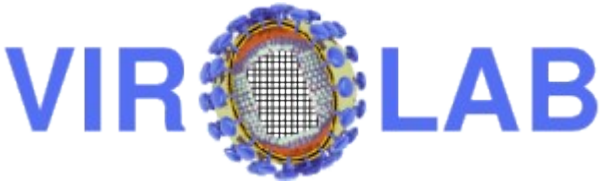

Deliverable 3.3 Appendix 3
ViroLab Virtual Laboratory:
Virtual Laboratory Developers' Manual

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

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

Log of Document

Version	Date	Changes Summary	Authors
0.1	29/08/2007	Initial version of the manual	Tomasz Gubala
0.2	03/09/2007	Main contribution	Robert Pajak, Dariusz Krol, Marek Kasztelnik, Piotr Regiel, Eryk Ciepiela, Tomasz Bartynski, Joanna Kocot, Piotr Nowakowski
0.3	04/09/2007	Formatting extending GRR paragraph, section 3.1 and 3.2	Marek Kasztelnik
0.4	05/09/2007	Formatting and minor changes in section 4.1	Michał Pelczar
0.5	06/09/2007	Checked and refined GrAppO and EMI parts.	Tomasz Gubala
0.6	06/09/2007	Checked, refined, supplemented and formatted sections: 2.1, 2.2.1, 2.3.	Eryk Ciepiela
0.7	06/09/2007	Reviewed and updated section 2.2.3	Piotr Nowakowski
0.8	06/09/2007	Checked, refined, supplemented and formatted section 2.2.2	Tomasz Bartyński
0.9	07/09/2007	Minor changes	Kuba Wach
1.0	13/09/2007	Manual title changed, minor changes	Marian Bubak, Marek Kasztelnik

TABLE OF CONTENTS

List of Figures

COPYRIGHT NOTICE

1. INTRODUCTION

This document contains a set of manuals and tutorials for a person that would like to administrate or extend existing ViroLab Virtual Laboratory. The sections inside contains instructions how to download, configure and extend virtual laboratory components.

1.1. TARGET AUDIENCE

The intended audience of this document includes any person that would like to administrate (administrator) or extend (developer) ViroLab Virtual Laboratory. Virtual laboratory components are created mostly in Java [JAVA] and Ruby [RUBY] languages, thus knowledge about these programming languages is crucial. ViroLab Virtual Laboratory components like Grid Resources Registry, Domain Ontology Store, PROTos and DAS are deployed into third party container. That is why administrator or developer has to have knowledge about containers such as Tomcat [TOMCAT], Jetty [JETTY] and Globus Toolkit 4.0 [GT4].

1.2. MORE INFORMATION

This document is not the only source of information for future administrators and 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. GRIDSPACE ENGINE DEVELOPER'S MANUAL

2.1. ARCHITECTURE OF THE GRIDSPACE ENGINE

GridSpace Engine (in short GSEngine) implementation involves several tightly-coupled components.

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

The aforementioned components use external libraries such as: JRuby 1.0 and SVNKit 1.1.2. Having in mind that GScript is actually a Ruby language with external libraries provided, JRuby [JRUBY] is used as a Ruby [RUBY] language implementation written in Java that allows integration and cooperation between Java code and Ruby code. SVNKit [SVNKIT] is used as a client of the SVN-based implementation of Application Repository. The dependencies between all the internal as well as external components of GSEngine are depicted on the diagram in Figure 2-1.

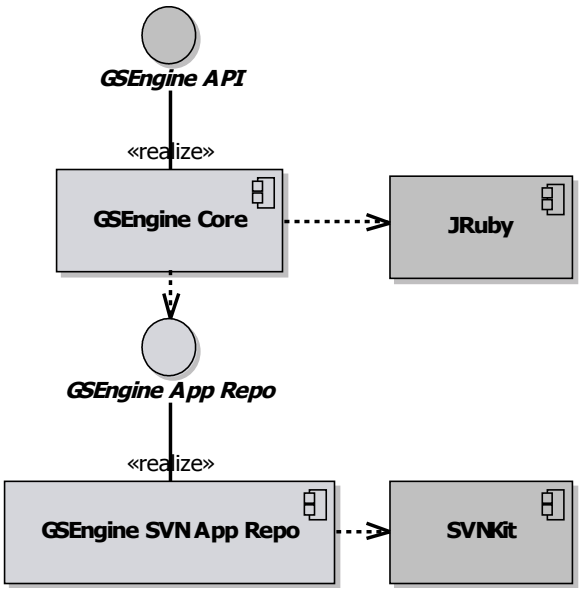

Figure 2-1: Main components and interfaces of GSEngine. GridSpace Engine implementation involves several tightly-coupled components; the internal components are GSEngine Core that implements GSEngine API and uses GSEngine SVN App Repo that, in turn, implements GSEngine App Repo API; the external libraries such as: JRuby 1.0 and SVNKit 1.1.2 are used.

GSEngine incorporates the Virolab-specific libraries of GOI and DAC by including them in the JRuby interpreter classpath, assuming that they are placed in the directory pointed by `GS_HOME` environment variable. Moreover, Virolab-specific libraries are provided with a GridSpace Engine Application Context that contains

all the parameters required by the libraries and that is passed by GSEngine as Ruby constants to the JRuby interpreter runtime (see: Table 2-1).

Attribute name	*Description*	*Value Datatype*	*Who provides*	*Who uses*	*Ruby global constant name*
Optimization Policy	Indicates goal(s) of GrAppO. URL to an optimization policy configuration file is needed.	URL	EPE/EMI	GrAppO	GS_OPTIMIZATION_POLICY
GrAppO URL		URL	EPE/EMI	GOI	GS_GRAPPO_URL
GRR Base URLs		list of URLs	EPE/EMI	GOI	GS_GRR_BASE_URL
DAS URL		URL	EPE/EMI	DAC	GS_DAS_URL
PROToS URL		URL	EPE/EMI	M-Ring GSMP-enabled log appender	GS_PROTOS_URL
DOS URL		URL	EPE/EMI	Future DOS client	GS_DOS_URL
User handle	A SAML portion with Shibboleth user handle	String	EPE/EMI	delegation to DAC and GOI for accessing resources	GS_USER_HANDLE
ACID	Application Correlation Identifier (ACID)	String	GSEngine	M-Ring GSMP-enabled log appender	GS_ACID

Table 2-1 GridSpace Engine Application Context that contains all the parameters required by the GSEngine libraries (such as GOI and DAC) that is passed by GSEngine as Ruby constants to the JRuby interpreter runtime.

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

GridSpace Engine is intended to support a set of evaluation request types such as request for the evaluation of local file GScript, request for the evaluation of explicitly provided GScript, and finally, of the script staged in the Application Repository. The evaluation request types set is open and may be expanded in the future by providing an appropriate evaluation request subclass and the corresponding evaluation request support subclass following the design pattern shown in Figure 2-2.

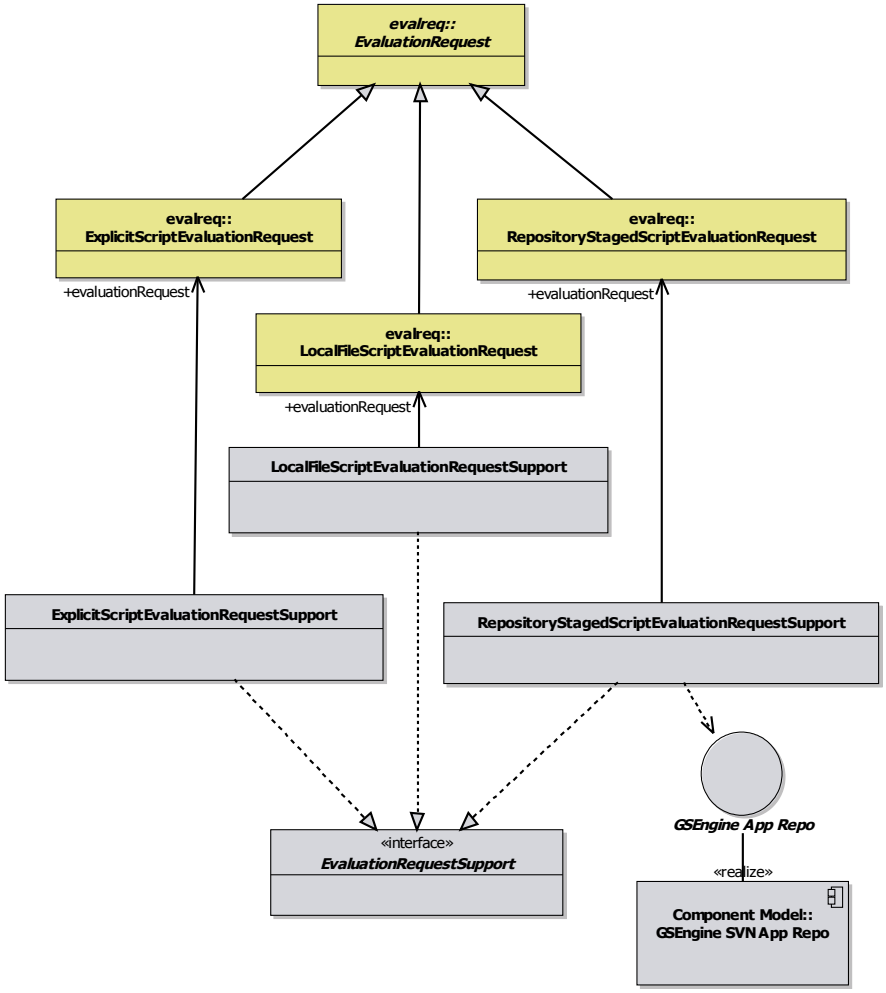

Figure 2-2: Class diagram of supported types of evaluation request. The evaluation request types set is open and may be expanded in the future by providing an appropriate evaluation request subclass and the corresponding evaluation request support subclass following such a design pattern.

2.1.1. GSEngine API

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 other main classes of GSEngine API 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 exepriment 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:
  - `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.

2.1.2. GSEngine Core

GSEngine Core is the core part and the facade for GSEngine in the same time. It implements GSEngine API. The diagram from Figure 2-3 shows the structure of the GSEngine Core.

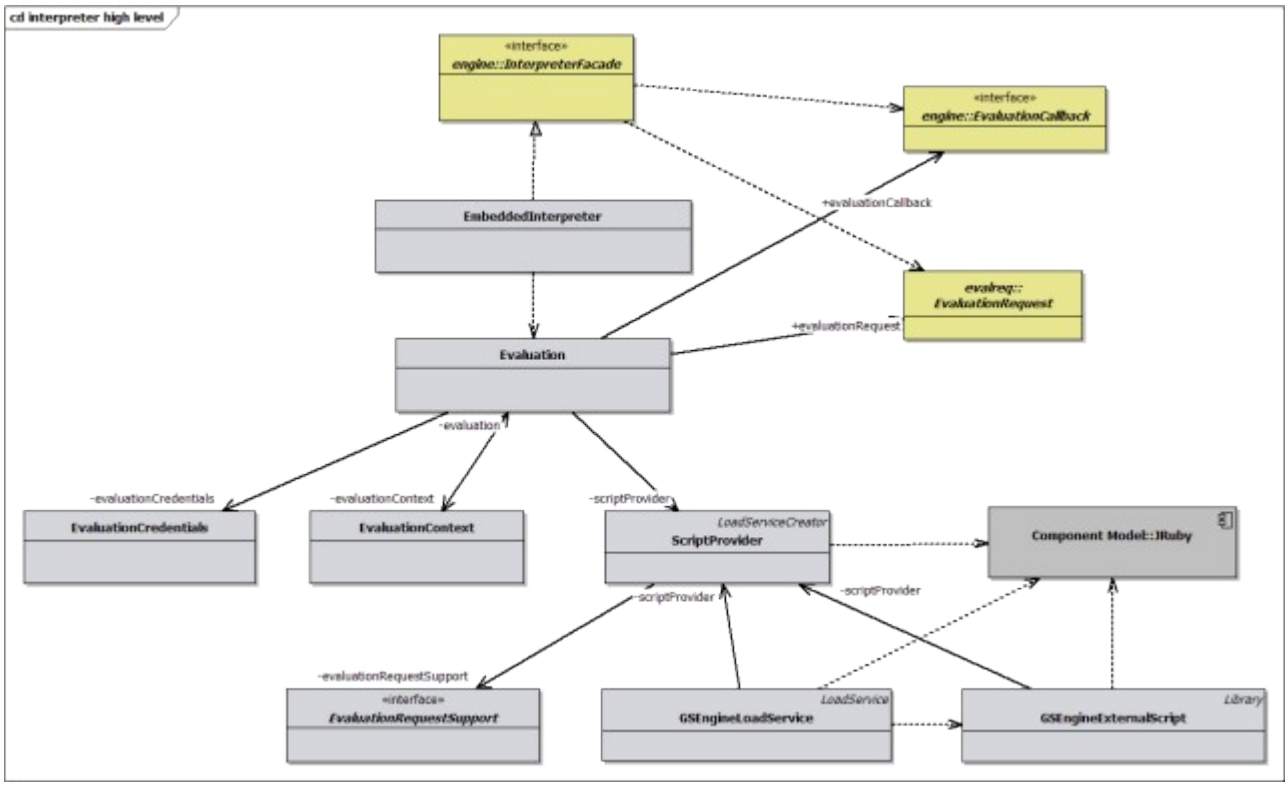

Figure 2-3: GSEngine Core main classes with marked relations with GSEngine API classes, which are coloured as yellow.

`EmbeddedInterpreter` is the root class of GSEngine Core that realizes the `InterpreterFacade` interface. It delegates resposibility of a processing of the evaluation request to the Evaluation class objects that are instantiated one per each evaluation request. `Evaluation` object takes care of evaluation request throughout its whole lifecycle:

- First, `Evaluation` generates `EvaluationCredentials` - specific attributes such as ACID (Application Correlation Identifier) or GSEID (GSEngine Identfier) which are assigned to the the concrete application evaluation.
- Second, `Evaluation` creates `ApplicationContext` which contains of attributes which provide context information for the GSEnigne runtime libraries such as DAC and GOI. The library code is enabled to access the Application Context attributes just like any other Ruby constants. For the details about attributes please refer to Section 2.1.
- Next, `Evaluation` instantiates `ScriptProvider`. `ScriptProvider` is a factory for `GSEngineLoadService` which role is to provide on JRuby interpreter's demand the code of the application (hereby realizes JRuby `LoadService interface`). At the runtime, when the demand for source code of external file occurs, `GSEngineLoadService` delegates the source code request to the appropriate `EvaluationRequestSupport` (discussed later), wraps source code as a JRuby library (realization of JRuby `Library` interface) and load it to the JRuby runtime.

- Finally, `Evaluation` instantiates ruby runtime object with appropriate `ScriptProvider` set and appropriate path to ruby libraries set. Then, ruby runtime object performs actual evaluation of the application.

The subsequent step of the evaluation scenario are depicted in the sequence diagram from Figure 2-5.

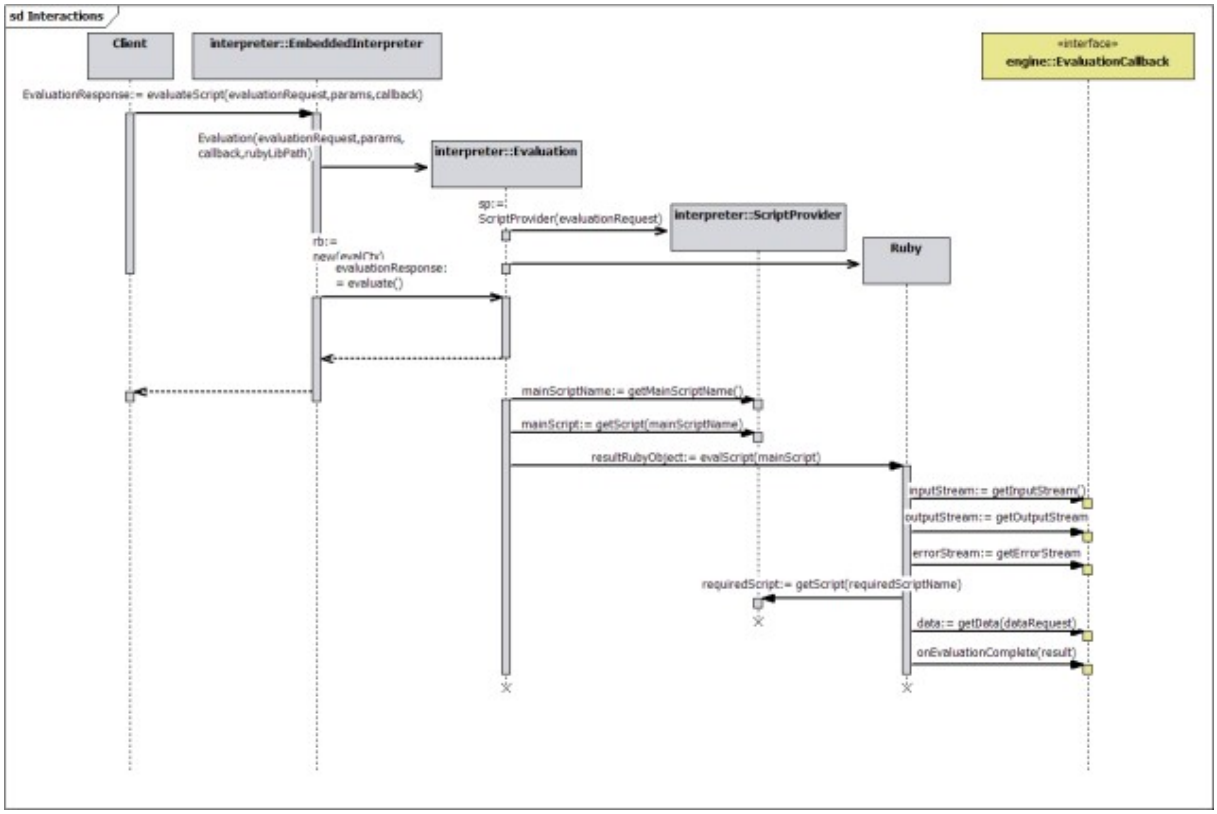

Figure 2-4: Sequence diagram of the evaluation scenario.

As long as GSEngine Core uses JRuby interpreter for evaluation of the application with sources taken from arbitrary location, JRuby has to be extended with GSEngine Core classes. GSEngine Core classes realizes JRuby interfaces as shown in Figure 2-5.

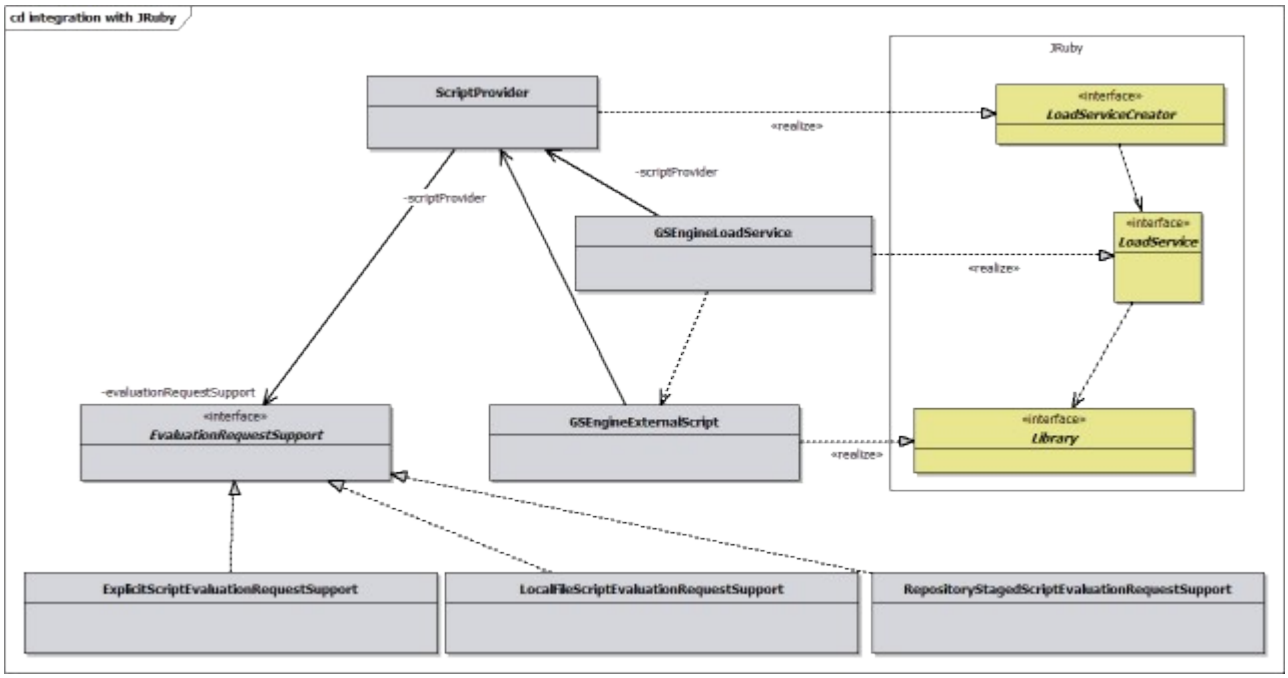

Figure 2-5: GSEngine Core classes extending JRuby interfaces in order to provide application code from an arbitrary location. The JRuby interfaces are coloured as yellow.

As mentioned earlier, `ScriptProvider` realizes JRuby `LoadServiceCreator` interface, `GSEngineLoadServic`e realizes JRuby `LoadService` interface. `GSEngineLoadService` provides source code and wraps it as a `GSEngineExternalScript` object that realizes JRuby `Library` interface. The appropriate `ScriptProvider` instance is provided to the ruby runtime object so it is used by JRuby interpreter as a script provider.

The architecture of GSEngine Core is intended to support any of possible types of evaluation requests. Custom evaluation request is obliged to extend the `EvaluationRequest` superclass. Moreover, it has to be provided corresponding realization of `EvaluationRequestSupport` interface, and the mapping between aforementioned two has to be stored in the `ScriptProviderClass`.

For the present, all of the three types defined in the GSEngine API are supported, as shown in Figure 2-6.

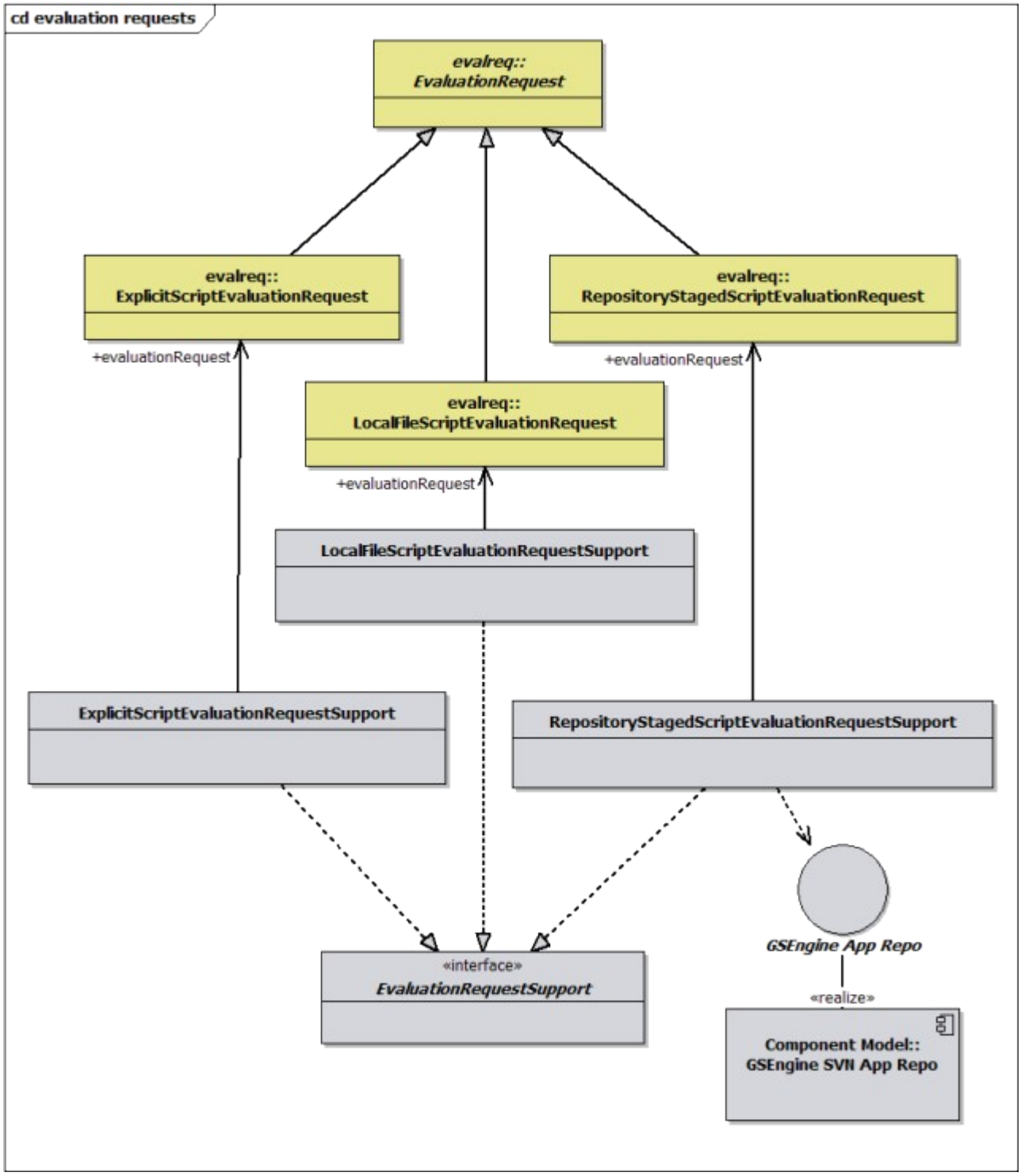

Figure 2-6: Supporting various evaluation request types – class diagram. Classes of GSEngine API are coloured as yellow.

2.1.3. GSEngine Application Repository Client API

GSEngine Application Repository Client API is a generic API of a GSEngine component that accesses application repositories (e.g. experiment repository).

Figure 2-7 contains a diagram of classes that belong to the client API (situated in package `cyfronet.gridspace.engine.apprepo`), described further in this section.

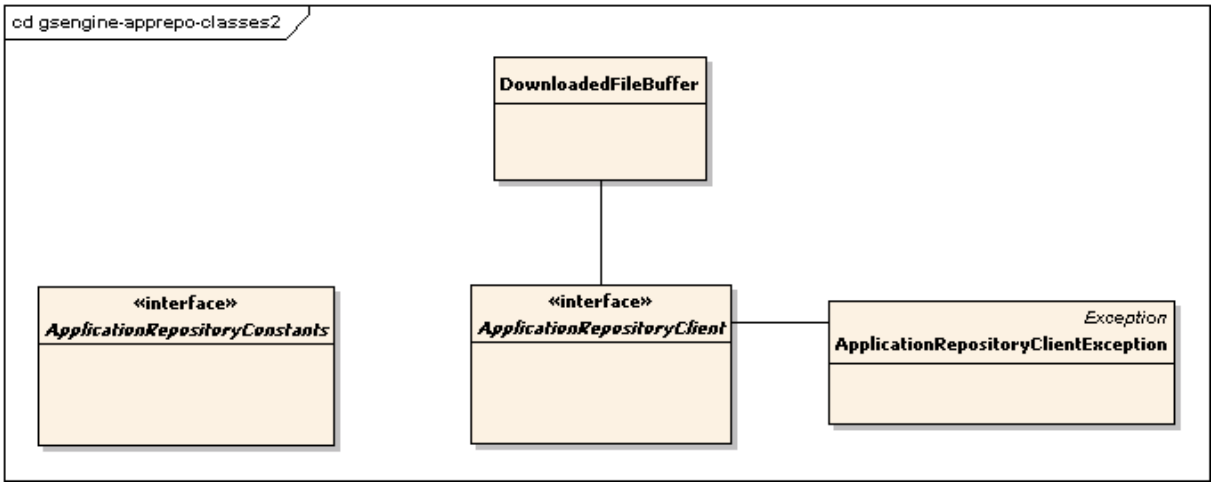

Figure 2-7: GSEngine Application Repository Client API classes.

The client API provides an interface (`ApplicationRepositoryClient`), that declares methods for accessing application repository contents for the application users as well as for their developers.

The interface `ApplicationRepositoryConstants` stores all the values specific to the application repositories as opposed to ordinary file repositories (e.g. names of the directories that constitute the application's structure and are common to all applications in all application repositories).

2.1.4. GSEngine SVN Application Repository Client

GSEngine SVN Application Repository Client is an implementation of GSEngine Application Repository Client API intended for accessing Subversion [SVN] repositories. The library only implements the API methods and does not provide any extensions to it.

Figure 2-8 shows the GSEngine SVN Application Repository Client classes and their relations to the structures provided by client API (GSEngine Application Repository Client API).

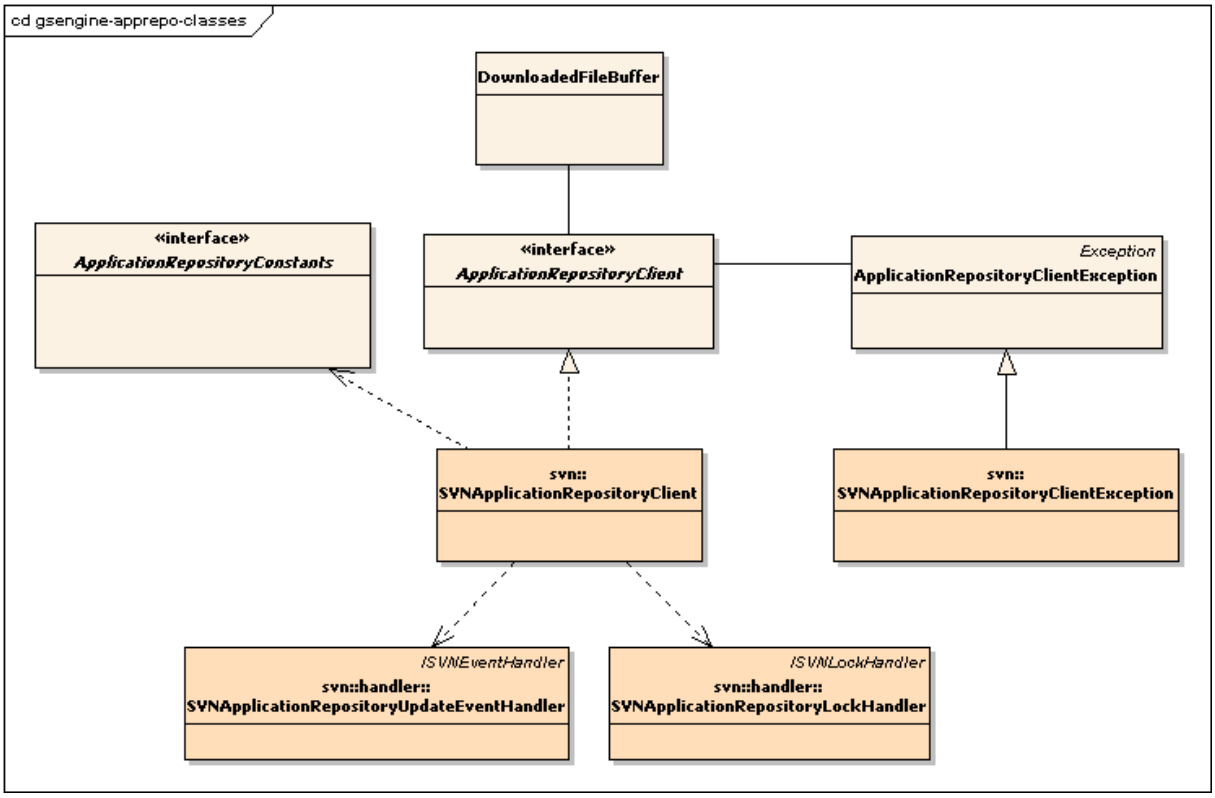

Figure 2-8: GSEngine Application Repository Client classes.

The main class of the library is `SVNApplicationRepositoryClientException`, which implements the `ApplicationRepositoryClient` interface. In case of errors a `SVNApplicationRepositoryClientException` is thrown (an extension of `ApplicationRepositoryClientException`). The `*Handler` classes define the client's behavior when one of update (`SVNApplicationRepositoryUpdateEventHandler`) or lock (`SVNApplicationRepositoryLockHandler`) is called on the repository.

For the detailed description of all the GSEngine SVN Application Repository Client classes and methods please see code documantation (javadoc) of packages `cyfronet.gridspace.engine.apprepo.svn.*`.

2.2. ALTERING THE FUNCTIONALITY OF THE GRIDSPACE ENGINE

2.2.1. Adding New Java or Ruby Library Dependencies of the Engine

Once you have installed GridSpace Engine with an environment variable pointing at its home directory (`$GS_HOME` - in the case of Linux, `%GS_HOME%` - in the case of Windows) you can add there the additional dependencies of both Java and Ruby libraries you need while extending GridSpace Engine.

Having added external libraries (both Ruby and Java) you're enabled to use them from the GridSpace application source code level.

Adding Ruby library dependency. In order to use external (not built-in within GridSpace Engine) Ruby libraries all you need it to copy them to the `$GS_HOME/ruby` or `%GS_HOME%\ruby` according to OS, Linux and Windows, respectively.

Alternatively, you may add external libraries to the JRuby distribution (`$JRUBY_HOME/lib/ruby`), although, consider that it make libraries shared in all application launched by a jruby interpreter.

Adding Java library dependency. In order to use external (not built-in within GridSpace Engine) Ruby libraries all you need it to copy them to the `$GS_HOME/java` or `%GS_HOME%\java` according to OS, Linux and Windows, respectively. That will result in including jars in the classpath, GridSpace Engine command line tool. However, when using GridSpace Engine via its Java API all the jars from the `$GS_HOME/java/%GS_HOME%\java` directory stil have to be present in the Java application classpath.

2.2.2. Adding Support for New Remote Computation Technologies

GridSpace supports leading Grid middleware technologies. Currently, GridSpace users can employ Web Services, MOCCA Grid components, EGEE jobs (LCG) and WTS([WTS]) in experiments. It is planned to implement AHE and WSRF client side libraries. Moreover, GOI and GRR design and implementation makes enriching GSEngine with support for new technologies absolutely undemanding. It requires only two things to be done:

1. extending Grid Operation Invoker by implementing adapter and resource classes and placing them in appropriate directory
2. extending Grid Resource Registry by inserting technical information describing new technology

This manual provides a step-by-step guide how to extend run-time capabilities of GridSpace by adding support for new middleware technologies.

Extending Grid Operation Invoker

Grid Operation Invoker support for various middleware technologies is based on adapters concept. GOI can be extended by adding adapter and resource classes for new technology. Adapter is a factory that is capable to produce representatives for all Grid Object Instances published in one concrete middleware technology. Such representative should be an object of a resource class, that extends *GridResource* class that can be found in *$GS_HOME/ruby/cyfronet/gridspace/goi/adapters/* directory. Resource object is a proxy for Grid Object Instance and is able to interact with it in its specific protocol, but from the experiment developer's point of view it acts as any other Ruby object. Adapter class must expose one class method, *create_instance*, which takes *Hash* object containing technology data and returns Grid Object Instance representative of TechnologyResource class. Resource class must implement only one method, *initialize* taking Hash with technology information. This method semantics is very similar to Java constructor, it is called after Ruby allocates memory for an objects and all arguments passed to *new* are passed to *initialize* method that sets up object state. The first thing that needs to be done in the body of this method is calling the *initialize* method of super class (*GridResource*) with he following code:

```
def initialize(techInfo)
  super(techInfo)
```

```
        #method body
    end
```

Next, TechnologyResource object must be made capable to handle invocations of operations on Grid Object Instance. Developer is not limited in the way how to achieve it, albeit it is strongly recommended to implement the *method_missing* operation, which takes responsibility for delegating operation execution to Grid Object Instance and returning result.

Code snippets below present the simplest adapter and resource classes that support Web Service technology and explains what activities are performed.

```
require 'cyfronet/gridspace/goi/adapters/ws_resource.rb'

require 'java'

if !defined? JLogger

  include_class('org.apache.log4j.Logger'){|package,name|"J#{name}"}

end

class WsAdapter

  @@logger = JLogger.getLogger('goi.adapter.ws')

  def WsAdapter.create_instance(wsTechInfo)

    @@logger.debug('Using service ' + wsTechInfo['endpoint'])

    gridObjectInstance = WsResource.new(wsTechInfo)

    return gridObjectInstance

  end

end
```

Lets inspect what is done in this adapter:

1. *WsResource* class is required
2. Java support is enabled by requiring 'java'
3. log4j class is included if it not already defined
4. adapter class is defined
    - class variable, @@logger is defined that is used to print debug information and in future will be used to integrate GOI with monitoring infrastructure
    - representative for Grid Object Instance is created and retuned

```
require 'cyfronet/gridspace/goi/adapters/grid_resource'

require 'java'

if !defined? JLogger

  include_class('org.apache.log4j.Logger') {|package,name| "J#{name}"}

end

class WsResource < GridResource

  attr_reader :soap

  def initialize(techInfo)

    super(techInfo)

    # ...

  end

  def method_missing(methodSymbol, *args)

    # ...

  end

end
```

The code above:

1. requires *GridResource* class, enables Java usage and include log4j class
2. WsResource class is defined
    - it extends *GridResource*
    - defines local variable soap and create getter method for it
    - defines *initialize* method (body of this method is not included)
    - defines *method_missing* that delegates operation invocations to Grid Object Instance and returns result (body of this method is not included)

Source code of *WsAdapter* and *WsResource* classes can be found in *ws_adapter.rb* and *ws_resource.rb* files placed in the *$GS_HOME/ruby/cyfronet/gridspace/goi/adapters* directory.

Since adapter class is required during run-time, adapters' class names must obey certain naming convention. Due to the fact, that Ruby requires files, which names does not necessarily correspond to the name of class that is inside, naming convention is also imposed on file names. The naming pattern is as follows:

- Adapter class name is a concatenation of two words. Technology name, starting with capital letter with all following letters in lower case, and 'Adapter'. For instance, adapter for Web Services is named *WsAdapter* (it could be named *WebserviceAdapter* as well, but *WebServiceAdapter* is not a valid name), adapter for MOCCA is named *MoccaAdapter*.

- File name should reflect the name of the adapter class it contains according to the Ruby convention, that is to use only lower case letters and underscore characters (uppercase letter in class name must be replaced with underscore and lower case letter). Files must have 'rb' extension. File names for adapters mentioned above are *ws_adapter.rb* and *mocca_adapter.rb*.

It is not obligatory to name TechnologyResource classes in accordance to the mentioned naming convention, although it is profitable to name them analogically to adapter classes. For instance classes of representatives for Web Services is named WsResource and for MOCCA components is named *MoccaResource*, files contaning these classes are *ws_resource.rb* and *mocca_resource.rb*.

Files containing adapter and resource classes should be placed in *$GS_HOME/ruby/cyfronet/gridspace/goi/adapters* directory. If any additional JRuby classes are used they should be placed in *$GS_HOME/cyfronet/gridspace/goi/utils*. These files should be required within adapter and resource classes like this:

```
require 'cyfronet/gridspace/goi/utils/additional_class_name'
```

Adapter class can be implemented in JRuby, therefore it can contain pure Ruby code, as well as include and use Java objects. If any Java classes, other than those provided by Java Runtime Environment, are used, jar files containing theses classes should be placed in *$GS_HOME/java/ directory*.

For more information on Ruby language please refer to [PR] and [DPIR].

Extending Grid Resource Registry

Grid Resources Registry is responsible for storing information about any accessible resources in ViroLab environment. What is more, it hides resources technologies complexity from the user. That is why if new technology is added to GOI, GRR has to be extended by adding support for this technology.

Before new technology will be added to GRR by the user, he or she has to be able to download, compile and pack GRR. For more information about this processes see online GRR tutorial [START_EXT_GRR].

To add new technology to GRR a few steps are needed. At the beginning the name of the new technology has to be added to enum class `cyfronet.gridspace.grr.GridObjectType` located in `grr-dto` GRR module. Currently there are available following technologies:

```
public enum GridObjectType {
  /** Web Service (WS) */
```

```
  WS,

  /** Metacomputing-oriented CCA Framework (MOCCA) */
  MOCCA,

  /** Local gem */
  LOCAL_GEM,

  /** Web Services Resource Framework (WSRF). Not implemented yet,
   * but planned.
   */
  WSRF,

  /** Grid job. Not implemented yet, but planned. */
  JOB
}
```

All information about resources technologies infos are stored in database and hibernate [HIBERNATE] framework is used to map this information to object oriented Java [JAVA] classes and vice versa. New class that will be used to map information about new technology from object oriented to relations database world should be placed in `cyfronet.gridspace.grr.businessobject.technology` package of the `grr-hbn` GRR module.

A simple example class that describes new technology is presented below.

```
@Entity                                  // 1
@DiscriminatorValue("NewTechnology")  // 2
public class NewTechnologyImplementation extends
 GridObjectImplementation {  // 3
  private String additionalInformation;  // 4

public void setAdditionalInformation(String additionalInformation)
{  // 5
    this.additionalInformation = additionalInformation;
}

public String getAdditionalInformation(String additionalInforma-
tion)
{  // 5
    return additionalInformation;
}
```

```
@Override
public Map<String, String> getTechnicalInfo()
{
  Map<String, String> technicalInfo = super.getTechnicalInfo();
  //put some additional technical inforamtion
  technicalInfo.put("additionalInformation",additionalInformation)
;//6
  return technicalInfo;
}
}
```

New technology class has to have additional information that is needed by hibernate [HIBERNATE] and GRR mechanisms to create technical info for GOI:

1. Class has to define `@Entity` annotation that is used to map class instance to relations database structure
2. Class has to define `@DiscriminatorValue("NewTechnology")` annotation. It is necessary for hibernate inheritance mechanism. This annotation takes one *String* parameter that has to be specific for concrete technology.
3. Class has to extend `cyfronet.gridspace.grr.businessobject.GridObjectImplementation` that defines basic information about technology and annotations necessary to map GridObjectImplementation and all classes that extend it to relational database.
4. Class can add additional information (parameters) specific for new technology
5. All additional parameters have to be accessible through getters and setters methods
6. And at the end if there is technology specific information, new technical info for GOI has to be created

After new technology class is created then this class has to be added to hibernate configuration file (`spring-config.xml`) that is located in `grr-ws` module:

```
<bean id="sessionFactory"
class="org.springframework.orm.hibernate3.annotation.AnnotationS-
essionFactoryBean">
  <property name="dataSource"><ref bean="mysqlDataSource"/></prop-
erty>
  <property name="hibernateProperties">
    <ref bean="hibernateProperties"/>
  </property>
  <property name="configurationClass">
```

```
    <value>org.hibernate.cfg.AnnotationConfiguration</value>
  </property>
  <property name="annotatedClasses">
<value>
  cyfronet.gridspace.grr.businessobject.technology.WSImplementa-
tion
</value>
<value>
  cyfronet.gridspace.grr.businessobject.technology.MOCCAImplemen-
tation
</value>
<!-- new implementation -->
<value>
cyfronet.gridspace.grr.businessobject.technology.NewTechnologyIm-
plementation
</value>
<!-- end -->
   </list>
 </property>
</bean>
```

At the end new registry has to be compiled, packed and deployed to tomcat container (see start extending GRR online tutorial [START_EXT_GRR] and section 3.1 for more infos).

2.2.3. Adding New Types of Data Sources through DAC Connectors

Extendability of DAC to cover new data sources is currently achieved by extending the DACConnectClass constructor to handle additional types of source labels. DACConnectClass is a JRuby class and can be edited as present in the code of the GSEngine Runtime System (it is not compilable). In particular, it is possible to extend DAC to handle additional types of relational databases. This is done by inserting an appropriate JDBC plugin (usually a JAR archive) in the classpath of GSEngine and adding a clause within DACConnectClass instructing the module to use that plugin for a given **db_tech** input value. While this still requires modifying DAC code, it is envisioned that a more automated data source plugin handling mechanism will be present in the final version of DAC.

It is currently planned to extend DB tech support with dynamic data storage and object-based representation (as Data Objects) inside the experiment scripts.

2.2.4. Adding New Experiment Execution Optimization Techniques

Introduction.

A module responsible for optimization of an application execution in GSEngine is called Grid Application Optimizer (in short GrAppO). The algorithm used for optimization are executed by a dedicated GrAppO component - Optimization Engine. Such algorithm has to be an implementation of an interface (namely `cyfronet.gridspace.grappo.engine.OptimizationAlgorithm`), therefore allowing multiple realizations of an optimization algorithm to be pluggable into the optimizer. Which algorithm implementation will be used for optimization is a matter of external configuration. However, there always is a default optimization algorithm available. On Figure 2-9 a sample relation between the general optimization algorithm and its different implementations is shown.

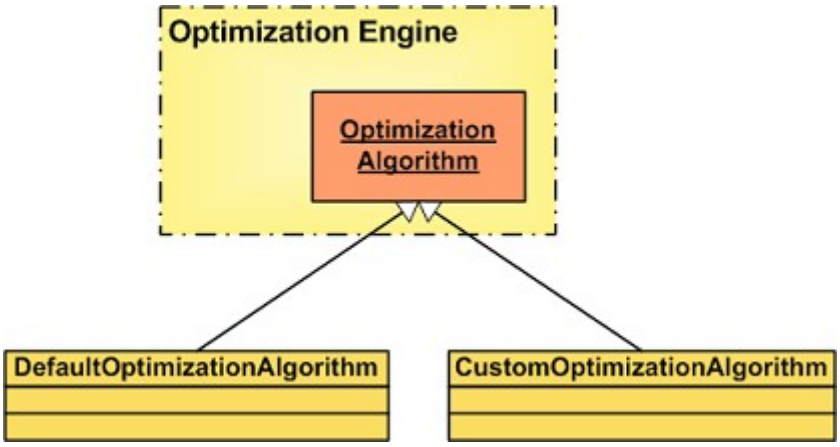

Figure 2-9: Optimization algorithms in GrAppO.

OptimizationAlgorithm Interface.

The interface `cyfronet.gridspace.grappo.engine.OptimizationAlgorithm` declares the following two methods:

- `OptimizationResult` **`optimize`**`(GObDataSet optimizationData) throws OptimizationException`

for optimization concerning one Grid Object at a time (so called *short-sighted optimization*), and

- `List<OptimizationResult>` **`optimize`**`(List<GObDataSet> optimizationData) throws OptimizationException`

for optimization of a group of Grid Objects.

Each algorithm that is going to be used by Grid Application Optimizer has to be developed as a class implementing the `OptimizationAlgorithm` interface, and thus the both mentioned methods. The methods results and parameters will be described further in this section.

`OptimizationResult` (`cyfronet.gridspace.grappo.OptimizationResult`) returned in both methods is an object wrapping the result of optimization of one Grid Object - either an identifier of Grid Object Instance or an identifier of Grid Object Implementation with resource location where the implemenation will be

deployed. An `OptimizationResult` class instance can be created in the body of each method with one of the constructors:

- `OptimizationResult(String gobClassName, Long gobInstanceID)`

or

- `OptimizationResult(String gobClassName, Long gobImplementationID, String resourceURI)`

`GObDataSet` (`cyfronet.gridspace.grappo.gob.GObDataSet`) that appears as a parameter in both optimization methods (`optimize(...)`) carries all the data concerning available Grid Object Implementations of a given Grid Object Class, their Grid Object Instances and resource locations. The class provides methods:

- `Map<Long, GridObjectImplementation>` **`getGridObjectImplementations`**`()`
- `Map<Long, GridObjectInstance>` **`getGridObjectInstances`**`()`
- `Map<String, H2OKernel>` **`getKernels`**`()`

that return respectively:

- a map (`java.lang.Map`) of Grid Object Implementation representations, with implementation's identifiers as keys,
- a map (`java.lang.Map`) of Grid Object Instance representations, with instance's identifiers as keys,
- a map (`java.lang.Map`) of resource representations, with their locations as keys (for now only the H2O kernels are taken into consideration).

The representations of the mentioned entities are simple classes located in package `cyfronet.gridspace.grappo.gob`. Their detailed API can be browsed at [GRAPPO-API].

The `GObDataSet` class offers also rankings of all its contents, which can be used through methods:

- `void` **`addGObImplementationRankingPoints`**`(long id, int points)`
- `int` **`getGObImplementationRankingPoints`**`(long id)`

- `void` **`addGObInstanceRankingPoints`**`(long id, int points)`
- `int` **`getGObInstanceRankingPoints`**`(long id)`

- `void` **`addKernelRankingPoints`**`(String url, int points)`
- `int` **`getKernelRankingPoints`**`(String url)`

where `points` is the number of ranking points for a GObImplementation/GObInstance/resource with given `id/url` (the latter in the case of a resource).

Note: Currently (up to version 0.3.0 of GSEngine), the Grid Resources Registry does not provide information about available resources (H2O kernels) and their locations, therefore the `getKernels()` and `get/addKernelRankingPoints(...)` methods should not be used.

Optimization Algorithm Configuration.

The class providing implementation of `OptimizationAlgorithm` has to be specified either in optimization policy configuration file or as a field of `OptimizationPolicy` object, passed to Grid Application Optimizer in its constructor. The situations are illustrated by Figure 2-10 and Figure 2-11, respectively.

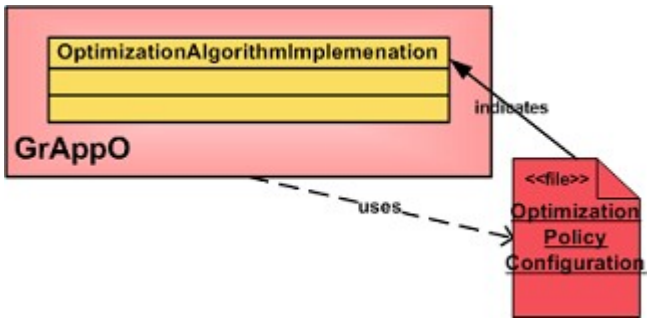

Figure 2-10: Configuring GrAppO with a file.

In the first case the GrAppO constructor is called with a String parameter pointing to location of a configuration XML or `.properties` file. The file should contain *Optimization Policy* configuration, including a line:

```
<properties>
...
  <entry
    key="optimization.algorithm">qualified.AlogrithmClassName</en-
try>
</properties>
```

if it is an XML file, or

```
optimization.algorithm=qualified.AlogrithmClassName
```

in case of a `.properties` file. The `qualified.AlogrithmClassName` should be replaced with a qualified name of the class implementing the `OptimizationAlgorithm` interface, for example :

```
cyfronet.gridspace.grappo.sample.RandomOptimizationAlgorithm.
```

It is important that the class should have a default constructor. If the described option is not specified, a default (random) algorithm will be used.

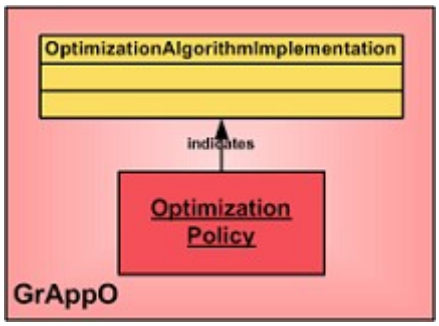

Figure 2-11: Configuring GrAppO with an optimization policy object.

The second way to configure an optimization algorithm is to use its constructor in which the parameter is of `OptimizationPolicy` (`cyfronet.gridspace.grappo.configuration.OptimizationPolicy`) class. The supplied parameter has to be an `OptimizationPolicy` class instance (created with a default constructor) with `optimizationAlgorithm` field set to the algorithm of choice. To do this, an appropriate setter method is provied, namely:

- `void` **`setOptimizationAlgorithm`**`(OptimizationAlgorithm optimizationAlgorithm)`

in which the parameter should be an instance of a class implementing the `OptimizationAlgorithm` interface.

Other Features Configured with Optimization Policy.

Apart from the optimization algorithm implementation, optimization policy serves to configure the following features:

1. Preferred implementation type – defines which service implementation technology should be preferred to others (e.g. the preference could be given to services implemented as MOCCA components). It also says whether other than preferred implementation technologies will be allowed
    - property key:
        - type preferrence: `preferred.service.type`
        - allowing other implementation technologies:
          `void` **`setPreferredType`**`(GridObjectType preferredType)`
    - `OptimizationPolicy` setter method:
        - type preferrence: `preferred.strict`
        - allowing other implementation technologies:
          `void` **`setStrictTypePreference`**`(boolean strictTypePreference)`
2. Whether Monitoring / Provenance Tracking System should be contacted to obtain useful optimization data or the connection shouldn't be allowed
    - property key: `use.monitoring` / `use.provenance`

- `OptimizationPolicy` setter methods:
  `void` **`setUseMonitoring`**`(boolean useMonitoring)`
  `void` **`setUseProvenance`**`(boolean useProvenance)`

Additional Information :

- Grid Application Optimizer API [GRAPPO-API]
- A MSc Thesis on the optimization of a Grid application execution [OPT-THESIS]

2.3. ACCESS TO THE MONITORING AND PROVENANCE EVENTS SYSTEM

All the modules incorporated into GridSpace Engine which indeed take part in the application evaluation are enabled to emit events related to it. Such events may be forwarded through the monitoring data bus and reach monitoring data consumers such as a provenance system.

As long as entirely all modules of GridSpace Engine are written either in Java or Ruby languages they have an access to the Java classes. Therefore, GridSpace Engine uses log4j [LOG4J] along with a dedicated logging category, namely `monitoring`, with assigned dedicated logging appender that constitutes an entry point to the monitoring data bus.

The following example shows the sample Ruby method which emits the message from withing application evaluation, using the Application Correlation ID (ACID).

```
def log(msg)
  Logger.getLogger("monitoring").info('Message from an application
  '+ GS_ACID + ': ' + msg)
end
```

For the present, they are not any established set of events types that are recognizable for the monitoring and provenance system. In the future the event types will be defined and the Java classes representing events of each defined type will be provided.

2.4. SOURCE CODE ACCESS, BUG REPORTING AND AUTHORS CONTACT INFORMATION

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

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

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*].

3. GRIDSPACE SERVERS AND PORTLETS ADMINISTRATION MANUAL

3.1. GRID RESOURCES REGISTRY INSTALLATION MANUAL

Below there is a list of instructions needed to run the Grid Resources Registry.

1. Download, install and set the home directory of the Tomcat server [TOMCAT]. The preferred version of the Tomcat server is 5.5.x or 6.0.x and it can be downloaded from the Tomcat site (http://tomcat.apache.org). After unpacking the distribution, set the environmental variable `CATALINA_HOME` so it points to the Tomcat home directory.

2. Download the newest version of Grid Resources Registry from maven 2 repository (notice that `SNAPSHOT` version is not stable)

```
#debian based linux systems
wget http://virolab.cyfronet.pl/maven2/cyfronet/gridspace/grr/grr-ws/$GRR_VERSION/grr-ws-$GRR_VERSION.war
```

3. install mysql

```
#debian based linux systems
sudo apt-get install mysql-server
```

4. Create new database and database user

```
create database database_name;
grant all on database_name.* to 'user_name'@'localhost' identified by 'password'
```

5. Unwar downloaded GRR distribution to

```
#notice that war is a zip archive
$CATALINA_HOME/webapps/grr_name
```

6. Edit spring-config.xml

```
vim $CATALINA_HOME/webapps/grr_name/WEB-INF/spring-config.xml
```

7. Set correct database name, user name and password

```
<bean id="mysqlDataSource" class="org.springframework.jdbc.data-source.DriverManagerDataSource">
  <property name="driverClassName">
      <value>com.mysql.jdbc.Driver</value>
  </property>
  <property name="url">
      <value>jdbc:mysql:///database_name</value>
  </property>
```

```
  <property name="username"><value>user_name</value></property>
  <property name="password"><value>password</value></property>
</bean>
```

8. Uncomment

```
<prop key="hibernate.hbm2ddl.auto">create</prop>[1]
```

9. Start Tomcat

```
$CATALINA_HOME/bin/startup.sh (linux)
$CATALINA_HOME\bin\startup.bat (windows)
```

10. Stop Tomcat

```
$CATALINA_HOME/bin/shutdown.sh (linux)
$CATALINA_HOME\bin\shutdown.bat (windows)
```

11. Comment (we don't want database structure to be created every time Tomcat is (re)started)

```
<!--<prop key="hibernate.hbm2ddl.auto">create</prop>-->
```

12. Start Tomcat

```
$CATALINA_HOME/bin/startup.sh (linux)
$CATALINA_HOME\bin\startup.bat (windows)
```

13. GRR runs, see `localhost:8080/grr_name/services` for available web services WSDL's

3.2. WEB RESOURCES REGISTRY BROWSER INSTALLATION MANUAL

Below there is a list of instructions needed to run the Grid Resources Registry.

1. Download, install and set the home directory of the Tomcat server. The preferred version of the Tomcat server is 5.5.x or 6.0.x and it can be downloaded from the Tomcat site (http://tomcat.apache.org). After unpacking the distribution, set the environmental variable `CATALINA_HOME` so it points to the Tomcat home directory.

2. Download the newest version of Grid Resources Registry from maven 2 repository (notice that `SNAPSHOT` version is not stable)

[1] If this property is uncommented, a database structure will be created every time Tomcat is (re)started

```
#debian based linux systems
wget http://virolab.cyfronet.pl/maven2/cyfronet/gridspace/grr/grr-web-browser/$GRR_VERSION/grr-web-browser-$GRR_VERSION.war
```

3. Unwar downloaded GRR Web Browser distribution (notice that war is a zip archive)

```
#notice that war is a zip archive
$CATALINA_HOME/webapps/grr_name
```

4. Open `$CATALINA_HOME/webapps/grr_web_browser_name/WEB-INF/classes/properties.xml` file and set VO configuration properties file url.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
  <comment>GRR Web Browser properties</comment>
  <entry key="voconfig.url">
     http://virolab.cyfronet.pl/config/properties-SNAPSHOT.xml
  </entry>
</properties>
```

5. (Re)start Tomcat

```
#linux
$CATALINA_HOME/bin/shutdown.sh
$CATALINA_HOME/bin/startup.sh

#windows
$CATALINA_HOME\bin\shutdown.bat
$CATALINA_HOME\bin\startup.bat
```

6. Web browser is available at `http://localhost:8080/grr_web_browser_name/cyfronet.gridspace.grr.web.browser.ResourcesBrowser/ResourcesBrowser.html`

3.3. DOMAIN ONTOLOGY STORE INSTALLATION MANUAL

The setup of a new installation of the Domain Ontology Store takes several required components:

- a web server able to serve the web application front-end

- an ontology store that operates as the main server part that provides the business logic
- the ontology models themselves that should be loaded into the store to populate it
- (optional, yet strongly recommended) a database backend for performance, scalability and persistence reasons.

The manual below provides instructions how to set up such a store with specific tools. It is possible that some of those tools could be exchanged for different ones. Although this manual only covers Linux installation, the same procedure should be similar for other operating systems as 100% of software used is developed in Java (apart from the optional back-end database).

3.3.1. Downloading tools

In order to setup an environment ready for DOS installation, please download:

- Java JRE, the latest update in 1.5.0 version
  - `http://java.sun.com/javase/downloads/index_jdk5.jsp`
  - you may also try to use ready, on-board Java usually provided with many Linux distributions
- Jetty Web Server (tested with 6.0.1)
  - `http://jetty.mortbay.com/`
- Sesame RDF store (tested with 1.2.7)
  - `http://openrdf.org/`

For optional (through encouraged) MySQL database back-end support, you'll also need:

- MySQL JDBC Java Connector (tested with 5.0.7)
  - `http://dev.mysql.com/downloads/connector/j/5.0.html`

3.3.2. Installation and running

Java JRE installation.

- Change the rights of the downloaded JRE file to have it executable
- Run the file and answer **yes** for the license agreement
  - The package will decompress to your current directory with a proper subdirectory name
- Set the **`JAVA_HOME`** environment variable to point to that subdirectory
- If you like, add **`JAVA_HOME/bin`** entry to your **`PATH`** variable for convenience

Sesame and Jetty installation.

- Decompress both **`jetty-x.y.z.zip`** and **`sesame-x.y.z-bin.tar.gz`** packages to separate directories
    - There should be a **`sesame.war`** in the **`sesame-x.y.x/lib`** subdirectory
- Create a **`sesame`** subdirectory inside **`jetty-x.y.z/webapps`**
- Decompress the **`sesame.war`** file inside the newly created **`webapps/sesame`** location
- Edit **`jetty-x.y.z/etc/jetty.xml`** to change important settings, like e.g. server port number, log file etc.
- Copy **`webapps/sesame/WEB-INF/system.conf.example`** to **`webapps/sesame/WEB-INF/system.conf`** for a fresh configuration
- Edit the **`system.conf`** file and add any users you'd like in **<userlist>** part
    - other setting changes there are also possible

Running the server.

After these steps you should be ready to start a fresh ontology server

- Go to **`jetty-x.y.z/`**
- Type: **`java -jar start.jar etc/jetty.xml`**
    - This should make it run; check with your browser: `http://your.server.address:jettyport/sesame/index.jsp` and compare it with the Sesame front page in Figure 3-12 (yours should be similar with just different server name)
    - You have just a bunch of generic, empty repositories at the moment - we'll try to fix this in subsequence sections
- You can shutdown the server by simply killing the running java process.

Figure 3-12: The front page of a generic Sesame installation.

3.3.3. Setting up MySQL back-end database

We assume you have an access to a MySQL database server. Sesame is also able to work with PostgreSQL if you like. Please, set up (or ask the database administrator to) a new, empty data base called e.g. **sesame** and grant remote access rights for you (you may, if you like, set up a dedicated MySQL user for Sesame).

Now edit **`jetty-x.y.z/webapps/sesame/WEB-INF/system.conf`** and add the following element as a subpart of the <**repositorylist**> section:

```
<repository id="dos-rdfs-db">
  <title>DOS RDFS DB</title>
  <sailstack>
    <sail
    class="org.openrdf.sesame.sailimpl.sync.SyncRdfSchemaReposito-
ry"/>
    <sail
    class="org.openrdf.sesame.sailimpl.rdbms.RdfSchemaRepository">
      <param name="jdbcDriver" value="com.mysql.jdbc.Driver"/>
      <param name="jdbcUrl"
        value="jdbc:mysql://your.mysql.server:3306/sesame">
      <param name="user" value="mysql-user-name"/>
      <param name="password" value="mysql-user-password"/>
    </sail>
  </sailstack>
  <acl worldReadable="true" worldWritable="false">
    <user login="testuser" readAccess="true" writeAccess="true"/>
  </acl>
</repository>
```

One final thing you need is the MySQL connector:

- Decompress **`mysql-connector-java-5.0.7.zip`** file
- Copy **`mysql-connector-java-5.0.7/mysql-connector-java-5.0.7-bin.jar`** to **`webapps/sesame/WEB-INF/lib`** directory

Now just shutdown and start the Jetty server again. This time you should be able to find the **DOS RDFS DB** repository in the drop-down list on the main screen - choose it and you are there: in the middle of the store, able to read and explore it. In order to write anything, you just use the default testuser account (see the **`webapps/sesame/WEB-INF/system.conf`** file for password).

In order to use another Sesame user for this:

- Edit the <**acl**> part of the repository section you've just added in **`webapps/sesame/WEB-INF/system.conf`**
- Run `http://your.server.address:jettyport/sesame/config URL` in your browser and supply the Sesame administrator password (it's usually *admin* by default) to make the server reread its configuration dynamically.

3.3.4. Loading ontology models

Having a fresh repository set up and write access rights, you are ready for the final step of loading models. For the purpose of this tutorial we'll use the ViroLab domain ontology models:

- http://virolab.cyfronet.pl/onto/
    - we will focus on the most important **`vlom-upper.owl`** and **`vlom-data.owl`** models

Open your Sesame web interface and from there:

- Load our brand-new **DOS RDFS DB** repository
- Login to get the write rights
- In the upper *Modify actions* bar click the **Add (www)** link
- Paste in `http://virolab.cyfronet.pl/onto/vlom-upper.owl` URL in the first form and press the **Add data** button below
- Repeat the same action for `http://virolab.cyfronet.pl/onto/vlom-data.owl` file

In order to test if the loading procedure was successful:

- In the upper *Read actions* bar follow the **Explore** link
- Type in **Patient** in the second row (labeled *Or do a substring search:*) and press the **Find string** button
    - A small result table should appear with a single `http://www.virolab.org/onto/data/Patient` resource there (see Figure 3-13)

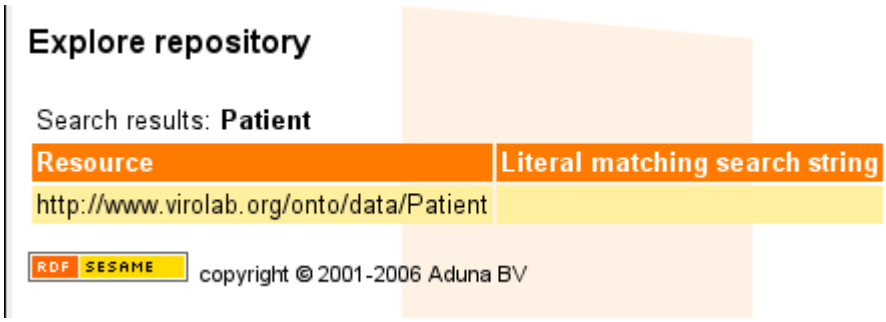

Figure 3-13: The picture of the proper testing search result.

And this is it. A new instance of the Domain Ontology Store is operational and accessible.

3.4. EXPERIMENT MANAGEMENT INTERFACE INSTALLATION MANUAL

Below is a list of instructions needed to run the EMI portlets.

1. Download, install and set the home directory of the Grid Space Engine.

The latest release of the Grid Space engine can be obtained from the GSEngine download site:

- http://virolab.cyfronet.pl/trac/vlruntime

After unpacking the distribution, set the environmental variable `GS_HOME` so it points to the GSEngine home directory.

2. Download, install and set the home directory of the JRuby interpreter.

The preferred version of the JRuby distribution is 1.0 and it can be downloaded from the JRuby download site (http://www.jruby.org/). After unpacking the distribution, set the environmental variable `JRUBY_HOME` so it points to the JRuby home directory.

3. Download, install and set the home directory of the Tomcat server.

The preferred version of the Tomcat server is 5.5.x and it can be downloaded from the Tomcat download site (http://tomcat.apache.org/). After unpacking the distribution, set the environmental variable `CATALINA_HOME` so it points to the Tomcat home directory.

4. Download and install the Gridsphere portal.

The preferred version of the Gridsphere portal is 3.0.7 and it can be downloaded from the Gridsphere download site (http://www.gridsphere.org/gridsphere/gridsphere/guest/download/). After unpacking the distribution run:

- **`ant install`**

in the Gridsphere home directory. This will build and deploy the Gridsphere application onto the Tomcat server.

5. Check out the EMI portlets project from the subversion repository at https://virolab.cyfronet.pl/svn/emi:

- `svn co https://virolab.cyfronet.pl/svn/emi`

and install the project.

After checking out the project set the home directory of the Gridsphere in the **`build.properties`** file by changing the gridsphere.home property. It is also required to copy all the jar files shipped with the GSEngine distribution and stored in the java folder of the main GSEngine directory into the lib directory of EMI project. After that run:

- **`ant install`**

in the main directory of the EMI project directory and start the Tomcat server.

3.5. SOURCE CODE ACCESS, BUG REPORTING AND AUTHORS CONTACT INFORMATION

The entire source code of the GRR, DOS and EMI modules is accessible through the Subversion repository (the anonymouns read-only access is granted for everyone):

```
#> svn checkout https://gforge.cyfronet.pl/svn/grr
#> svn checkout https://gforge.cyfronet.pl/svn/dos
#> 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 Trac website. For GRR and DOS modules:

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

For the EMI module:

- 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.

Authors list and contact information: Marek Kasztelnik (GRR) [*m.kasztelnik@cyfronet.pl*], Tomasz Gubała (DOS) [*gubala@science.uva.nl*], Daniel Harężlak (EMI) [*d.harezlak@cyfronet.pl*].

4. PROTOS DEVELOPERS' MANUALS

4.1. SEMANTIC EVENT AGGREGATOR CONFIGURATION MANUAL

Semantic Event Aggregator is responsible for building ontological information from xml data. The component is configured by *extension ontology* containing information of how to aggregate monitoring events and how to transform xml data into ontology individuals. This information is provided in a declarative convention. In such approach, in order to configure the aggregator, there should be provided declarations referring to:

- the conditions when aggregation should be triggered
- the derivation of instantiated individuals

The aggregator constantly monitors the gathered xml documents regarding the aggregation rules. Whenever a new aggregation is triggered, aggregator is informed of what classes should be instantiated and searches aggregated xml data for properties derivations.

The domain extension ontology *http://www.virolab.org/onto/extension* defines following concepts:

- *Derivation* concept describing how to establish property value
- *derivedFrom* annotation property that associates *Derivation* with class or property

There exist two concepts derived from *Derivation*, namely:

- *XMLDerivation* refers to properties directly accessible in xml data
- *DelegateDerivation* refers to properties being a kind of transformation applied to xml data

The domain ontology defines also:

- *AggregationRule* concept describing the conditions when aggregation should be triggered and what actions should be undertaken

4.1.1. XML derivation

To indicate value localization in xml document, *element* property should be defined - a path, with elements separated by '/' character, pointing to xml element or xml attribute.

The *XMLDerivation* concept has different meaning depending on the type of associated object. If associated with ontological class, it defines individual identifier value (useful when individuals are to be named in a particular convention), as presented in following example:

```
<ext-ns:XMLDerivation rdf:ID=”ExperimentDerivation”>
  <ext-ns:element
  rdf:datatype=”http://www.w3.org/2001/XMLSchema#string”>
    experimentStarted/ACID/experiment/id
  </ext-ns:element>
</ext-ns:XMLDerivation>
<rdf:Description
rdf:about=”http://www.virolab.org/onto/exp-protos#Experiment”>
    <ext-ns:derivedFrom rdf:resource=”#ExperimentDerivation”/>
<rdf:Description>
```

All instantiated *Experiment* individuals have identifiers identical as ACID experiment they refer to.

If associated with ontological property, the concept indicates the property value. In following example – data type property *beginning* is extracted directly from *experimentStarted/time* tag:

```
<ext-ns:XMLDerivation rdf:ID=”BeginningDerivation”>
  <ext-ns:element
  datatype=”http://www.w3.org/2001/XMLSchema#string”>
    experimentStarted/time
  </ext-ns:element>
</ext-ns:XMLDerivation>
<rdf:Description
rdf:about=”http://www.virolab.org/onto/exp-protos#beginning”>
  <ext-ns:derivedFrom rdf:resource=”#BeginningDerivation”/>
<rdf:Description>
```

4.1.2. Object properties

Most of the class derivation individuals should be associated with property *rootElement*. Like in *element* property, its meaning depends on the object type that the individual is related with. If associated with an ontological class, it indicates the top-level xml element corresponding with this class (in most cases, the parent of all elements containing this class properties). Such approach addresses the situation when single xml document refers to many classes so there is a need to localize documents fragments referring to given concepts. The root element is used to establish how many instances should be created (note, that if no *rootElement* is specified, there is an assumption that the root element is the parent of all aggregated xml documents, therefore a single instance is created be default).

If associated with an object property, *rootElement* indicates the root element of individual to be bound by this property. The next example refers to two aggregated xml document presented below:

```
<gridOperationInvoking>
  <inputParameter> … </inputParameter>
  <inputParameter> … </inputParameter>
</gridOperationInvoking>
<gridOperationInvoked>
  <outputParameter> … </outputParameter>
</gridOperationInvoked>
```

It should be indicated that *GridOperationParameter* concept is related with two xml elements:

```
<ext-ns:XMLDerivation rdf:ID="ParameterDerivation">
  <ext-ns:rootElement
  datatype=”http://www.w3.org/2001/XMLSchema#string”>
    gridOperationInvoking/inputParameter
  </ext-ns:rootElement>
  <ext-ns:rootElement
  datatype=”http://www.w3.org/2001/XMLSchema#string”>
    gridOperationInvoked/outputParameter
  </ext-ns:rootElement>
</ext-ns:XMLDerivation>
```

The derivation of *GridOperationInvocation* individual properties *inputParameter* and *outputParameter* (which have the same range, namely *GridOperationParameter*) should indicate that the individual localization in xml file depends on xml path:

```
<ext-ns:XMLDerivation rdf:ID="InputParameterDerivation">
  <ext-ns:rootElement
  datatype=”http://www.w3.org/2001/XMLSchema#string”>
    gridOperationInvoking/inputParameter
  </ext-ns:rootElement>
</ext-ns:XMLDerivation>
<ext-ns:XMLDerivation rdf:ID="OutputParameterDerivation">
  <ext-ns:rootElement
  datatype=”http://www.w3.org/2001/XMLSchema#string”>
    gridOperationInvoked/outputParameter
```

```
  </ext-ns:rootElement>
</ext-ns:XMLDerivation>
```

In presented configuration, exactly three individuals of class *GridOperationParameter* are created. Also, when instantiating *GridOperationInvocation* class, one value of *inputParameter* and two values of *outputParameter* property are associated, extracted from the xml elements in arbitrary order.

4.1.3. Delegate derivation

In more complicated cases, a property cannot be directly associated with an xml element, but is a form of transformation to be applied to xml primitives. In such situation, an ontology extension author is expected to develop delegate java classes providing desirable operations. The execution of operation is described semantically by *DelegateDerivation* concept. This concept is related with following properties examined by aggregator engine:

- *delegateClass* utilized class qualified name
- *delegateMethod* utilized method name
- *delegateParameter* method parameter

Method parameter concept is associated with following properties:

- *parameterElement* the xml element where the parameter value is placed
- *parameterOrder* the parameter order number in method signature
- *parameterType* qualified parameter Java type name, necessary to identify the method due to methods overloading issue

In the next example, the created ontology contains *experimentDuration* property while xml events contain only beginning time and end time data. The ontology author develops simple delegate class (note, that all non-primitive return types are converted to String type):

```
public class AggregationHelper {
     public static Long computeDuration(Long beginning, Long end)
{
              return end – beginning;
     }
}
```

The invocation of presented method is semantically described below:

```
<ext-ns:DelegateParameter rdf:ID="BeginningParameter">
  <ext-ns:parameterOrder
```

```
  rdf:datatype=”http://www.w3.org/2001/XMLSchema#int”>
    1
  </ext-ns:parameterOrder>
  <ext-ns:parameterType
  datatype=”http://www.w3.org/2001/XMLSchema#string”>
    java.lang.Long
  </ext-ns:parameterType>
  <ext-ns:parameterElement
  datatype=”http://www.w3.org/2001/XMLSchema#string”>
    experimentStarted/time
  </ext-ns:parameterElement>
</ext-ns:DelegateParameter>
<ext-ns:DelegateParameter rdf:ID="EndParameter">
  <ext-ns:parameterOrder
  rdf:datatype=”http://www.w3.org/2001/XMLSchema#int”>
    2
  </ext-ns:parameterOrder>
  <ext-ns:parameterType
  datatype=”http://www.w3.org/2001/XMLSchema#string”>
    java.lang.Long
  <ext-ns:parameterType>
  <ext-ns:parameterElement
  datatype=”http://www.w3.org/2001/XMLSchema#string”>
    experimentFinished/time
  </ext-ns:parameterElement>
</ext-ns:DelegateParameter>
<ext-ns:DelegateDerivation rdf:ID="DurationDerivation">
  <ext-ns:delegateClass
  datatype=”http://www.w3.org/2001/XMLSchema#string”>
    pl.cyfronet.virolab.mring.aggregator.AggregationHelper
  </ext-ns:delegateClass>
  <ext-ns:delegateMethod
  datatype=”http://www.w3.org/2001/XMLSchema#string”>
    computeDuration
  </ext-ns:delegateMethod>
  <ext-ns:delegateParameter>
    #BeginningParameter
  </ext-ns:delegateParameter>
```

```
  <ext-ns:delegateParameter>
    #EndParameter
  </ext-ns:delegateParameter>
</ext-ns:DelegateDerivation>
```

4.1.4. Aggregation rules

The aggregation rule concept is related with following properties:

- *eventType* events that should be aggregated (aggregation rule is satisfied if all events types are gathered)
- *instantiatedClass* the ontological class that should be instantiated
- *acidCoherency* the ACID coherency level (if set to 1, only experiment identifier is checked, if set to 2, also task identifier is checked, etc.), the aggregation rule is applied only to the xml events with corresponding ACID

In following example, the aggregation of events *gridOperationInvoking*, *gridOperationInvoked* (referring to the same invocation) results in *GridOperationInvocation* and *GridOperationParameter* classes instantiations:

```
<ext-ns:AggregationRule rdf:ID="GridOperationInvocationAggrega-
tion">
  <ext-ns:eventType
  datatype="http://www.w3.org/2001/XMLSchema#string">
    gridOperationInvoking
  </ext-ns:eventType>
  <ext-ns:eventType
  datatype="http://www.w3.org/2001/XMLSchema#string">
    gridOperationInvoked
  </ext-ns:eventType>
  <ext-ns:instantiatedClass
  datatype="http://www.w3.org/2001/XMLSchema#string">
    http://www.virolab.org/onto/exp-protos#GridOperationInvocation
  </ext-ns:instantiatedClass>
  <ext-ns:instantiatedClass
  datatype="http://www.w3.org/2001/XMLSchema#string">
    http://www.virolab.org/onto/exp-protos#GridOperationParameter
  </ext-ns:instantiatedClass>
  <ext-ns:ACIDCoherency
  rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
    2
```

```
  </ext-ns:ACIDCoherency>
</ext-ns:AggregationRule>
```

4.2. AUTHORS CONTACT INFORMATION

Authors list: Jakub Wach, Michał Pelczar, Bartosz Baliś.

Developer team contact: Jakub Wach [*wach.kuba@gmail.com*].

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

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

[DPIR] *Example design Patterns in Ruby* on http://www.rubygarden.org/

[GRAPPO-API] GridSpace Application Optimizer API documents, http://virolab.cyfronet.pl/~asia/grappo/apidocs/

[GT4] Globus Toolkit 4.0, http://globus.org

[HIBERNATE] Hibernate Relational Persistence for Java and .NET, www.hibernate.org

[JAVA] Sun Corporation, Java Programming Language, http://java.sun.com

[JETTY] Jetty Web Server, http://jetty.mortbay.com/

[JRUBY] JRuby – Java powered Ruby implementation, http://jruby.codehaus.org

[LOG4J] Apache logging library http://logging.apache.org/log4j/docs/index.html

[OPT-THESIS] Joanna Kocot, Iwona Ryszka: *Optimization of Grid Application Execution* Master of Science Thesis supervised by Marian Bubak; AGH University of Science and Technology, June 2007, Krakow, Poland, http://virolab.cyfronet.pl/~asia/msc/MScThesis_OptGridAppExecution.pdf

[PR] Dave Thomas, Chad Fowler and Andy Hunt: *Programming Ruby - The Pragmatic Programmer's Guide, Second Edition*, The Pragmatic Programmers, 2004

[RUBY] Ruby language, http://www.ruby-lang.org

[START_EXT_GRR] Start extending Grid Resources Registry tutorial http://virolab.cyfronet.pl/trac/vlruntime/wiki/StartExtendingGRR

[SVN] Subversion, version control system, http://subversion.tigris.org/

[SVNKIT] SVN Kit, pure Java Subversion implementation http://svnkit.com/

[TOMCAT] Apache Tomcat http://tomcat.apache.org

[WTS] Pieter Libin, Bart De Deckere, Joris Van Santvoort: *Wts: a stateful web service infrastructure*, http://wts.sf.net/