

## Deliverable 3.3 Appendix 1

### ViroLab Virtual Laboratory: Experiment Users' Manual

<b>Project Start:</b>	01-03-2006
<b>Project Duration:</b>	36 Months
<b>Priority area</b>	2.4.11
<b>Contract No.:</b>	INFSO-IST-027446
<b>Website:</b>	<a href="http://www.virolab.org">http://www.virolab.org</a>

<b>Due-Date:</b>	31-08-2007
<b>Delivery:</b>	13-09-2007
<b>Lead Partner:</b>	CYFRONET
<b>Coordinator</b>	UvA, Prof. Dr P.M.A. Sloot
<b>Dissemination Level:</b>	Public
<b>Status:</b>	Final
<b>Approved:</b>	Quality Board, Steering Committee
<b>Version:</b>	1.0

## Log of Document

Version	Date	Changes Summary	Authors
0.1	29/08/2007	Initial version of the manual	Tomasz Gubala
0.2	31/08/2007	Contribution from developers	Robert Pajak, Dariusz Krol, Marek Kasztelnik, Piotr Regiel, Eryk Ciepiela, Tomasz Bartynski, Joanna Kocot, Piotr Nowakowski
0.3	05/09/2007	Section 5.1 checked, refined, supplemented and formatted	Eryk Ciepiela
0.4	06/09/2007	Added EMI section	Tomasz Gubala
0.5	06/09/2007	Section 5.1 rechecked and reformatted	Eryk Ciepiela
0.6	07/09/2007	Quatro section added	Kuba Wach
1.0	13/09/2007	Manual title changed, minor changes	Marian Bubak

# TABLE OF CONTENTS

<b>COPYRIGHT NOTICE.....</b>	<b>5</b>
<b>1. INTRODUCTION.....</b>	<b>6</b>
1.1. TARGET AUDIENCE.....	6
1.2. MORE INFORMATION.....	6
<b>2. VIRTUAL LABORATORY ENVIRONMENT DESCRIPTION.....</b>	<b>8</b>
2.1. EXPERIMENT PIPELINE IDEA.....	8
2.2. DEFINED CLASSES OF USERS.....	9
2.3. FUNCTION OF THE EXPERIMENT REPOSITORY.....	10
<b>3. EXPERIMENT MANAGEMENT INTERFACE USER'S MANUAL.....</b>	<b>12</b>
3.1. ACCESSING AND RUNNING EXPERIMENT MANAGEMENT INTERFACE.....	12
3.1.1. <i>Accessing EMI</i> .....	12
3.1.2. <i>EMI portlets explained</i> .....	12
3.1.3. <i>Experiment Repository configuration and browsing</i> .....	13
3.1.4. <i>Experiment execution</i> .....	15
3.2. SOURCE CODE ACCESS, BUG REPORTING AND AUTHORS CONTACT INFORMATION.....	16
<b>4. QUERY TRANSLATION TOOLS USER'S MANUAL.....</b>	<b>18</b>
4.1. ACCESSING AND RUNNING QUERY TRANSLATION TOOLS.....	18
4.1.1. <i>QUaTRO step-by-step</i> .....	18
4.1.2. <i>QUery TRanslation tOols (QUaTRO) GUI Features Overview</i> .....	21
4.2. AUTHORS CONTACT INFORMATION.....	23
<b>5. GRIDSPACE EXPERIMENT USER MANUAL.....</b>	<b>24</b>
5.1. INSTALLATION AND USE.....	24
5.1.1. <i>Installation</i> .....	24
5.1.2. <i>Working with gsengine command line tool</i> .....	25
5.1.3. <i>Working with gsquery command line tool</i> .....	31
5.1.4. <i>Working with GSEngine API</i> .....	31
5.2. SOURCE CODE ACCESS, BUG REPORTING AND AUTHORS CONTACT INFORMATION.....	35
<b>ABBREVIATIONS.....</b>	<b>37</b>
<b>REFERENCES.....</b>	<b>40</b>

## List of Figures

FIGURE -1: THE GENERIC, SIMPLEST VERSION OF THE EXPERIMENT PIPELINE.....	8
FIGURE -2: EXPERIMENT PLAN BEING CONCEIVED, SHARED AND DEVELOPED.....	10
FIGURE -3: THE PROCESS OF RELEASING AND USING EXPERIMENT PLAN.....	11
FIGURE -4: MAIN EXPERIMENT MANAGEMENT INTERFACE (EMI) WINDOW.....	12
FIGURE -5: THE EXPERIMENT REPOSITORY CONFIGURATION WINDOW.....	13
FIGURE -6: EXAMPLE OF A LIST OF EXPERIMENTS STORED INSIDE A REPOSITORY,.....	14
FIGURE -7: EXPERIMENT DETAILS IN THE EXPERIMENT CONTEXT PORTLET.....	14
FIGURE -8: FEEDBACK WINDOW IN THE EXPERIMENT CONTEXT PORTLET.....	15
FIGURE -9: EXPERIMENT LICENSE TEXT IN THE EXPERIMENT CONTEXT PORTLET.....	15
FIGURE -10: THE EXPERIMENT EXECUTION FIELD.....	15
FIGURE -11: AN EXAMPLE OF AN EXPERIMENT EXECUTION RESULT.....	16

## **COPYRIGHT NOTICE**

Copyright (c) 2007 by **Academic Computer Centre CYFRONET AGH**. All rights reserved.

Any use of the products described in this document is subject to the terms stated in the GPL license agreement: <http://opensource.org/licenses/gpl-license.php>.

## **1. INTRODUCTION**

This document contains a set of manuals and tutorials for a person that would like to use experiments prepared within the ViroLab virtual laboratory. The sections inside contain instructions how access and run tools provided for a scientists. The tools are mostly in form of web-based user interfaces (sometimes stand-alone, sometimes displayed inside the ViroLab Portal). They include the Experiment Management Interface that allow for experiment download and execution, the Query Translation Tools that help the user form questions on past experiments and results provenance and finally the GridSpace Engine manual for the users that would like to access the laboratory runtime in other, more complex ways.

### **1.1. TARGET AUDIENCE**

The construction of the ViroLab Virtual Laboratory assumes a smooth collaboration of various types of users, the most important of which are scientific programmers that design future experiments and prepare them for use and the researchers themselves, who would like to execute the prepared experiments to obtain interesting scientific results. This document is intended for the later.

The proper and effective use of experiment user's tools does not require deep knowledge of technicalities of the laboratory itself. The basic grasp of the important notions, mechanisms and processes taking place in this collaborative space is however important for an effective cooperation with fellow researchers. The Section 2 provides this knowledge on sufficient level.

On the other side, the set of tools that are provided to the user (and that are described in this manual) includes modules of different complexity. Therefore some of them are simple to use as the tasks they perform are conceptually simple; other could require understanding of more complex ideas as they model more complex behaviour – in any case, this document provides the needed explanations of these concepts so the reading of its content should be sufficient to start using the virtual laboratory.

### **1.2. MORE INFORMATION**

This document is not the only source of information for future experiment developers. The ViroLab Virtual Laboratory web pages provide the most recent and frequently updated versions of the enclosed tutorials. Please check:

<http://virolab.cyfronet.pl>

for a thorough, complete introduction to Virtual Laboratory and its mechanisms, tools, runtime etc. In the upper right corner of the page you will find set of hyperlinks to development sites, where you may:

- obtain the latest releases of the virtual laboratory modules
- read about the development plans and future release time schedule
- report a bug or a feature request, discuss it and monitor its lifetime

The authors of this manual and the software it describes would like to ask for the assistance of all the developers that would like to use the virtual laboratory. Please don't hesitate to use the bug submission and feature request mechanism

in the virtual laboratory development web pages to suggest the authors how to refine the software. With this process the tools we provide will be more useful and productive for the future experiment developers.

## 2. VIRTUAL LABORATORY ENVIRONMENT DESCRIPTION

The **Virtual laboratory** is a set of integrated components that, used together, form a distributed and collaborative space for science. Multiple, geographically-dispersed laboratories and institutes use the virtual laboratory to plan, and perform *experiments* as well as share their results. The laboratory is prepared to support virologists, epidemiologists and clinicians investigating the HIV virus and the possibilities of treating HIV-positive patients. Although the ViroLab **Virtual Laboratory** is being built specifically for this domain of science, the conceptual solutions and the technology developed could be reused for other domains.

### 2.1. EXPERIMENT PIPELINE IDEA

The central idea behind any virtual laboratory is an *in-silico* experiment.

**Experiment** is a process that combines together data with a set of activities that act on that data in order to yield experiment results. The substrate data required for an experiment may be obtained from multiple resources in various possible forms. The activities may be manual, semi-manual or fully automatic, depending on their nature. No definite restrictions are imposed on the level of complexity of such an experiment: it might be as simple as listing data inside some remote database, or much more complex, such as a set of simulators combined together to obtain some insight into complex phenomena. Moreover, the experiments are not required to involve just a single, local machine – in fact, the power of the virtual laboratory comes from combining multiple distributed resources, dispersed over various geographical locations.

The purpose of the laboratory we present is to support collaborative work of all the people who are effectively involved in any stage of the experimentation process. For our purpose, we refer to this process as the experiment pipeline. Below is a section that explains our view of the subject.



**Figure -1: The generic, simplest version of the experiment pipeline.**

Figure -1 presents the simplest picture of the experiment pipeline. First of all, one has to decide what an experiment is about and how it should proceed. The part of the pipeline concerning design of the future experiment process is called experiment planning. During that phase the user has to decide upon the main subject of the planned experiment, its intended results and the means by which these results should be obtained. Such a detailed description of the experiment results in experiment execution (the middle part of the process). During this phase the user performs the experiment according to the plan developed at the planning stage, using all the resources provided to that user within the virtual laboratory. The usual outcome of such execution is the result of the experiment. Since the result itself is of the highest importance for the user, special attention



is given to it in the last phase of the pipeline, called result management. Here the outcome of the experiment may be evaluated, described and stored. Given the strong collaboration aspect of a virtual laboratory, the results can also be shared among the users of the laboratory. This stage concludes a single experiment pipeline run.

It should be stressed that no single part of this pipeline has to be a one-off activity. It is very probable, and in fact expected, that various stages will be repeated in order to achieve the correct effect or desired quality.

## 2.2. DEFINED CLASSES OF USERS

As the virtual laboratory is meant to combine together people of various levels of expertise, we define two main classes of users who take part in the experiment pipeline. Their descriptions are provided here, while another class of users (sharing a common phase in the experiment pipeline) will be introduced later on.

**Experiment developer** is a person who designs experiments in a specific domain (like e.g. virology). First of all, this means that the person is skillful enough to design and denote the way an experiment should proceed. Apart from technical skills, the developer also possesses a certain level of domain-related knowledge to understand the nature of the processes the experiment should model – otherwise, the designed experiments will never be valid. The developer uses dedicated tools to – among others – search for available data sources, combine them with suitable computational activities and present the results in an appropriate form. The aim of the virtual laboratory is to give experiment developers a set of powerful tools making their task easier, while at the same time not constraining their skill and creativity in any way.

**Experiment user** is any person who runs a previously prepared experiment in order to obtain results. The user may or may not be involved in the process of experiment preparation – in the latter case it is probable that (future) experiment users would support developers with their expert knowledge about the modeled phenomenon. The main objective of the experiment user is to obtain valuable results that answer important scientific questions.

The person who develops the experiment requires some information on the resources needed to perform the future experiment. According to the definition, the experiment can involve both remote data sources and activities that need to be performed on the data in order to achieve the final result. In the end, the experiment, successfully completed, should be able to provide experiment results.

**Experiment result** in its most general meaning covers anything that is produced in the course of experiment execution. It could consist of some textual information, a generated image or a movie, a URL link to further information—almost anything. While an experiment outcome may be difficult to quantify, such a quantification is usually useful for easier management of those results. There are no strict rules here — common sense of developers and users should be applied to decide what forms a separate result (for instance: a single image, a table of illness classifications, a file with a database table snapshot...) Since the result of an experiment is usually of the highest importance for the experiment user, the virtual laboratory devotes special attention to the management and post-

experiment activities which act upon results (such as sharing, describing, storing etc.).

### 2.3. FUNCTION OF THE EXPERIMENT REPOSITORY

The purpose of the Experiment Repository within the ViroLab Virtual Laboratory is to store and provide experiment plans that are developed by experiment developers and that are used by experiment users. From this perspective the repository plays a meeting place for these two groups of users of the laboratory - they share among themselves the experiment plans.

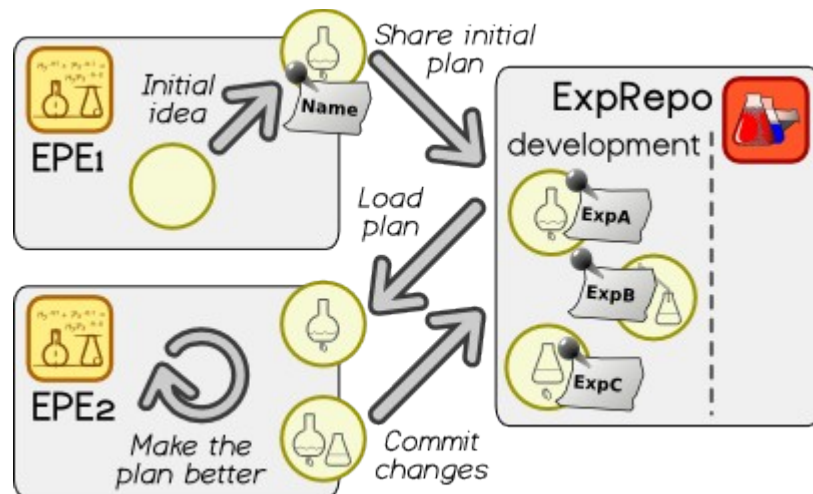


Figure -2: Experiment plan being conceived, shared and developed.

Usually, a typical life cycle of an experiment plan involves:

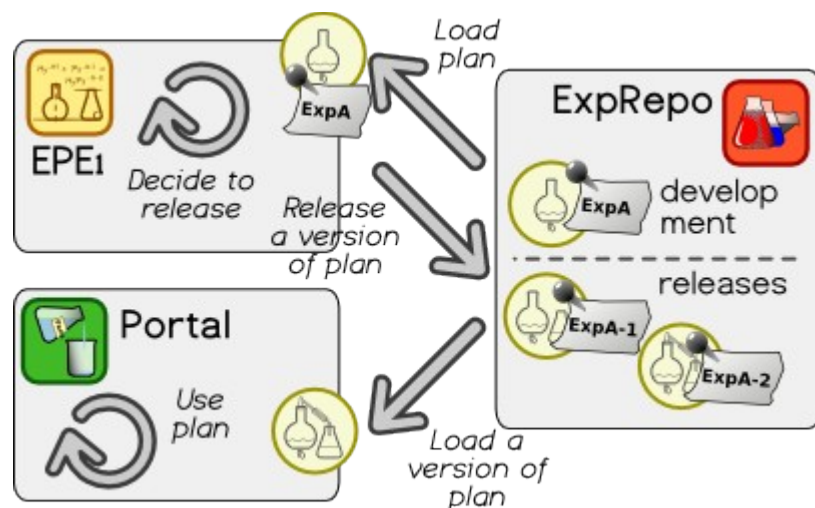
- initial conception and establishment of a new experiment plan
- shared, cooperative development of the plan by developers
- first release of the plan
- first uses (executions) of the plan by users
- gathering feedback and identifying shortcomings
- further refinement, next version releases
- and so on and so forth...

The Figure -2 shows us the first stages of this process. One developer has an idea about new experiment that could be developed. She puts those initial thoughts in form of a sketch of an experiment script and shares that with fellow developers through the Experiment Repository. In terms of software engineering this activity of experiment sharing is referred to as initial import.

After that another experiment developer, interested in similar ideas, loads the experiment plan (in cooperative development this is usually called *checkout*) to his development environment and contributes his effort to make the experiment better. The new changes and additions are shared with the community through a

commit operation that essentially synchronizes the content of the repository with the latest form of experiment developed by the given developer.

At this early stage of experiment planning, the plan in its current form is only stored in so-called development section of the repository. At the moment no user is able to obtain and execute this experiment - as it is assumed to be not mature enough.



**Figure -3: The process of releasing and using experiment plan.**

After a certain amount of time and some effort put in the experiment plan development, one of its developers may decide to release a version of the experiment plan (see Figure -3). Such publication yields a copy of the current state of the plan in a dedicated space of the Experiment Repository. The two important features of the releases space (in comparison to the development space) are: it is accessible by experiment users and the releases put there are frozen with respect to further changes.

The tool provided for the experiment user is now able to see the released versions of experiments and also, on request, is able to download a designated version of given experiment plan. The plan, using the functionality of the tool, could be now executed and the user may obtain results.

### 3. EXPERIMENT MANAGEMENT INTERFACE USER'S MANUAL

#### 3.1. ACCESSING AND RUNNING EXPERIMENT MANAGEMENT INTERFACE

##### 3.1.1. Accessing EMI

Currently EMI is deployed as a set of three portlets within the Gridsphere portal. In order to access these portlets it is required to have a valid user account on the portal. For Virolab the portal is hosted at

- <https://virolab.gridwisetech.pl/>

and its administrators should be contacted to obtain an account. After successfully logging into the portal a separate tab called "EMI" holds the portlets described below (see Figure -4).

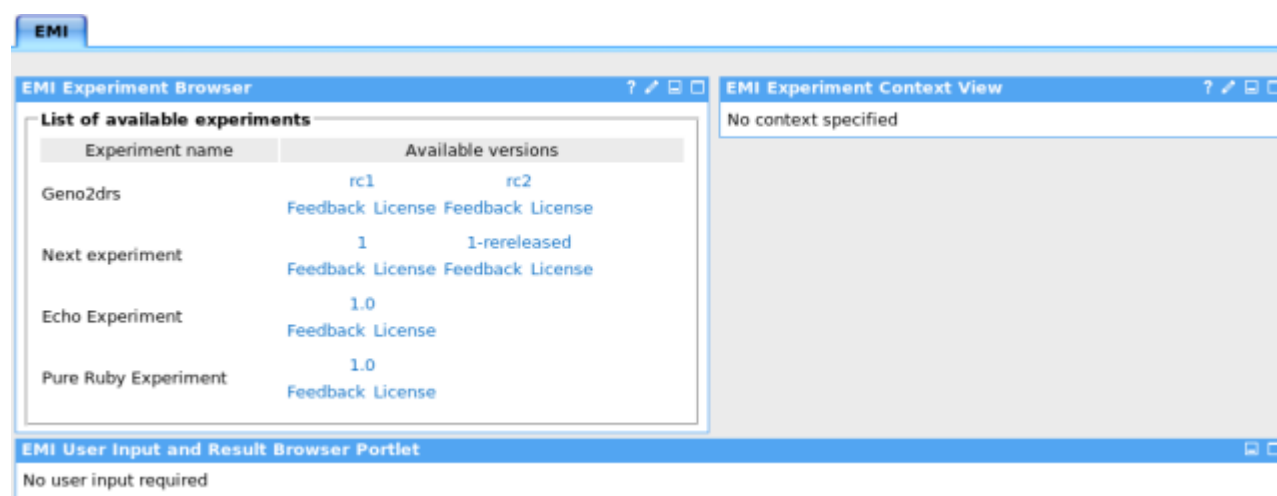


Figure -4: Main Experiment Management Interface (EMI) window.

##### 3.1.2. EMI portlets explained

As already mentioned the set of EMI portlets contains three separate portlets described in the subsections below.

###### Repository portlet.

The *Repository Portlet* allows for experiment browsing by contacting external experiment repositories and downloading a list of available experiments with detailed information (experiment versions, licenses and user feedback) and presenting the list to the user.

###### Context portlet.

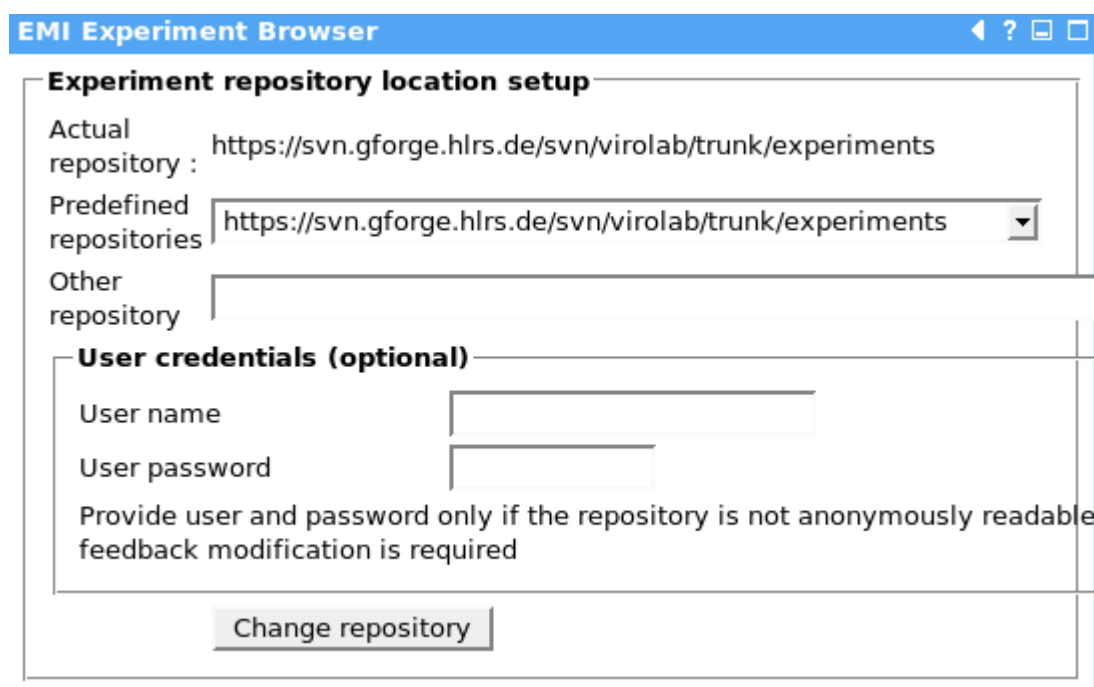
The *Context Portlet* is responsible for presenting experiment details to the user as well as for experiment execution and user feedback submission. The presented content is specific for a certain experiment context which is selected from within the *Repository Portlet*.

###### Result and user input portlet.

The third portlet is used to display the experiment output and to submit user input required by the experiment execution process. The content of this portlet is changed dynamically according to the state of the experiment execution engine.

### 3.1.3. Experiment Repository configuration and browsing

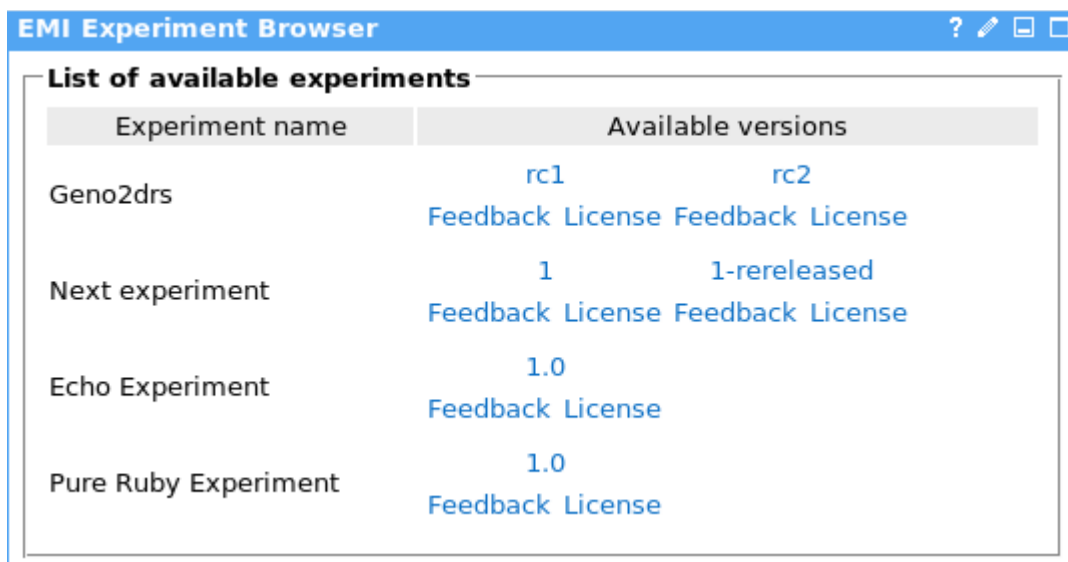
In order the repository portlet to work an experiment repository needs to be configured. This involves inputting the address of the remote repository together with user credentials (if no feedback will be sent and the repository allows for anonymous access the credentials are not necessary). This can be accomplished by using the configuration form in the edit page of the repository portlet (to access the edit page click the pencil symbol in top-right corner of the repository portlet). The repository configuration form contains predefined repository addresses (please see Figure -5).



The screenshot shows a web browser window titled "EMI Experiment Browser". Inside, there is a form titled "Experiment repository location setup". The form has three main sections: "Actual repository :", "Predefined repositories", and "Other repository". The "Actual repository :" field contains the URL "https://svn.gforge.hlr.de/svn/virolab/trunk/experiments". The "Predefined repositories" section has a dropdown menu with the same URL selected. The "Other repository" field is empty. Below these fields is a section titled "User credentials (optional)" which contains two input fields for "User name" and "User password". Below these fields is a note: "Provide user and password only if the repository is not anonymously readable feedback modification is required". At the bottom of the form is a button labeled "Change repository".

**Figure -5: The experiment repository configuration window.**

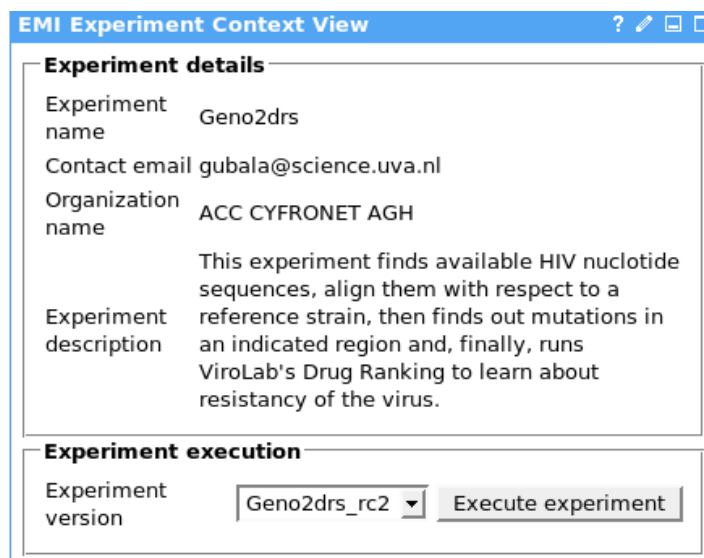
After providing the address and optionally the user credentials and submitting them to the server by clicking the Change repository button the triangle symbol in the portlet's bar should be clicked to return to the normal view mode. Immediately after going to this mode experiment list of the newly configured repository should be visible. Exemplary list of experiments is presented in Figure -6.



List of available experiments	
Experiment name	Available versions
Geno2drs	rc1
	rc2
Next experiment	1
	1-rereleased
Echo Experiment	1.0
	1.0
Pure Ruby Experiment	1.0
	1.0

**Figure -6: Example of a list of experiments stored inside a repository,**

On the right side of each experiment release versions are listed. For each version feedback and license links are provided. Selecting on of them updates the content of the context portlet. Examples of experiment details, feedback and license in the context portlet are presented in the following three images respectively: Figure -7, Figure -8 and Figure -9.



EMI Experiment Context View	
<b>Experiment details</b>	
Experiment name	Geno2drs
Contact email	gubala@science.uva.nl
Organization name	ACC CYFRONET AGH
Experiment description	This experiment finds available HIV nucleotide sequences, align them with respect to a reference strain, then finds out mutations in an indicated region and, finally, runs ViroLab's Drug Ranking to learn about resistancy of the virus.
<b>Experiment execution</b>	
Experiment version	Geno2drs_rc2
	<input type="button" value="Execute experiment"/>

**Figure -7: Experiment details in the Experiment Context Portlet.**

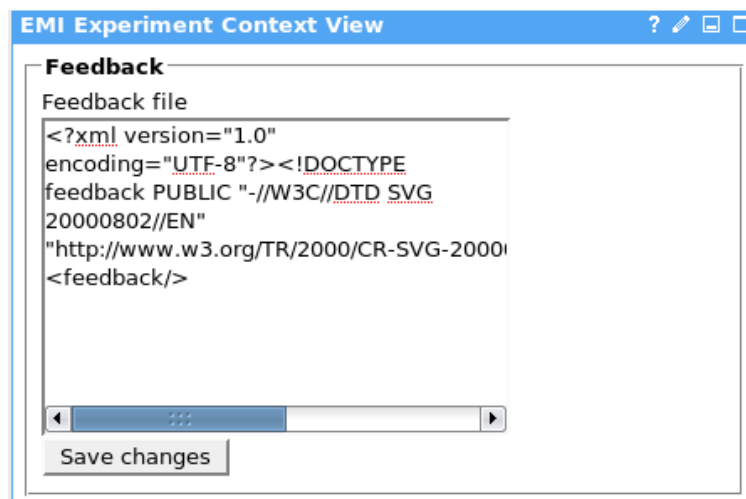


Figure -8: Feedback window in the Experiment Context Portlet.

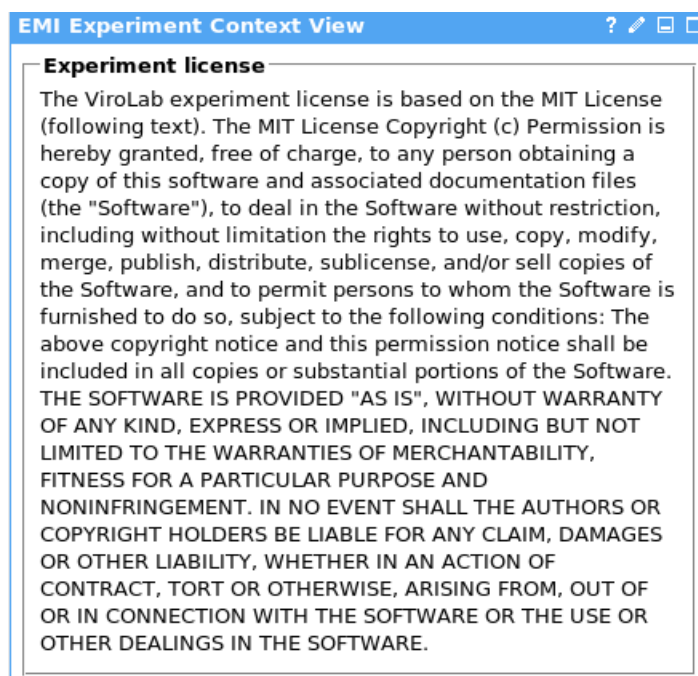


Figure -9: Experiment license text in the Experiment Context Portlet.

### 3.1.4.Experiment execution

#### Running an experiment.

To execute a particular version of an experiment, experiment details need to be displayed in the context portlet by clicking a version link of an experiment in the repository portlet. An execution field will then be displayed in the context portlet beneath the experiment details, similar to the one in Figure -10.

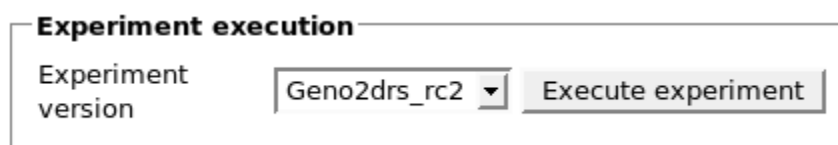


Figure -10: The experiment execution field.

After clicking the Execute experiment button a message about a successful experiment execution or an error message is displayed. According to the experiment plan the results will be presented or user input required through the third portlet. Details are described in the following subsections.

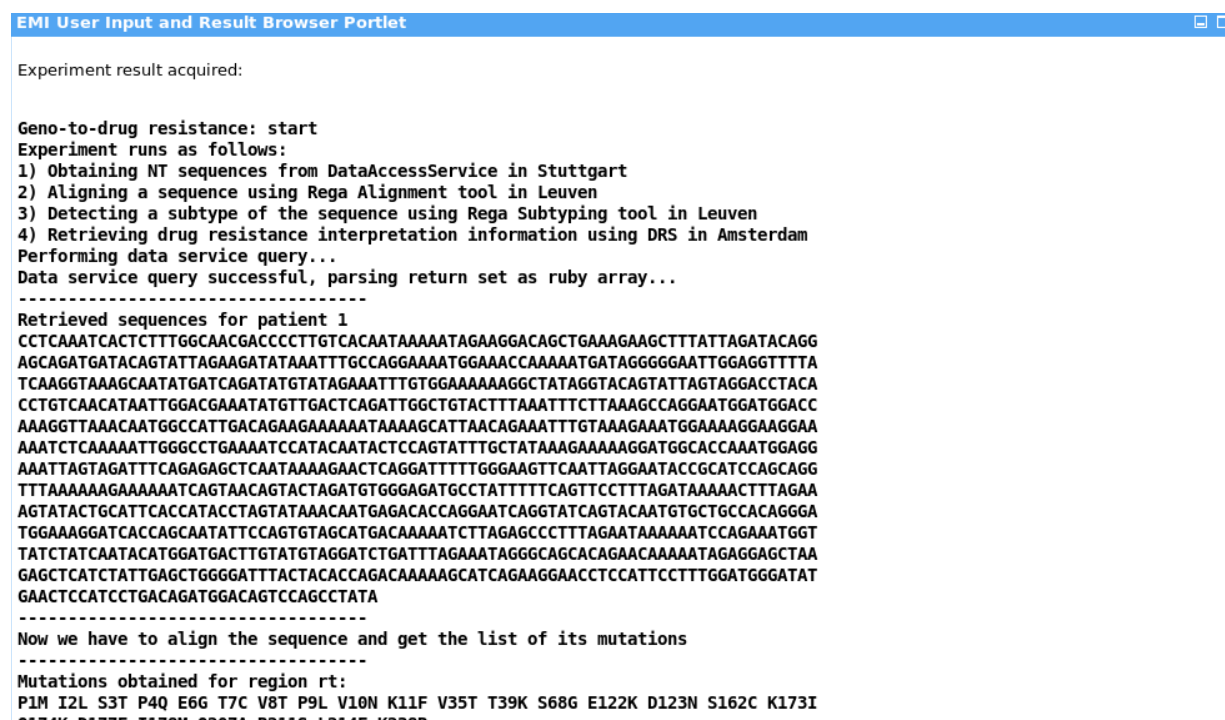
### Providing user input.

When a user input is required during an experiment execution the Result and user input portlet is updated and the user is asked to input required data.

After filling in the form Send data button should be clicked and the data are transferred back to the execution engine and the process continues. During one execution a couple of user data requests may be performed.

### Analyzing experiment output.

When the experiment execution finishes the output is presented to the user using the Result and user input portlet which is updated automatically if the user logs in or is logged in the portal. An exemplary experiment output is presented in .



```

EMI User Input and Result Browser Portlet
Experiment result acquired:

Geno-to-drug resistance: start
Experiment runs as follows:
1) Obtaining NT sequences from DataAccessService in Stuttgart
2) Aligning a sequence using Rega Alignment tool in Leuven
3) Detecting a subtype of the sequence using Rega Subtyping tool in Leuven
4) Retrieving drug resistance interpretation information using DRS in Amsterdam
Performing data service query...
Data service query successful, parsing return set as ruby array...
-----
Retrieved sequences for patient 1
CCTCAAAATCACTCTTTGGCAACGACCCCTGTGCACAATAAAAAAGGACAGCTGAAAGAAGCTTTATTAGATACAGG
AGCAGATGATACAGTATTAGAAAGATATAAATTTGCCAGGAAAAATGGAAACCAAAATGATAGGGGAATTGGAGGTTTAA
TCAAGGTAAGCAATATGATCAGATATGTATAGAAATTTGTGGAAAAAGGCTATAGGTACAGTATTAGTAGGACCTACA
CTGTCAACATAATTGGACGAAATATGTTGACTCAGATTGGCTGTACTTTAAATTTCTTAAAGCCAGGAATGGATGGACC
AAAGGTTAAACAATGGCCATTGACAGAGAAAAATAAAAGCATTAACAGAAATTTGTAAAGAAATGGAAAGGAAGGAA
AAATCTCAAAAAATGGGCCTGAAATCCATACAATACTCCAGTATTGCTATAAAGAAAAAGGATGGCACCAATGGAGG
AAATTAGTAGATTTTCCAGAGAGCTCAATAAAGAACTCAGGATTTTGGGAAGTTCAATTAGGAATACCGCATCCAGCAGG
TTTAAAAAAGAAAAATCAGTAACAGTACTAGATGTGGGAGATGCCTATTTTTCAGTTCCTTTAGATAAAAACTTTAGAA
AGTATACTGCATTACCATACCTAGTATAAACAATGAGACACCAGGAATCAGGTATCAGTACAATGTGCTGCCACAGGGA
TGGAAAGATCACCAGCAATATTTCCAGTGTAGCATGACAAAAATCTTAGAGCCCTTTAGAATAAAAAATCCAGAAATGGT
TATCTATCAATACATGGATGACTTGTATGTAGGATCTGATTTAGAAATAGGGCAGCAGACAGAAAAATAGAGGAGCTAA
GAGCTCATCTATTGAGCTGGGGATTTACTACCCAGACAAAAAGCATCAGAAGGAACCTCATTCTTTGGATGGGATAT
GAACTCCATCCTGACAGATGGACAGTCCAGCCTATA
-----
Now we have to align the sequence and get the list of its mutations
-----
Mutations obtained for region rt:
P1M I2L S3T P4Q E6G T7C V8T P9L V10N K11F V35T T39K S68G E122K D123N S162C K173I
Q174K D177E T179M Q207A R211S L214E K228R

```

Figure -11: An example of an experiment execution result.

After clicking the Ok button below the experiment output the content of the portlet is lost.

### Feedback mechanism.

In order to provide feedback to the experiment developer the feedback editor in the context portlet should be used. This requires however the user to have write permissions to the experiment repository. After filling in the remarks about the experiment the Save changes button should be clicked. If the feedback successfully is stored in the repository a proper message is displayed.

## 3.2. SOURCE CODE ACCESS, BUG REPORTING AND AUTHORS CONTACT INFORMATION



The entire source code of the EMI module is accessible through the Subversion repository (the anonymous read-only access is granted for everyone):

```
#> svn checkout https://virolab.cyfronet.pl/svn/emi
```

Should you find any bugs, missing functionality or you'd like to have some nice new features implemented, please use the ticket emission and management system on the EMI Trac website:

- Viewing tickets: <http://virolab.cyfronet.pl/trac/emi/report>
- Issuing new tickets: <http://virolab.cyfronet.pl/trac/emi/newticket>

You do not need any account for that, tickets could be submitted anonymously.

Author and contact information: Daniel Hareżlak [[d.harezlak@cyfronet.pl](mailto:d.harezlak@cyfronet.pl)].

## 4. QUERY TRANSLATION TOOLS USER'S MANUAL

Following guides explain typical User how to explore provenance query features through QUaTRO's graphical user interface. First section describes typical use of the QUaTRO - building a query. Second one lists all additional features of the GUI with short explanations.

### 4.1. ACCESSING AND RUNNING QUERY TRANSLATION TOOLS

#### 4.1.1. QUaTRO step-by-step

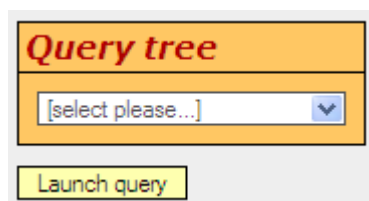
This manual page documents process of building an example, simple query using sophisticated QUaTRO GUI. Also some examples of more advanced queries are presented. For description of additional GUI features Reader should refer to the Quatro Features manual page.

##### 4.1.1.1. Introduction

QUaTRO allows for building provenance, knowledge-based queries from following building blocks:

- concepts from [ViroLab](#) ontologies, describing *experiment*, *applications* and *data*. All of them are loaded automatically from current descriptions published by the vl team. For example **NewDrugRanking** is a concept from the DRS *application* ontology
- properties of these concepts. There are two types of properties:
  - *object*, that model relations between concepts. For example property **usedRuleSet** model relation between concepts **NewDrugRanking** from DRS application ontology and **RuleSet** from data ontology
  - *datatype*, that model relation between a concept and a piece of data. This type of property could be treated as simple *attribute* of a concept. For example, concept **NewDrugRanking** has property **patientName** of type *string*.
- pieces of data, loaded automatically by using [ViroLab](#) Data Access Client (DAC). This allows user to choose existing parameters for new queries, that can be fulfilled.

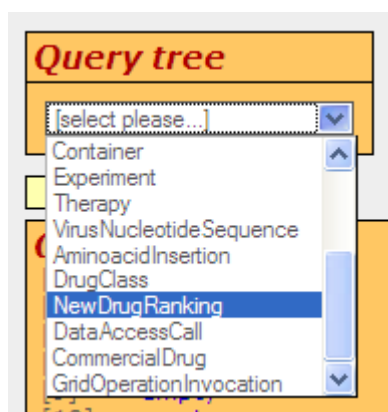
One more thing concerning concepts and data should be noted. Each data entity in [ViroLab](#) is modeled in the data ontology and thus loaded to QUaTRO as *concept*. In the relation database (RDB) notion it would be called *table*. As *piece of data* we mean data existing in [ViroLab](#) and accessible by the Data Access Client. In the RDB notion it would be called *column value*.



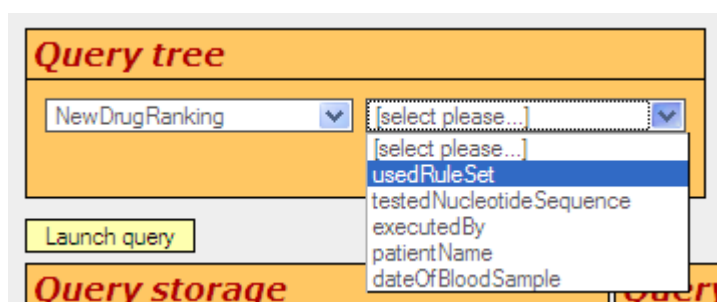
Above image depicts main panel of the query builder. It consists of a combo box, that is used to select concepts and *launch query* button. This is a part of the GUI, where our story begins...

#### 4.1.1.2.Simple Query Example

This sections describes process of creating simple query, that allows for selecting drug rankings (DRS application), that used rule sets from particular organization (**HIVDB**) and were performed on specified patients (with second name **Smith**).



Query starts with selecting chosen concept using combo box. Clicking on combo box will show list, containing available concepts from all VL ontologies. In our example we choose **NewDrugRanking** concept from the DRS application ontology.



After selecting a concept new combo box emerges. It contains list of all properties that exists for this particular concept. At this point *datatype* and *object* properties are mixed at one list. In our example we extend query by choosing **usedRuleSet** property.

The screenshot shows the 'Query tree' interface. It has a header bar with the title 'Query tree'. Below it, there are four dropdown menus: 'NewDrugRanking', 'usedRuleSet', 'RuleSet', and a fourth one with the text '[select please...]'. Below these dropdowns are two 'and' buttons. A 'Launch query' button is located below the dropdowns. The 'RuleSet' dropdown is open, showing a list of properties: 'id', 'lastupdate', 'resourceid', 'name', 'version', and 'location'. The 'name' property is highlighted in blue.

Because the property chosen was *object*, next combo box that emerges contains list of concepts that are *domain* for relation modeled by our property. In our example only **RuleSet** concept is suitable and thus was chosen. Picture depicts further extension to the query - selection of property of current concept (**RuleSet**). We do this same way as previously.

The screenshot shows the 'Query tree' interface. It has a header bar with the title 'Query tree'. Below it, there are four dropdown menus: 'NewDrugRanking', 'usedRuleSet', 'RuleSet', and a fourth one with the text '[select please...]'. Below these dropdowns are two 'and' buttons. A 'Launch query' button is located below the dropdowns. The 'RuleSet' dropdown is open, showing a list of operators: '=', '<', '>', '<=', '>=', and 'contains'. The '=' operator is highlighted in blue.

We have chosen *name* property of the **RuleSet** concept, which occurred to be a datatype property. We distinguish datatype property from object because instead of combo box with *domain* concepts, box with operators and input were rendered.

Now we choose operator for the *name* property. Because we want some exact organization to be taken into account, the equality operator must be chosen.

The screenshot shows the 'Query tree' interface. It has a header bar with the title 'Query tree'. Below it, there are four dropdown menus: 'NewDrugRanking', 'usedRuleSet', 'RuleSet', and a fourth one with the text '[select please...]'. Below these dropdowns are two 'and' buttons. A 'Launch query' button is located below the dropdowns. The 'RuleSet' dropdown is open, showing a list of values: 'ANRS', 'HIVDB', and 'HIVDB'. The 'HIVDB' value is highlighted in blue.

One last thing in current *branch* of the query is value for **name** property of the **RuleSet** concept, selected in previous steps. Thanks to unique data-ontology mapping employed in ViroLab, QUaTRO allows for two ways of value specification:

- by hand, filling the input box
- automatically, using existing values loaded via DAC

In current example we use the second way. By clicking on a button located right to the input we open list containing all distinct values for current attribute. Choice of **HIVDB** value from the list ends this branch of the query.

The screenshot shows the 'Query tree' interface with the following configuration:

- Left Column:** NewDrugRanking (selected), usedRuleSet (selected), RuleSet (selected).
- Operator:** and (checked).
- Right Column:** name (selected), != (selected), HIVDB (selected).
- Input Fields:** patientName (selected), contains (selected), Smith (entered).
- Buttons:** Launch query (yellow button).

Selected property appeared to be a datatype property, so *operator* and *value* boxes emerged. Because we want to select patient with second name **Smith** we choose operator **contains**. This particular operator will search for values with substring equal to the requested value, entered in the input box. Click on the **launch query** button will send our query to the processing.

#### 4.1.1.3. Advanced queries

The screenshot shows a complex query configuration in the 'Query tree' interface:

- Left Column:** Experiment (selected), ownerLogin (selected), != (selected), JamesBond (entered).
- Operator:** and (checked).
- Right Column:** inputData (selected), AminoacidInsertion (selected), rt\_insertion\_codon (selected), = (selected), insertion\_codon (selected).
- Input Fields:** AminoacidMutation (selected), hasStage (selected), DataAccessCall (selected), sourceFile (selected), != (selected), test (entered).
- Buttons:** Launch query (yellow button).

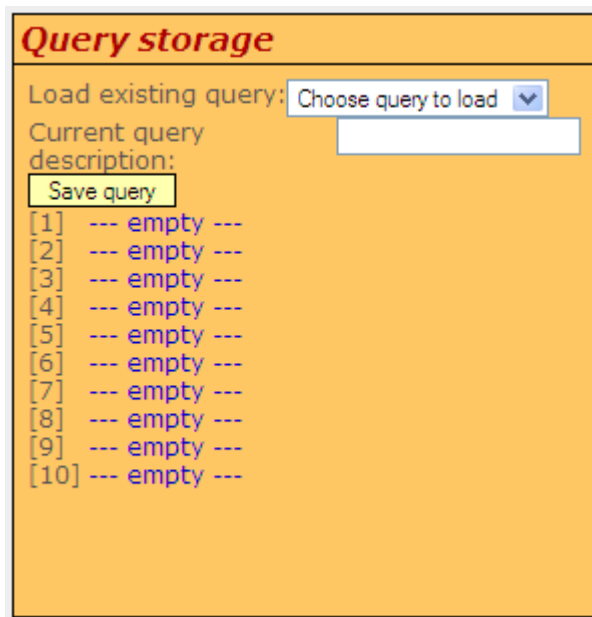
Above image depicts quite advanced query that could be built using QUaTRO GUI. It contains multiple branches joined at different levels, five concepts, many datatype and object properties. Nevertheless, process of building this query is very similar to the simple one, described in details in the previous section. Using the same simple mechanisms user is able to build even more advanced queries, containing tens of concepts and properties.

#### 4.1.2. QUery TRanslation tOols (QUaTRO) GUI Features Overview

This manual page documents some additional features present in the QUaTRO GUI. Although most of these features aren't necessary for typical User, they can be found handy. Thus we encourage Users to read this short description. For information about basic use - query building - Reader should refer to page.

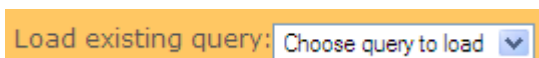
##### 4.1.2.1. Query Storage

Features grouped in this section allows for query storage and load. Storage can be temporary (quick saves) or persistent (persistent saves).



Above image depicts entire storage panel, as can be found in the QUaTRO GUI. Parts of the storage panel, relevant to specific functionalities are described in separate sections below.

#### **4.1.2.2.Query load**



Above picture shows combo box used for loading queries. The query storage is separate for every user that logs into ViroLab portal and uses QUaTRO GUI. Credentials for storage are taken automatically from portlet context, so each user will see only queries written by himself. This combo box allows only for loading persistent queries. User should choose previously saved query using combo box. After selecting a query, QUaTRO will automatically load it into the query tree. Query will be immediately ready to be launched.

#### **4.1.2.3.Query persistent storage**



Depicted panel part allows to persist current query. It will be stored in the internal data base and available for later use. Only user that saved the query will be able to load it in the future. To save current query (that is query present in the editor) user should enter its unique description (*Current query description*) and click button *save query*. QUaTRO will display message depending on the success or failure of saving.

#### **4.1.2.4.Query quick save / load**

```
[1] --- empty ---
[2] --- empty ---
[3] --- empty ---
[4] --- empty ---
[5] --- empty ---
[6] --- empty ---
[7] --- empty ---
[8] --- empty ---
[9] --- empty ---
[10] --- empty ---
```

Part of the panel depicted above show so called *quick slots* for queries. Every slot can hold one query, that will be kept in the memory, not persistent storage. This is much faster than using persistent saves but will last only as long as portal session of the current user.

This type of the short-time storage allows user to build for example few versions of the same query for testing purposes. Actually it acts as a clipboard for QUaTRO users.

To use this clipboard, users should first input its description in the input box *Current query description* and click on chosen slot. If slot is already in use, old query will be replaced (and thus lost).

#### **4.2. AUTHORS CONTACT INFORMATION**

Authors list: Jakub Wach, Bartosz Baliś.

Developer team contact: Jakub Wach [[wach.kuba@gmail.com](mailto:wach.kuba@gmail.com)].

## 5. GRIDSPACE EXPERIMENT USER MANUAL

### 5.1. INSTALLATION AND USE

For versions:

- *GSEngine-0.3.0*

#### 5.1.1. Installation

##### Prerequisites

- Java Runtime JRE 1.5, java executable needs to be in the PATH environment variable
- JRuby v1.0 or higher binary distribution with its installation directory set as JRUBY\_HOME
  - <http://dist.codehaus.org/jruby/jruby-bin-1.0.zip>
  - <http://dist.codehaus.org/jruby/jruby-bin-1.0.tar.gz>

##### What is inside

- Latest versions of GridSpace Engine (GSEngine) components
- Integrated with: Generic Data Access Client (DAC), GridSpace Application Optimizer (GrAppO) and Grid Operations Invoker (GOI)

##### Installation steps for Linux

- Download the archive containing GSEngine release: [gsengine-<version>.tar.gz](#) from GSEngine download page ([http://gforge.cyfronet.pl/frs/?group\\_id=45](http://gforge.cyfronet.pl/frs/?group_id=45))

- Extract the content of the archive:

```
tar zxvf gsengine-<gsengine_version>.tar.gz
```

or if you downloaded the .zip version of the archive:

```
unzip gsengine-<gsengine_version>.zip
```

- Set environment variable GS\_HOME:

```
export GS_HOME=`pwd`/gsengine-<gsengine_version>/
```

- To use gsengine from different directories you may also adjust the PATH environment variable:

```
export PATH=$GS_HOME/bin:$PATH
```

##### Installation steps for Windows XP

- Download the archive containing GSEngine release: [gsengine-<version>.zip](#) from GSEngine download page ([http://gforge.cyfronet.pl/frs/?group\\_id=45](http://gforge.cyfronet.pl/frs/?group_id=45))



- Extract the content of the archive:
  - click right mouse button on the archive, select *"Extract files..."* and follow the wizard steps
- Set environment variable GS\_HOME:

```
set GS_HOME=<gsengine_home>
```

- where <gsengine\_home> is the path to the directory where the archive was extracted (e.g. C:\gsengine-<gsengine\_version>)
- To use gsengine from different directories you may also adjust the PATH environment variable:

```
set PATH=%GS_HOME%\bin;%PATH%
```

In order to check whether GSEngine is properly installed and what is the version, try to launch it from a command line by typing:

```
gsengine --version
```

### 5.1.2. Working with gsengine command line tool

As long as you have GSEngine's bin directory included in PATH environment variable you are enabled to launch command line tool by simply typing:

```
gsengine <parameters>
```

The parameters are discussed further. However, it is worth to mention here that every time you need a hint on how to use GSEngine command-line tool or you just want to check whether the tool is ready just type:

```
gsengine --help
```

or simpler:

```
gsengine -h
```

and if it is properly installed you can expect to see a help on the syntax of the command:

```
To use GSEngine interpreter command-line tool, type 'gsengine'
followed by (use one of the options):
<script_file> [params]* - evaluates the script from the specified
file, the <script_file> can be file path relative to the current
directory; [params] are arguments passed to the script
-r <evaluation_request_file> [params]* - evaluates the script
contained in the evaluation request read from
<evaluation_request_file>; [params] is a list of parameters passed to
evaluation request and to the script itself, i.e. the successive
values correspond to the parameters of the evaluation request
specified as $1, $2 etc., the ones that are not used by the
evaluation request will be passed to the evaluated script.
For information on creating evaluation request files, please visit
http://virolab.cyfronet.pl/trac/vlruntime/wiki/GSEngineUserManual#Ev
aluationrequesttypes
--help, -h - displays this help
```

```
--version, -v - displays GSEngine version
```

As the hint says, in general, the command accepts evaluation request file option and other optional parameters. Evaluation request file is a plain XML file. GSEngine accepts several evaluation request types described by different [XML- Schema files](http://virolab.cyfronet.pl/~asia/gsengine/files/xsd/) (<http://virolab.cyfronet.pl/~asia/gsengine/files/xsd/>). Every type of evaluation request may be parameterized with variables named \$1, \$2, \$3 etc., that will be assigned values provided by parameters no. 1, 2, 3 and so on. The parameters which are not used by the evaluation request will be passed to the evaluated script as its arguments.

When no evaluation request is provided with the command, a default one is used - it is stored in GSEngine installation directory, namely `<GS_HOME>/default.evaluation.request.xml`. The default evaluation request provided with the distribution is of type local file script evaluation request. Naturally, the user is allowed to customize this default evaluation request by modifying the aforementioned file. In order to use the default evaluation request just type:

```
gsengine <space-separated parameters>
```

The subsequent sections are dedicated to the specific evaluation request types.

## Evaluation request types

A core part that all evaluation request types have in common is an XML sub element `evaluationRequest`, which is contained in every specific evaluation request.

We provide XML-Schema files that specify the evaluation request files syntax for all the evaluation request types. They can be downloaded from <http://virolab.cyfronet.pl/~asia/gsengine/files/xsd/>. Note: The file `er-schema.xsd` is required for all the evaluation request types.

```
<evaluationRequest>
  <dasUrl>...</dasUrl>
  <dosUrl>...</dosUrl>
  <grappoUrl>...</grappoUrl>
  <grrBaseUrls>
    <grrBaseUrl>...</grrBaseUrl>
    <grrBaseUrl>...</grrBaseUrl>
  </grrBaseUrls>
  <optimizationPolicy>...</optimizationPolicy>
  <protosUrl>...</protosUrl>
  <userHandle>...</userHandle>
</evaluationRequest>
```

The semantics of the elements nested in the `evaluationRequest` element is explained below:

- `dasURL` element - URL pointing to Data Access Service.
- `dosURL` element - URL pointing to Domain Ontology Store service.

- `grappoURL` element - URL pointing to GridSpace Application Optimizer service - the application execution optimization service.
- `grrBaseURLs` element - list of possible locations of Grid Resources Registry.
- `grrBaseURL` element - URL of Grid Resources Registry.
- `optimizationPolicy` element - URL pointing to an XML file that defines a policy to be used during experiment execution optimization.
- `protosURL` element - URL pointing to Provenance Tracking System (PROToS) service.
- `userHandle` element - includes credentials used by security system.

The specific evaluation request types specify a parent element which contains specific XML elements aside from the `evaluationRequest` element.

- *local file script evaluation request* - a request for evaluation of an experiment stored in the local filesystem. Provides information of the scripts localization.
  - `localFileScriptER` root element.
  - `experimentDirectory` element - the directory in which the experiments are stored. The scripts dependencies are resolved using paths relative to this directory.
  - `experimentMainFile` element - the name of the main experiment script file. It will be executed by GSEngine.
  - `rubyLibPath` element - contains 'path' sub elements that specify the directories where the external ruby dependencies of the application are stored.

```
<localFileScriptER>
  <evaluationRequest>
    ...
  </evaluationRequest>
  <experimentDirectory>...</experimentDirectory>
  <experimentMainFile>...</experimentMainFile>
  <rubyLibPath>
    <path>...</path>
    <path>...</path>
    <path>...</path>
  </rubyLibPath>
</localFileScriptER>
```

- *explicit script evaluation request* - request for evaluation of explicitly specified experiment. Provides a way of direct specification of application's scripts.
  - `explicitScriptER` root element.

- `scripts` element - enclose all scripts and specifies the name of the main experiment file to be evaluated in `mainScriptName` attribute.
- `script` element - specifies an individual script content in its body, and the script name in the `name` attribute.

```
<explicitScriptER>
  <evaluationRequest>
    ...
  </evaluationRequest>
  <scripts mainScriptName="...">
    <script name="..."><![CDATA[...]]></script>
    <script name="..."><![CDATA[...]]></script>
    <script name="..."><![CDATA[...]]></script>
  </scripts>
</explicitScriptER>
```

- *repository staged script evaluation request* - a request for evaluation of an application stored in an application repository. Provides information of the scripts localization, credentials necessary to obtain it.
  - `repositoryStagedScriptER` root element.
  - `repoUrl` element - URL to the experiment repository (it must point to the directory where experiments are stored).
  - `repoLogin` element - user login, if required for the experiment repository access.
  - `repoPassword` element - user password (corresponding to `repoLogin`), when required for the experiment repository access.
  - `experimentName` element - the name of the experiment to be executed.
  - `experimentVersion` element - version of the experiment to be executed.
  - `experimentMainFile` element - the name of the main experiment script file. It will be executed by GSEngine.

```
<repositoryStagedScriptER>
  <evaluationRequest>
    ...
  </evaluationRequest>
  <repoUrl>...</repoUrl>
  <repoLogin>...</repoLogin>
  <repoPassword>...</repoPassword>
  <experimentName>...</experimentName>
  <experimentVersion>...</experimentVersion>
  <experimentMainFile>...</experimentMainFile>
```

```
</repositoryStagedScriptER>
```

In each evaluation request type element's text value don't have to be explicitly specified. Instead, a variable may be applied that will be assigned the value given as a parameter with the command. For example:

```
<repositoryStagedScriptER>
  <evaluationRequest>
    ...
  </evaluationRequest>
  <repoUrl>...</repoUrl>
  <repoLogin>$1</repoLogin>
  <repoPassword>$2</repoPassword>
  <experimentName>...</experimentName>
  <experimentVersion>...</experimentVersion>
  <experimentMainFile>...</experimentMainFile>
</repositoryStagedScriptER>
```

will expect two parameters given as the parameters in a command, for example:

```
gsengine -r myRepositoryStagedScriptER.xml me mySecret
```

Not having found the parameters, the tool will print out an adequate error message, e.g.:

```
cyfronet.gridspace.engine.EvaluationRequestParameterNotFound: No
parameter at index:1
```

## Examples

### *Example: explicit script evaluation request*

Evaluation request of a simple Ruby scripts with nested dependencies that are specified explicitly in evaluation request. Application takes 3 parameters that will be printed out subsequently.

The evaluation request in below may be found at `<GS_HOME>/samples/er-explicit.xml`.

```
<explicitScriptER>
  <evaluationRequest>
    ...
  </evaluationRequest>
  <scripts mainScriptName="main.rb">
    <script name="main.rb"><![CDATA[require 'additional_file'
require 'one_more_file'
@@bclass = BClass.new
@@bclass.printSth()
puts ARGV[2]
]]></script>
    <script name="additional_file.rb"><![CDATA[
class AClass
def printSth()
```

```

    puts ARGV[0]
  end
end
]]></script>
    <script name="one_more_file.rb"><![CDATA[
class BClass
require 'additional_file'
  def printSth()
    AClass.new.printSth()
    puts ARGV[1]
  end
end
end
]]></script>
    </scripts>
</explicitScriptER>

```

Launching it is resulting in the following output:

```

$ gsengine -r <GS_HOME>/samples/er-explicit.xml a b c
a
b
c

```

#### *Example: local file evaluation request*

Evaluation request of any of the scripts contained in the local filesystem. It takes two parameters evaluation request parameters:

- scripts\_dir script\_name
- the path to the directory containing scripts to be executed and name of the script file (either relative or absolute path)

The evaluation request in below may be found at <GS\_HOME>/samples/er-local.xml.

```

<localFileScriptER>
  <evaluationRequest>
    ...
  </evaluationRequest>
  <experimentDirectory>$1</experimentDirectory>
  <experimentMainFile>$2</experimentMainFile>
</localFileScriptER>

```

#### *Example: repository staged script evaluation request*

Evaluation request of a simple echo\_experiment staged in the SVN repository (at gforge.cyfronet.pl). It takes two evaluation request parameters:

- user\_login
- user\_password (for the repository authentication)

The evaluation request in below may be found at `<GS_HOME>/samples/er-echo-svn.xml`.

```
<repositoryStagedScriptER>
  <evaluationRequest>
    <...
  </evaluationRequest>
  <repoUrl>https://gforge.cyfronet.pl/svn/gsengine/trunk/gsengine-
  exprepo-svn/etc/test-files</repoUrl>
  <repoLogin>$1</repoLogin>
  <repoPassword>$2</repoPassword>
  <experimentName>echo_experiment</experimentName>
  <experimentVersion>1.0</experimentVersion>
  <experimentMainFile>echo_experiment.rb</experimentMainFile>
</repositoryStagedScriptER>
```

### 5.1.3. Working with gsquery command line tool

gsquery is an executable that overlays Data Access Client (DAC). In its current form it enables querying data sources. Invoking it with `--help` or `-h` option will print out the usage instructions.

```
To use GSQuery command-line tool, type 'gsquery' followed by (use
one of the options):

    <dbplatform>:<address>:<dbname>:<dbuser>:<dbpassword> <query>
-
executes the given <query> on the database specified by the
    <dbplatform>:<address>:<dbname>:<dbuser>:<dbpassword> sequence
--help, -h - displays this help
--version, -v - displays GSEngine version
```

Notice that a query is actually a query in terms of Java Database Connectivity (JDBC) and therefore is not allowed to perform data update.

For further details please refer to the Data Access Client (DAC) documentation.

### Examples

*Example: simple query*

Suppose we need to access mysql database called `my_database`, where we have an account with login `my_login` authenticated by password `my_password` and make a sample query for flowers stored in a database.

```
gsquery
mysql:my_database_host.org:my_database_name:my_login:my_password
"select * from flowers"
```

### 5.1.4. Working with GSEngine API

In order to use GSEngine via its Application Programming Interface (API) you need to have GSEngine previously according to instructions from the first part of this tutorial. Moreover, you must to include in your application classpath all the jar files contained in the `<GS_HOME>/java` directory and its subdirectories.

An entry point to the GSEngine API is the `cyfronet.gridspace.engine.InterpreterFacade` interface, that provides two 'evaluateScript' overloaded methods as seen below.

```
public interface InterpreterFacade {
    public EvaluationResponse evaluateScript(
EvaluationRequest evaluationRequest, EvaluationCallback callback)
        throws InterpreterException,
InvalidEvaluationRequestException,
        ScriptNotFoundException,
        EvaluationRequestParameterNotFoundException,
        UnsupportedEvaluationRequestType;

    public EvaluationResponse evaluateScript(
        EvaluationRequest evaluationRequest,
EvaluationRequest.Parameters parameters, EvaluationCallback
callback)
        throws InterpreterException,
InvalidEvaluationRequestException,
        ScriptNotFoundException,
        EvaluationRequestParameterNotFoundException,
        UnsupportedEvaluationRequestType;
}
```

The main classes that developer deals with while using `InterpreterFacade` are:

- `cyfronet.gridspace.engine.evalreq.EvaluationRequest` - an abstract class that represents the request for an application evaluation. It contains all the data needed for creating evaluation context and obtaining the application script. The subtypes defines a way how application code is provided from arbitrary location (e.g. local files, files stored in experiment repository). Each subclass is serializable to the XML form discussed in a section devoted to gsengine command line tool. Subclasses expose simple setter methods for setting evaluation requests attributes. Available subclasses (discussed in details in Section 5.1.2):
  - `ExplicitScriptEvaluationRequest`
  - `LocalFileScriptEvaluationRequest`
  - `RepositoryStagedScriptEvaluationRequest`
- `cyfronet.gridspace.engine.evalreq.EvaluationRequest.Parameters` - a class that simply stores evaluation request parameters organized in a list. The evaluation request is filled with the parameters values as explained previously.
- `cyfronet.gridspace.engine.EvaluationCallback` - an interface that establishes callback data transfer channel while evaluating a script. Callback provides a way of streaming data during the call (input and



output streams), supporting interaction (data inputs) and notification and information related to application evaluation status sent towards the application executors. All of the member methods are called by GSEngine facade implementations. This interface code is highly self-explanatory and is shown below.

```
public interface EvaluationCallback {
    /**
     * Invoked when evaluation is complete.
     *
     *
     * @param result
     *         serialized ruby object that is a result of the
application
     */
    public void onEvaluationComplete(String result);
    /**
     * This method is called just before application execution and
sets
     * GSEngine
     * id id (GSEID)
     *
     * @param gseid
     */
    public void setGseid(String gseid);
    /**
     * This method is called in order to make GSEngine provided with
     * output stream where the output of executed application has to
     * be directed.
     *
     * @return
     */
    public OutputStream getOutputStream();
    /**
     * This method is called in order to make GSEngine provided
with
     * error stream where the error output of executed application has
to
     * be directed.
     *
     * @return
     */
    public OutputStream getErrorStream();
    /**
     * This method is called in order to make GSEngine provided
with
     * input stream from which where the input of executed
application

```

```

        * has to taken from.
        *
        * @return
        */
public InputStream getInputStream();
/**
 * This method is called each time the application request for
data
 * to be provided by the application executor. The application
is
 * blocking until this call returns.
 *
 * @param dataRequest
 * @return
 */
public String getData(String dataRequest);
/**
 * Invoked each time interpreter raises exception.
 *
 * @param interpreterException
 */
public void raise(InterpreterException interpreterException);
}

```

- `cyfronet.gridspace.engine.EvaluationResponse` - a class that represents a result of evaluation after it is complete. It stores serialized form of Ruby object returned by a script if any.

`GSEngineFacade` interface is covering the underlying implementations. It is in the `GSEngine` roadmap to provide multiple realizations of this interface that would enable remote evaluation of script. However, while the `GSEngine` is now under rapid development the only one available implementation is `cyfronet.gridspace.engine.impl.interpreter.EmbeddedInterpreter` that enables scripts evaluation embedded within the same JVM within which interpreter facade client runs.

For further explanation please refer to the javadoc contained in the distribution.

## Example

This example presents general way in which the `GSEngine` API is used in order to evaluate an application.

At first the concrete evaluation request has to be instantiated and configured by setting its attributes. If the evaluation request or application itself takes the parameters, the `Parameter` object has to be instantiated and filled with parameters values. Further we need to provide our realization of `EvaluationCallback` interface that establishes the interaction and data flow between application executor and application. Finally, the `InterpreterFacade`

realization has to be created and its `evaluateScript()` may be called, that returns `EvaluationResponse` object

```
// first, the concrete evaluation request should be instantiated and
// configured
ExplicitScriptEvaluationRequest er =
new ExplicitScriptEvaluationRequest();

// this attributes in below are common for each kind of evaluation
// request
er.setDasUrl(...);
er.setDosUrl(...);
er.setGrappoUrl(...);
List<String> grrBaseUrls = new ArrayList<String>();
grrBaseUrls.add(...);
er.setGrrBaseUrls(...);
er.setOptimizationPolicy(...);
er.setProtosUrl(...);
er.setUserHandle(...);

// this attributes in below are specific to the
// ExplicitScriptEvaluationRequest
er.addScript("main", "puts ARGV[0]\nputs ARGV[1]\nputs ARGV[2]\n");
er.setMainScriptName("main");

// since 'main' script takes input arguments they should be provided
Parameters params = new EvaluationRequest.Parameters();
params.addParam("arg0");
params.addParam("arg1");
params.addParam("arg2");

// we have to provide our implementation of EvaluationCallback
EvaluationCallback callback =
new MyImplementationOfEvaluationCallback();

// instantiation of InterpreterFacade implementation
InterpreterFacade inter = new EmbeddedInterpreter();
EvaluationResponse ir = inter.evaluateScript(er, params, callback);
```

For further explanation please refer to the javadoc contained in the distribution.

## **5.2. SOURCE CODE ACCESS, BUG REPORTING AND AUTHORS CONTACT INFORMATION**

The entire source code of the GridSpace is accessible through the Subversion repository (the anonymous read-only access is granted for everyone):

```
#> svn checkout https://gforge.cyfronet.pl/svn/gseengine
```

Should you find any bugs, missing functionality or you'd like to have some nice new features implemented, please use the ticket emission and management system on the Trac website of the GridSpace Engine:

- Viewing tickets: <http://virolab.cyfronet.pl/trac/vlruntime/report>
- Issuing new tickets: <http://virolab.cyfronet.pl/trac/vlruntime/newticket>

You do not need any account for that, tickets could be submitted anonymously.

Authors list: Joanna Kocot, Eryk Ciepiela, Piotr Nowakowski, Tomasz Bartyński, Maciej Malawski.

Developers team contact person: Eryk Ciepiela [[e.ciepiela@cyfronet.pl](mailto:e.ciepiela@cyfronet.pl)].

## ABBREVIATIONS

Fix the list **[Tomasz Gubala]**

Abbreviation/Term	Explanation
AAS	Aminoacid Sequence
API	Application Programmer's Interface
ARID	Application Run Identifier
CCA	Common Component Architecture
DAC	Data Access Client
DB	Database
DEISA	Distributed European Infrastructure for Supercomputing
DGE	Data Gathering Engine
DO	Domain Ontology
DRAM	Drug Resistance Associated Mutations
DRE	Data Retrieval Engine
DRS	Drug Ranking System
DS	Distributed Storage
DSS	Decision Support System
EGEE	Enabling Grids for e-Science in Europe
EPL	Experiment Planning Language
FLOWR	For-Let-Where-Order by-Return
GOB	Grid Object Class
GOBI	Grid Object Instance
GOBID	Grid Object Identifier
GOBImpl	Grid Object Implementation
GOp	Grid Operation
GOI	Grid Operation Invoker
GrAppO	Grid Application Optimizer
GRR	Grid Resources Registry
GT	Globus Toolkit
GUI	Graphical User Interface
HIV	Human Immunodeficiency Virus
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronic Engineers
JMX	Java Management Extensions
JSR	Java Specification Request
JVMTI	Java Virtual Machine Tool Interface

Abbreviation/Term	Explanation
LCG	LHC Computing Grid
LHC	Large Hadron Collider
LOB	Large Object
MQL	Meta Query Language
NS	Nucleotide Sequence
OGSA	Open Grid Services Architecture
OGSA-DAI	Open Grid Services Architecture - Data Access Integration
OGSA-DQP	Open Grid Services Architecture - Distributed Query Processing
OO	Object-Oriented
OR	Object-Relational
OWL	Web Ontology Language
PDP	Policy Decision Point
PROToS	Provenance Tracking System
RAD	Rapid Application Development
RBAC	Role-Based Access Content
RDF	Resource Description Framework
RDQL	Resource Description Framework Data Query Language
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SN	Storage Node
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSH	Secure Shell
SSL	Secure Socket Layer
SSN	Storage Super Node
SSO	Single Sign-On
SVN	Subversion
TLS	Transport Level Security
UML	Unified Modeling Language
URI	United Resource Identifier
UTF8	8-bit Unicode Transformation Format
VL	Virtual Laboratory
VM	Virtual Machine
VO	Virtual Organization
VPN	Virtual Private Network
WP	Work Package
WS	Web Service

Abbreviation/Term	Explanation
WS-I	Web Services Integration
WSDL	Web Services Definition Language
WSRF	Web Services Resource Framework
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language

## REFERENCES

Remove the list if not used at all [**Tomasz Gubala**]

- [AXIS]                      The Apache Axis project, a Java platform for creating and deploying Web Services applications,  
<http://ws.apache.org/axis/>