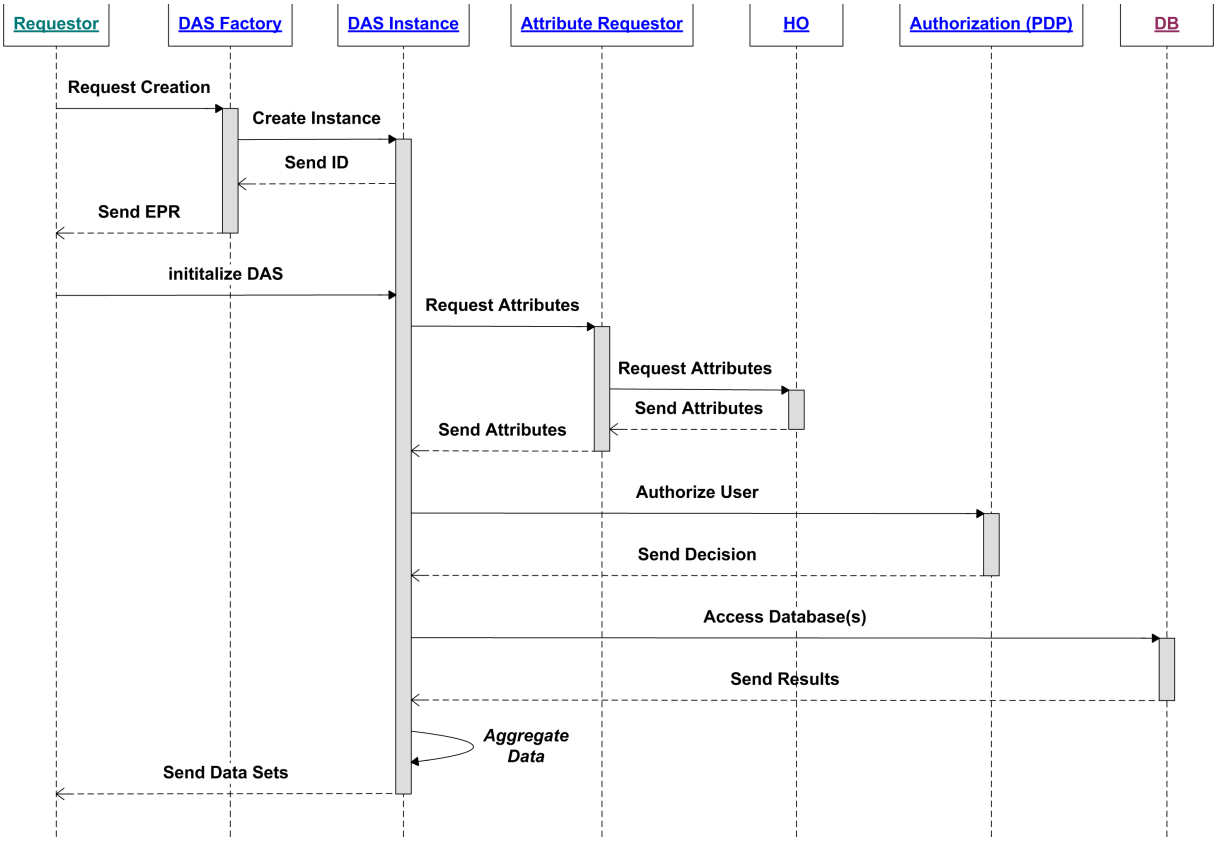


Figure 55 Components involved in a data access process flow

Basically, the Attribute Requestor Library (see section 5above) is the only required component of the overall security infrastructure which is directly interacting with the DAS. The library provides common interfaces to request user attributes from corresponding home organizations (HO). Therefore, DAS need to pass the user's handle as well as the current IDP address to the particular library functions. These parameters are used to determine the user's HO before they are sent to the respective IDP, which returns with a list available attributes. In case of the DAS, the obtained attributes are then internally used (in the future, an external PDP will be asked) to authorize a user for certain resources.

5.1.2Integration of Security with EPE (Authentication plugin)

The Experiment Planning Environment is used, by the experiments developers, for creating, executing and sharing experiments. To perform these operations it is integrated with the GSEngine and the Experiment Repository components.

As it was mentioned above, in order to ensure security they need to be 'shibbolized'. Therefore it is necessary to augment the functionality of the EPE with an authentication and authorization mechanism. In order to fulfill this requirement an authentication plugin has been developed. It is integrated with the EPE environment by exposing an activator on the workbench toolbar which leads to a dialog window (Figure 56). Its main

goal is to provide a user-friendly way for retrieving a Shibboleth handle from a given Id provider on behalf of developer.

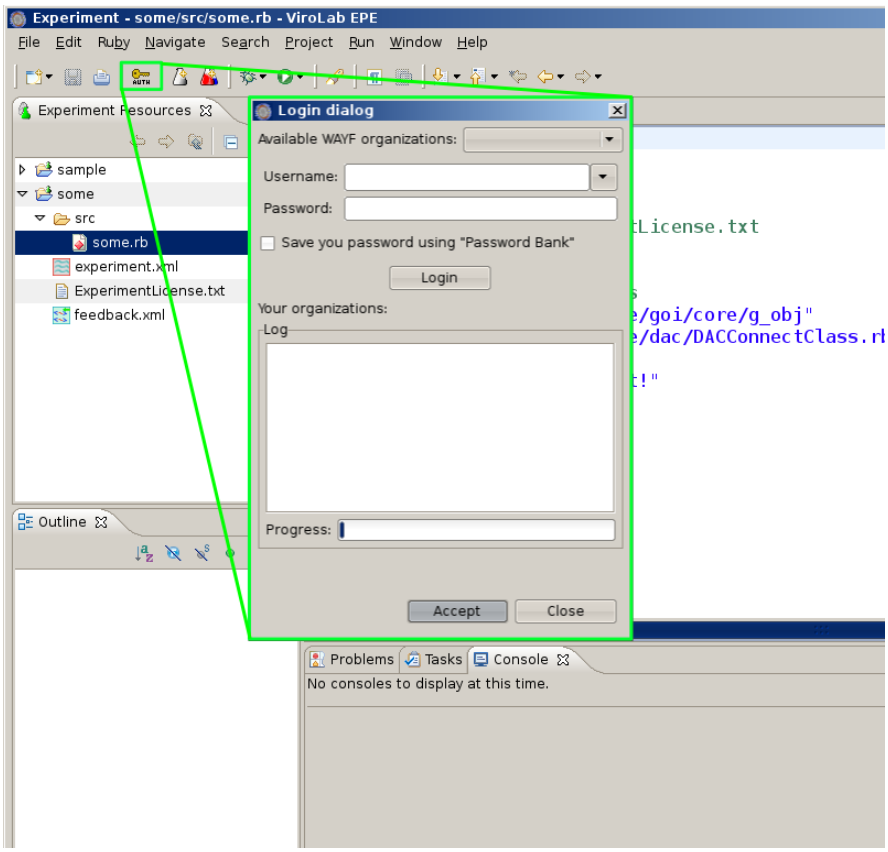


Figure 56: The authentication plugin

Clicking the toolbar button opens the authentication dialog. To retrieve the handle the developer has to select home organization that should perform authorization process and fill the username and password text fields. After clicking the "Login" button the authorization process starts and if success the handle will be returned and cached. From this moment on, the developer is granted with access to the Experiment Repository and the GSEngine runtime system. The authentication plugin will pass the handle to both the Experiment Repository client and the GSEngine client. It is done transparently thus the developer could stay focus upon his/her main objectives.

Planned Functionalities of the Security Framework

Security layer in the version 1 provides several functionalities, which make flexible creating of Virtual Organizations possible. The modules at the Home Organization site are responsible for authentication. At the Portal site there is the integration point of all HOs (WAYF) and at the Resource Provider site there are all modules which provide authorization functionalities.

At the current state of work, the needed security modules have been identified, designed, implemented and went through preliminary Proof-of-

Concept testing (e.g. IdP, WAYF, ShibAuthAPI, ShibRPC) Further development should not add new features but rather focus on flexible and practical access to this complex functionality. Examples of system aspects we would like to focus on include:

- User-friendly user interface for editing resource access rights. Resource owners would have possibility for flexible defining and editing access rules.The procedure for adding new users to Home Organizations and especially "Public Home Organization" (database for users who do not have their own HO and IdP)
- User interface for adding and editing users in standard HO
- Some types of resources, such as GSEngine, require more in-depth integration at the level of access policies, mapping users to small-grained resources such as files.

Security – Conclusions

The content of this project deliverable results from the earlier assumption of following the distributed authentication approach in the ViroLab security, in particular based on Shibboleth implementation. The rationale for this decision has been subject to detailed discussion in previous deliverables [D2.1], [D2.2]. Although the goal of this deliverable is to demonstrate the working proof-of-concept deployment of our technology, it would be left incomplete without a reference to the valuable discussion on the pros and cons of the distributed authentication approach. This discussion in which our project gladly participated took part during the "*Workshop on Security and Privacy issues for bio-medical applications based on Grid middleware"* organized by the European Commission in Brussels, May 2008, for ViroLab and five other e-Health projects.

In the previous sections we have demonstrated successful integration of the virtual organization, which was the goal of this Deliverable D2.3. However beyond this goal, the security team at ViroLab achieved much more in this past year. The cross-cutting nature of security and the need for integration resulted in tight cooperation of grassroots, informal team of expert engineers, primarily from GridwiseTech, Cyfronet, University of Stuttgart (HLRS), and University of Amsterdam. The need for pioneering work on building Shibboleth-based virtual organizations resulted in additional insight in the nature of this approach and technology.

As seen from previous chapters, integration with Shibboleth was not straightforward and numerous features and libraries had to be implemented from scratch. This effort was well rewarded, as we now dispose of unique and universal software components, bridging Shibboleth with popular technologies such as Apache Axis or SVN. In spite of some restrictions of Shibboleth technology we continue to consider it a valuable approach, giving us flexibility which other paradigms could not provide. Shibboleth admittedly lacks in the area of usage cases, documentation and configuration, but this solution is continuously developed and most of the problems and restrictions are being eliminated step by step. Recently the new version (Shibboleth 2.0) was released, so we will consider to

adopt it to ViroLab.

Today, the Shibboleth is one of the main participants in the stream of federated authentication technologies. We spend significant amount of time on gaining the practical experience in using that technology in securing such sensitive resources like medical data. We identified the strengths and weaknesses of the applied solution, and gained knowledge which will lead to secure and reliable security framework. Our work is dedicated to create a stable security framework. Coupled with powerful user interface, this should remain an important impact of ViroLab.
This know-how counterbalanced by our previous, solid knowledge of alternative solutions (PKI-family / X.509 certificate-based approach), gives us unique collection of know-how which additionally strengthens and complements European proficiency in the distributed security area.

Abbreviations

Abbreviation/Term	Explanation
ViroLab	a virtual laboratory for decision support in viral diseases treatment
CCA	Common Component Architecture
DAS	Data Access Services
DEISA	Distributed European Infrastructure for Supercomputing
DOS	Domain Ontology Store
DRS	Drug Ranking System
EGEE	Enabling Grids for e-Science in Europe
EMI	Experiment Management Interface
EPE	Experiment Planning Environment
GOI	Grid Operation Invoker
GrAppO	Grid Application Optimizer
GRR	Grid Resources Registry
GSEngine	GridSpace Engine
GUI	Graphical User Interface
HO	Home Organization
IDE	Integrated Development Environment
IdP	Identity Provider
JMX	Java Management Extensions
LCG	LHC Computing Grid
QUaTRO	Query Translation Tools
PDP	Policy Decision Point
PROToS	Provenance Tracking System
RAD	Rapid Application Development
SSO	Single Sign-On
SVN	Subversion
TLS	Transport Level Security
VL	Virtual Laboratory
VO	Virtual Organization
WAYF	Where Are You From
WP	Workpackage
WS	Web Service
WSRF	Web Services Resource Framework

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.

[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 Suvey, Design and Workpackage Spcification, ViroLab Project Consortium, 2006

[D2.2] ViroLab Project D2.2 - Architecture for presentation layer. VO pilot deployment with basic middleware for data access, resourcemanagement , and information systems - report and demonstration, 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

[D3.3]	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
[D3.4]	Virolab Project, Deliverable D3.4: Integration of presentation layer and session manager, workflow provenance system and process flow template - report and demonstration
[D3.3-A2]	ViroLab Project, Deliverable D3.3 Appendix 2: ViroLab Virtual Laboratory:Experiment Developers' Manual
[EPEMI]	Environment for collaborative development and execution of virtual laboratory applications, Wlodzimierz Funika, Daniel Harezlak, Dariusz Krol, and Marian Bubak; Accepted to the ICCS'08 conference, June 2008, Krakow, Poland
[EXP]	ViroLab experiments repository https://svn.gforge.hlrs.de/svn/virolab/trunk/experiments
[MOCCA]	Maciej Malawski, Dawid Kurzyniec, and Vaidy Sunderam, MOCCA – towards a distributed CCA framework for metacomputing. In Proceedings of the 10th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS2005), 2005
[POM]	Maven - Introduction to the POM http://maven.apache.org/guides/introduction/intorduction-to-the-pom.html
[SeRQL]	The SeRQL query language (revision 1.2), http://www.openrdf.org/doc/sesame/users/ch06.html
[SESAME]	User Guide for Sesame 2.0, http://www.openrdf.org/doc/sesame2/2.0.1/users/index.html
[Shibboleth]	Shibboleth, http://shibboleth.internet2.edu/
[TBCA]	A Tool for Building Collaborative Applications by Invocation of Grid Operations, Maciej Malawski, Tomasz Bartyński, Marian Bubak; Accepted for the ICCS'08 conference.
[VIROLAB-VL]	The ViroLab Virtual Laboratory Website. http://virolab.cyfronet.pl/
[VLINV]	Invocation of Grid Operations in the ViroLab Virtual Laboratory, Tomasz Bartyński, Maciej Malawski, Marian Bubak; in Cracow Grid Workshop 2007 Workshop Proceedings, pp.59-64, ACC CYFRONET AGH 2008