Akademia Górniczo – Hutnicza
im. Stanisława Staszica
w Krakowie
Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki

Katedra Informatyki

Paweł Charkowski

Środowisko do zarzadzania eksperymentami na gridzie

Praca magisterska

Kierunek: Informatyka
Specjalność: Systemy rozproszone i sieci komputerowe

Promotor:
dr inż. Marian Bubak

Konsultacja:
dr inż. Maciej Malawski

Nr albumu: 120519

Kraków 2009

Oświadczenie autora

Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie i że nie korzystałem ze źródeł innych niż wymienione w pracy.

Paweł Charkowski

AGH University of Science and Technology
in Kraków

Faculty of Electrical Engineering, Automatics, Computer Science and Electronics

Institute of Computer Science

Paweł Charkowski

Environment for Management of Experiments on the Grid

Thesis

Major: Computer Science
Specialization: Distributed Systems and Computer Networks

Supervisor:
dr Marian Bubak

Consultancy:
dr Maciej Malawski

Album id: 120519

Kraków 2009

Oświadczenie autora

Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie i że nie korzystałem ze źródeł innych niż wymienione w pracy.

Paweł Charkowski

Abstract

The subject of this thesis is the development of an environment for management of experiments on the grid. Nowadays scientific experiments executed on the Grid get more and more complicated. This means that scientists must spend more and more time to manually schedule tasks to the Grid, collect results, etc. The purpose of the management environment is to automate those processes, allowing scientists to concentrate on their research rather than managing experiments execution.

This thesis presents EMGE - the Environment for Management of Grid Experiments, a Java implemented environment designed for requirements of the ViroLab virtual laboratory. It shows complete development process of the Environment for Management of Grid Experiments: design and implementation of the database for storing experiment related information, project integration with the GSEngine, implementation of a user web interface allowing experiment monitoring and submission of new experiments to execution. It also contains tests of developed environment including unit and integration tests.

This thesis contents is organized as follows: Chapter 1 introduces subject of this work, presenting motivation and objectives to be achieved. Chapter 2 contains detailed problem analysis. It also gives a short description of existing experiment management environments available for the Grid and cluster computing, such as DIANE, Nimrod/G, ZENTURIO and Askalon environment. Detailed requirements for the EMGE system are presented in Chapter 3. Third chapter also contains core concepts of the designed environment. Chapter 4 presents EMGE's architecture details, designed database model and identified use cases. Chapter 5 presents EMGE's implementation details: UML models and sequence diagrams of most important actions performed by EMGE's components. Chapter 6 contains description of unit and integration test cases, together with conclusions gathered during test experiment of protein folding execution. Last chapter contains conclusion of this thesis, with a list of expanse proposals for future project development.

Key words

grid, experiment, management environment, ViroLab, virtual laboratory, user interface, job scheduling

Acknowledgements

First of all, I would like to express my gratitude to my supervisor, dr Marian Bubak, for guidance, patience and invaluable advices. I would like to sincerely thank dr Maciej Malawski for his support in design of the system and implementation counsels.

Contents

List of Figures

Chapter 1

Introduction

The subject of this thesis is the Environment for Management of Grid Experiments (EMGE) - a tool that will help to automate experiment running, scheduling, collecting experiment results and post-processing them. First section presents a short motivation for the management application which is the subject of thesis. In second section objectives of this work are presented. Finally, the organization of this document is presented.

1.1. Motivation for the Environment for Management of Grid Experiments

Fully functional, user friendly experiment management environment is a relevant part of each modern virtual laboratory, allowing users with no (or very basic level of) programmatic skills to fully explore the virtual laboratory potential in their scientific work: schedule experiments, monitor their status, etc. - in general one can say: manage experiments.

Currently existing job scheduling mechanisms in the ViroLab virtual laboratory [30] (both the command line GSEngine client or the web portal EMI - Experiment Management Interface) allow only to schedule a single job in one operation. In the case of parameter study experiments or batch experiments composed of several tasks that must be executed in specified sequence, the experimentator has to launch the experiment as one task that will have long execution time (looping through all combination of parameters in the case of parameter study experiment) or manually manage task execution - pass results between tasks, monitor their status. In order to better support such types of

experiments, a dedicated experiment management environment for the ViroLab needs to be created

1.2. Objectives

Main goals of this thesis are to provide analysis, design and implementation of a management application responsible for execution of long-running experiments in the ViroLab virtual laboratory, that will allow experimentators to monitor such experiment state and automate tasks scheduling, results collecting. and passing these results between tasks.
To achieve this, goals presented bellow need to be accomplished.

- perform research of already existing execution environments; Point out weak and strong points of these solutions in respect of the ViroLab virtual laboratory,
- analysis of requirements that need to be filled by the management application; The analysis should consider all aspects of the target environment - the ViroLab virtual laboratory,
- design and implementation of task execution and scheduling manager for the ViroLab virtual laboratory environment,
- proving usefulness of created manager for the ViroLab virtual laboratory and analyzing its performance.

1.3. Document organization

This thesis is organized as follows: Chapter 2 contains detailed problem analysis. It also gives a short description of existing experiment management environments available for the Grid and cluster computing, such as DIANE, Nimrod/G, ZENTURIO and Askalon environment. Detailed requirements for the EMGE system are presented in Chapter 3. Third chapter also contains core concepts of the designed environment. Chapter 4 presents EMGE's architecture details, designed database model and identified use cases. Chapter 5 presents EMGE's implementation details: UML models and sequence diagrams of most important actions performed by EMGE's components. Chapter 6 contains description of unit and integration test cases, together with conclusions gathered during test experiment of protein folding execution. Last chapter contains conclusion of this thesis with a list of expanse proposals for future project development.

Chapter 2

Background for the Environment for Management of Grid Experiments

This chapter presents background for this thesis. First section provides introduction to grid computing and the ViroLab virtual laboratory environment. In second section a brief description of existing experiment management systems is presented, pointing out weak and strong points of these solutions.

2.1. ViroLab[1]

The ViroLab is EU-funded Research Project of the EU 6th Framework Programme for Research and Technological Development in the area of integrated biomedical information for better health. The main purpose of the ViroLab project is to create a virtual laboratory that will support researchers and medical doctors in the area of viral disease treatment.

2.1.1. Rationale for the ViroLab virtual laboratory

A 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 term **experiment** in this context means a so-called in-silico experiment - that is, *a process that combines data and computations in order to obtain new knowledge on the subject of an 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." In context of the ViroLab

[1]The ViroLab section with all its subsections is based on [16], [30] and [31]

virtual laboratory each experiment has to be represented by a script that defines that experiment's plan, written in Ruby scripting language, in order to be executed using appropriate interpreter. Such script often invokes various operations on grid objects. As experiment plan files are written using standard scripting language they often contain loops, conditional statements and other flow control constructs that are built-in into the scripting language.

2.1.2. ViroLab users groups

Among all the ViroLab virtual laboratory users following classes of users are defined: [9]

- **Experiment developer**
  This is a person with programming skills, that possesses a certain level of scientific knowledge (eg. virology - in case of the ViroLab) able to design and implement a script, that will allow researchers to obtain useful scientific information and analyze produced by this script results. Experiment developer does not have to fully comprehend results and experiment process provided that there is expert assistance available to evaluate his work.
- **Experiment user**
  Person who runs a previously prepared script in order to obtain valuable results. Experiment user can be the script author himself or only partially involved in experiments design process. Of course it is not necessary for the experiment user to take any part in scripts creation, thus he/she can just be experiments executor that wants to obtain valuable results that answer his/her questions. In the ViroLab virtual laboratory following experiment user subtypes can be distinguished:

  A **Scientists**
  The scientists typically use specifically-tailored experiments, devoted to important areas of their science. Answers obtained from an experiment run usually denote modifications and upgrades to improve the experimental process. Repeatable process of experiment adjustment requires a significant amount of researcher time and is an important phase of their research.

  B **Clinical virologists**
  They are people who require very specific experiment to be available for them each day. Eg. this can be a person who works at a hospital and needs the experiment to get information regarding their patients infection case.

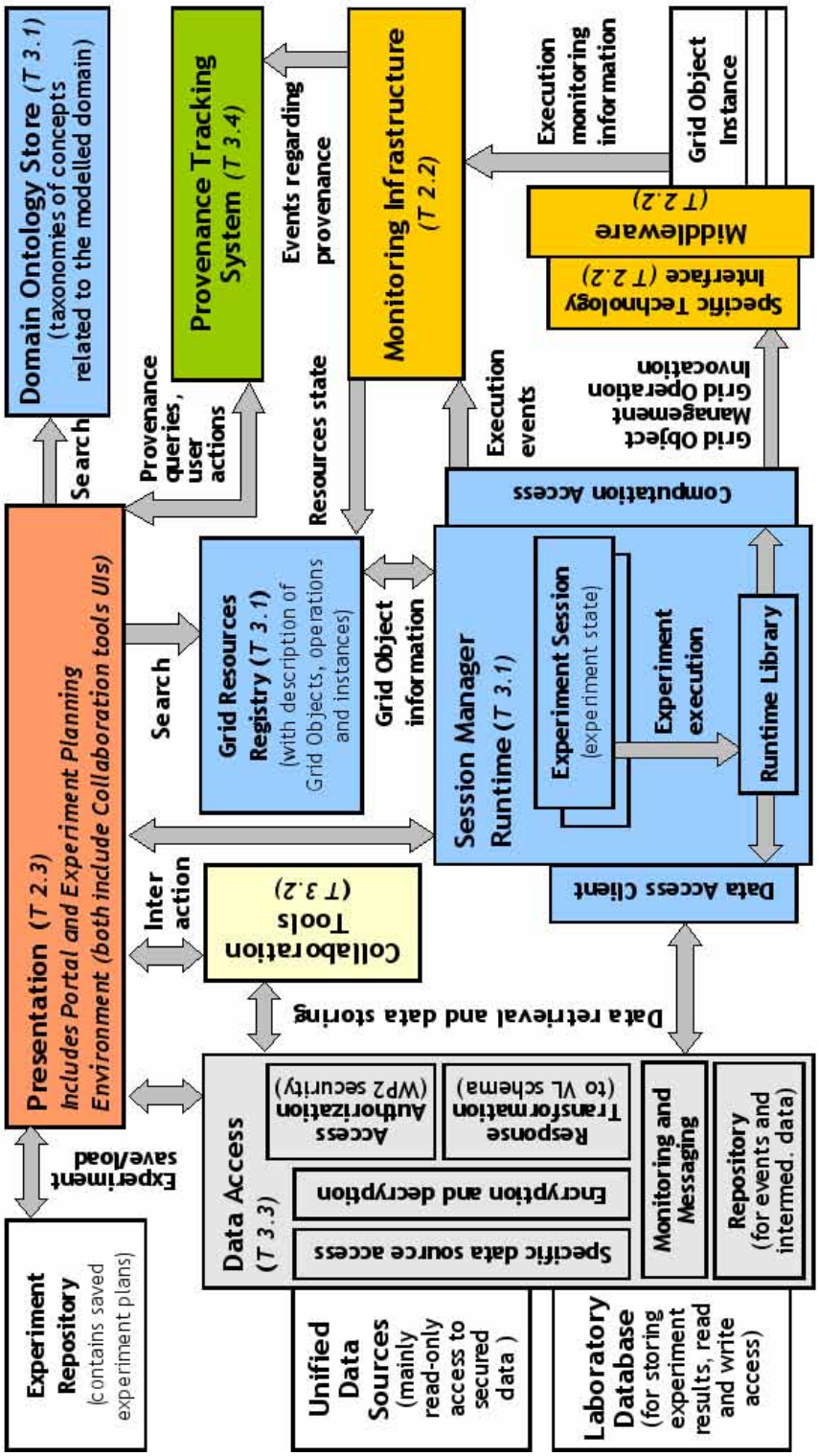

Figure 2.1. The diagram[2] presents architecture overview of the ViroLab virtual laboratory. It shows components, that are parts of the ViroLab, together with dependencies and relationship between these components.

[2]Figure 2.1 comes from the ViroLab virtual laboratory project site [30]

2.1.3. ViroLab architecture

Figure 2.1 presents an overview of the ViroLab virtual laboratory structure. It shows associations between Vlvl subparts together with main communication channels. The ViroLab architecture can be split into following layers (as shown in figure 2.2):

1. **Users**
   This layer represents users of virtual laboratory. More detailed user classes description has already been presented in section 2.1.2 of this thesis.
2. **Interfaces**
   Interfaces are tools used by the ViroLab virtual laboratory users to perform their tasks. It includes the Experiment Planning Environment (EPE) for experiment developers, and the Experiment Management Interface (EMI) for scientists and clinicians.[1]
3. **Runtime Components**
   Runtime tier serves as a bridge to both computational services and data sources available within the ViroLab virtual laboratory.
4. **Services**
   Services provide access to autonomous computation resources. Those can be Web Services, components, grid object, etc.. They also provide access to various types of data sources: relational databases and file systems in a unified way.
5. **Infrastructure**
   This tier represents physical resources on which experiment computation is performed. It ranges form personal computers, through computer networks up to advanced grid systems like EGEE or DEISA (projects sites respectively [22] and [19])

2.1.4. Experiment scheduling, execution and management in the ViroLab virtual laboratory

The first part of an experiment pipeline in the ViroLab virtual laboratory focuses on the design of an experiment process and is called **experiment planning**. During that phase *decisions how the required data are provided and how the experiment is executed are made*. During the next phase, provided by user with required input data, the experiment is executed. Finally, in the last phase of experiment pipeline, results obtained during the experiment execution are processed and stored to allow users their further analysis.
Since this thesis does not focus on experiment planing, more detailed description of this phase (experiment planning) will be omitted in the subsequent sections of this thesis.

[1] for more detailed description of EPE and EMI please refer to the ViroLab virtual laboratory site [30]

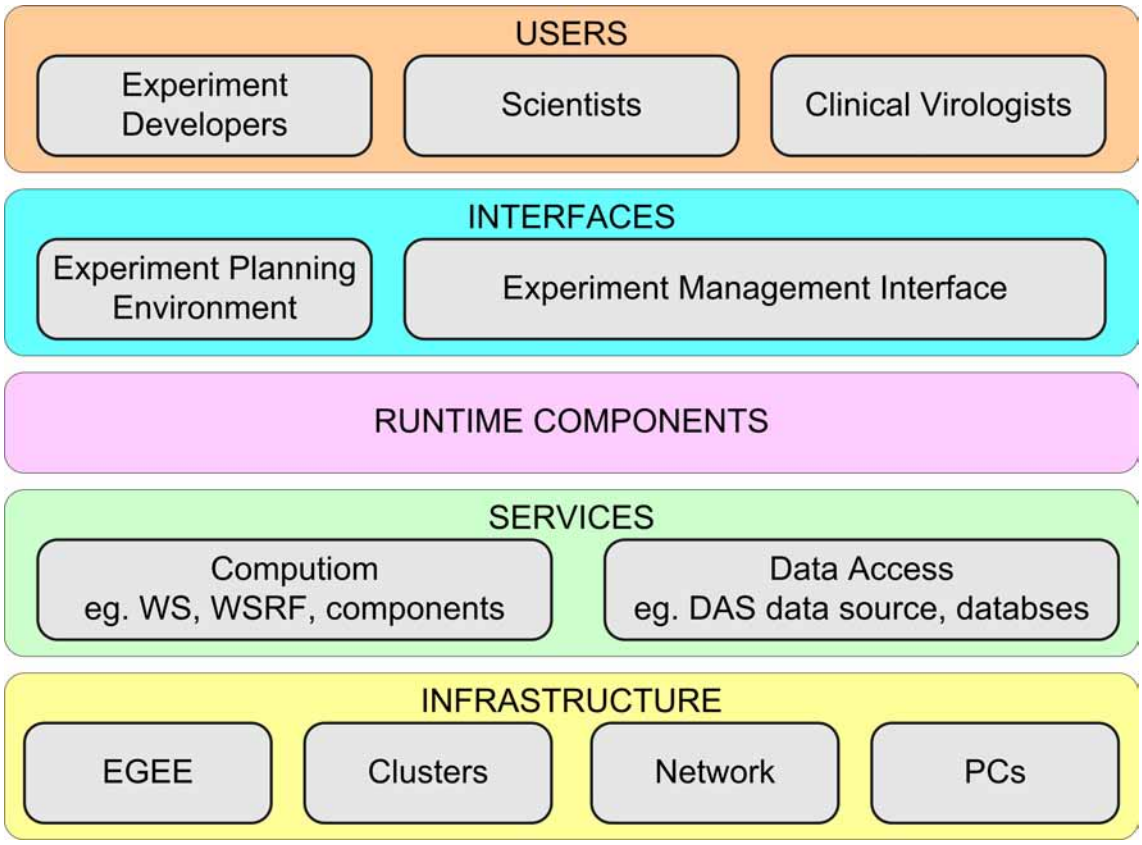

Figure 2.2. **Conceptual layers of the ViroLab virtual laboratory**. The "Users" tier represents all types of users of the ViroLab. "Interfaces" layer corresponds to the tools used by ViroLab users to perform their tasks. "Runtime components" serve as a bridge to computational resources and data stores distributed al over the ViroLab virtual laboratory. ViroLab "Services" provide access to autonomous computational resources accessible using the Internet network. "Infrastructure" tier corresponds to physical resources used in ViroLab for experiments computation.

The entity responsible for running experiments in the ViroLab virtual laboratory is called the GridSpace Engine (GSEngine). As mentioned earlier in this thesis, experiments in the ViroLab are represented by experiment plan scripts written in the Ruby language. In order to be able to execute experiment plans the GridSpace Engine has a built-in JRuby language interpreter, allowing it to interpret these scripts. Another important part of the GSEngine is a runtime library that provides all of the virtual laboratory specific functionalities that the interpreter standard library does not include. In order to execute ViroLab experiments user needs to provide a Ruby script containing experiment plan, either by specifying shared script location in the experiment repository or, in case of non-shared experiments, by providing script's file path on the local file system, and input data which will be used to perform the experiment.

The ViroLab virtual laboratory provides experiment users with following tools for

scheduling experiment execution: the GridSpace Engine client and the Experiment Management Interface.

GridSpace Engine client

The GridSpace Engine client (GSEngine [3]) is a command line utility that enables execution of experiment scripts in the ViroLab virtual laboratory. During each run of the GSEngine client the experiment script is interpreted only one time. Aside from providing information about location of the GSEngine server and arguments passed to the script, the user needs to provide previously generated security handle required for authentication process. The experiment script scheduled for execution may either be stored locally on users computer or accessed remotely form experiment repository.

Experiment Management Interface

The Experiment Management Interface (EMI) is a user web interface that supports experiments result browsing and experiment executing. It allows users to schedule execution of experiments, script code of which is available in the experiment repository. Unlike in the GSEngine client, the experimentator only needs to provide arguments for the script as GSEngine server connection parameters are provided by EMI. Although each user can have many experiments executed simultaneously, he/she needs to schedule execution of each of them separately.
Experiment Management Interface also grants user with a list of results, generated by all of his/hers experiments, stored using ResMan - Resource Management library. Each ResMan entry is shown as a separate result(even if an experiment generates more than one entry, they are shown as independent results) along with creation date and creation note. ResMan entries o not contain any information regarding the scripts that generated them, thus experiment user has to know it.

2.1.5. User authentication in the ViroLab virtual laboratory

Obviously, the access to the ViroLab and computing resources it provides is limited. Any person who wants to use those resources and gain access to the ViroLab virtual laboratory must successfully pass authentication procedure. In order to execute an experiment script using GSEngine client, the user needs to provide valid security credentials at the time execution is started. One of the ways to obtain such handle is to use the ShibIdpClient.

Shibboleth Identity Provider Client

Shibboleth Identity Provider Client (ShibIdpClient) is a tool that allows obtaining of Shibboleth security credentials using provided user login and password. The user may use command line version of ShibIdpClient distribution, called ShibIdpCliClient, that is a standalone console application that allows obtaining a valid security handle. ShibIdpClient is also available as Java archive to be used via API (Application Programming Interface).

2.2. GRID Computing

History of grid computing dates back to early 1990s. Increasing availability of the Internet combined with high-performance computing led to an idea of a virtual supercomputer for advanced science and engineering . Opposed to conventional mainframes, this concept assumed usage of geographically distributed hardware resources connected over public Internet network instead of vast number of processors connected by a local high-speed computer bus. It is important to remember that the 'grid' term denotes not only hardware infrastructure, but above all business logic together with scientific results that it delivers [6].

The first significant definition of the Grid was published in 1998 by I. Foster and C. Kesselman published [5] and it defined the **Grid** as *"a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities."*.

Since the Grid became highly popular not only in academic and scientific communities a great number of questions have been raised about when a system becomes a Grid. Answer to all those questions had been given by Ian Foster in 2002. In his article [4] Ian Foster presented main conditions that must be fulfilled in order to name system a Grid.

Primarily as Grid systems grant access and integrate geographically distributed computational resources that lie within different administration domains, those resources should be visible to end-user as unified resource. Managing issues like communication between resources, security, policy management, payment, membership lies within the Grid responsibility and is clearly specified by the sharing rules defined by resource providers and consumers.

Furthermore Grid environments should use open standards for communication, authentication and authorization. Open standards should also be applied for resource discovery, their access and coordination. Using open standards assure that the system is not application–specific , allowing easy integration and use of multiple resources.

Another basic requirement for Grid system is to provide so called **Quality of Service (QoS)** - a *set of boundaries that will be fulfilled by its resources to meet end-user requirements.* Quality of Service can specify for example response time, throughput, resource availability, security, failure recovery, etc., thus allowing very precise experiment planning.

Grid applications characteristics

Grid applications usually have a long execution time and are composed of a vast number of co-dependent or independent jobs. Such experiments often need some kind of checkpointing, not to mention persistent storing of their sub jobs results, that in case of a system failure the results of finished jobs will be accessible allowing their usage in job rescheduling without the need to reschedule already finished tasks.

Typical example of grid applications are **parameter-study** experiments - *experiments where same computation is executed large number of times using*

different application parameters in each execution, usually for each element of the cartesian product of parameter ranges. Because jobs, that compose parameter-study experiments are independent from each other they can be parallelly executed on different grid nodes, decreasing the overall experiment execution time in comparison to the time needed for their sequential execution.

Another important group of grid applications are the ones composed of vast, complex workflows. Apart from independent jobs that may be executed simultaneously the core of workflow applications are dependencies between tasks, where some of them need to be executed in a specified order as they depend on other jobs output.

2.3. Overview of Related Works

As mentioned earlier in this thesis Grid computing has became very popular last years. Experiments are becoming more and more complex, and the number of tasks that need to be calculated for a single experiment is growing fast. Due to this, "manual" management of such experiments is not much of pleasure for their designers.

Many Grid organizations have developed systems that allow experiment execution to become more automated. The following sections present a brief description and analysis of existing systems pointing out their advantages and disadvantages.

2.3.1. DIANE: Distributed Analysis Environment

DIANE: Distributed Analysis Environment (project site [20]) is a tool which helps application communities and smaller Virtual Organizations use the distributed computing infrastructures more efficiently. The DIANE framework provides automatic failure recovery, load balancing and job scheduling. It can be used for "mainframe" computing, as well as in heterogenous computing environments such as the Grid or interactive clusters. The DIANE project is the result of R&D in CERN IT Division focused on interfacing semiinteractive parallel applications with distributed Grid technology [8] [7], and is developed using Python programming language.

DIANE's architecture is based on a Master-Worker model. After the Master Process has been started it dynamically schedules jobs to available worker nodes called Worker Agents. Worker Agents are launched independently to the Master process. The Master Process keeps track of scheduled tasks. It allows him to react to job execution failures, spawn new tasks and provide synchronization. After a job is finished the Worker Agent sends results back to the Master process. A soon as all jobs are finished the Master Agent is terminated.

Communication between Master and Worker Agents is fully transparent to the user and is implemented with omniORB Object Resource Broker. Due to this, Diane users do not need to focus on providing communication between master and

workers, thus allowing them to fully focus on specifying jobs to be executed.

The DIANE framework provides users only with an ability to run large number of small independent tasks (like described earlier in this chapter parametric study experiments). To schedule execution of tasks the user needs to specify a so called 'run file', which is a file containing script written in Python, that will generate information about tasks that are to be executed.

2.3.2. Askalon - Grid Application Development and Computing Environment

Askalon (official project site [18]) is the most advanced of all presented in this thesis existing systems designed to improve on-grid experiment execution. ASKALON is a Grid environment for composition and execution of scientific workflow applications [11].

Askalon allows users to build a workflow using available components without need of any knowledge about the Grid complicity nor any implementation details of Grid services. Askalon provides Abstract Grid Workflow Language (AGWL) [15], an XML-based language which allows to specify task execution graph (it provides possibility to specify conditional constructs, conditional loops, parallel loops etc.). Abstract Grid Workflow Language also provides access mechanism to data repositories. The user can programmatically describe the experiment workflow using AGWL, or use a graphical editor for UML-based modeling to compose a workflow that will automatically be resolved into an AGWL file.

The scheduler embedded in Askalon has a two stage experiment scheduling mechanism. At first, the Workflow Converter module transforms given workflow described in AGWL into a direct acyclic graph. For conditions or parameters that are unknown at the moment of performing transformation, due to for example an if statement, the Workflow Converter assumes the most probable execution path in the workflow. After this transformation is complete, the Scheduling Engine comes into play, being responsible for proper scheduling of the newly evaluated workflow now presented as an acyclic graph. The workflow is executed until it is successfully finished or any interruption occurs during its execution. If wrongly made by the Workflow Converter assumptions are the cause of execution interruption, then workflow rescheduling is forced on the Workflow Converter (what may include making new execution predictions) and experiment is continued using newly created workflow [10].

UML-based workflow modeling approach to experiment scheduling and management introduced in the Askalon environment is not as much flexible as an approach based on invoking scripts execution. Users working with the Askalon environment need to possess at least basic programming knowledge. Moreover, using this system requires from them either learning the Abstract Grid Workflow Language, if they choose to write workflow description AGWL files by hand, or, if they decide to use workflow composition tool for Askalon, know how to model

applications using UML modelling language and possess programming skills since computations or operations invoking such computations (e.g web services) need to be programmed by the workflow creator.

2.3.3. ZENTURIO - Experiment Management System for Cluster and Grid Computing

Zenturio is an automatic experiment management system for cluster and Grid computing. It is purposely designed to specify and execute complex programs in the context of performance analysis and tuning, parameter studies, and software testing. [14]. Zenturio project's official site is [32].

Zenturio is designed as a distributed application, composed of several Grid services [13]:

- Registry service
  It contains a Registry Service (RS) which registers existing services and provides clients with information needed for locating those services. Each service registered in RS is granted a lease for a certain timeout period. If a service lease is not renewed before expiration RS deletes that service location information.
- Experiment Generator Service
  This module is responsible for generating a set of application instances, based on ZEN information provided in the application script, by computing the cartesian product of sets described in ZEN directives. ZEN [12] is a directive-based language that allows to specify parameter sets that shall be used to launch job instances and to specify arbitrarily complex program executions. ZEN directives are comment lines starting with a special ZEN prefix. They are ignored by systems that are unaware of their semantics, making ZEN non-dependent to a specific programming language.
- Experiment Executor Service
  Responsible for executing and managing experiments on the machine it operates on.

Client-side of Zenturio, that "provides the user with a GUI-based interface for submitting, monitoring, controlling, and analyzing experiments" [13] is composed of portlets: the Experiment Monitoring portlet that allows monitoring experiment status, and the Experiment Preparation portlet that allows preparing sets of experiments based on information provided by user.

Since all of the ZEN language directives are hidden in comments of executed application code any change in the workflow or parameters values require modification of executed experiment file. Due to requirement for Zenturio user to posses knowledge of ZEN programming, the user needs to learn completely new programming language in order to efficiently use functionality provided by the Zenturio environment.

2.3.4. The Nimrod Toolkit

Nimrod (official site [28]) is a project designed and developed on Monash University in Australia. It was designed as a tool for parametrized simulations. The most interesting from this thesis point of view is Nimrod/G - GRID aware version of Nimrod [1] [2].

Nimrod was designed for simulation experiments. Because this type of experiments is based on performing same operations on different sets of parameters, the main concept of Nimrod was to automate this process. All jobs in an experiment are described by a simple script file called a 'plan file'. This file contains ranges and steps of parameter values in the first section, followed by operations that need to be calculated on each set of parameters from the cross product of their values. The user does not have to write the plan file manually, he/she can use the Nimrod Portal, a web based portal, to generate the plan file and schedule its execution. Although the file is generated automatically the user still needs to provide all information needed to generate this file: parameter ranges, executed operations etc., you can say that instead of writing the plan file the user has to 'click it out'.

All information needed to complete the experiment by calculating all jobs are stored in a persistent database. The database contains information about scheduled tasks, results, task execution state, experiment information etc. Nimrod makes sure that the database content is up to date. Due to storing actual state of experiment in a persistent database, it is possible to "continue" the experiment after a system crash (or any other failure) basing on the information stored in the Nimrod's database.

Since Nimrod/G is a version of Nimrod written to perform experiments on the Grid, it distinguishes different resource types by the module it uses to interact with grid middleware software. Currently Nimrod/G uses features supported in the Globus toolkit.

Nimrod jobs are user commands that invoke some executable and are passed to shell for execution as console commands. While specifying the experiment plan file, either by manually writing the plan file or by using the Nimrod web portal, the user needs to provide the shell command to be executed. Taking into account that the ViroLab experiments are executed using the GridSpace Engine, GSEngine client would have to be installed on each node that may be used by Nimrod for job execution, or the user would need to include copying of GSEngine executables within each experiment. Since security credentials are passed to the GSEngine client as execution arguments the user would have to provide it during experiment scheduling. Because of that problems with Shibboleth security credentials, used in the ViroLab virtual laboratory, validation timeout may occur for long executing experiments.

2.4. Conclusions

Each of previously described management environments have certain limitations that discard them from being used in the ViroLab virtual laboratory. For example, DIANE does not support execution of complex workflows and manages only scheduling of parameter study experiments. Since the ViroLab virtual laboratory users are mainly clinicians and virologists (in general non-informaticians) they do not possess any programing knowledge. Using one of presented experiment management solutions in the ViroLab would require them to posses such knowledge: the DIANE uses Python run files, using Zenturio requires user to know the ZEN language, and the Askalon users either need to learn the AGWL language or know how to model using UML. Not to mention implementation skills required for usage of those systems. The Nimrod Toolkit on the other hand requires GSEngine client to be installed on each of the nodes it schedules jobs to, as it executes jobs as console commands. Using Nimrod in the ViroLab would also result in problems with validation of security credentials required for scheduling tasks using GSEngine.

Zenturio's distributed architecture concept is an interesting approach to experiment management environment. Combining it with workflow composition model (similar to the one implemented in Askalon) would provide an efficient environment, capable of executing complex workflows. Instead of modeling workflows in UML, as it is in Askalon, the mechanism implemented in EMGE would allow workflow composition from scripts in user-friendly way, where each node of the specified workflow is defined by script and input data the script operates on.

In conclusion presented solutions are not suitable for usage in the ViroLab virtual laboratory. Nevertheless they have some interesting concepts, which may be used during the design of the Environment for Management of Grid Experiments for the ViroLab.

Chapter 3

Concept of the Environment for Management of Grid Experiments

This chapter introduces a concept of the Environment for Management of Grid Experiments. At first, analysis of the problem is presented. Second section introduces requirements for the developed grid experiments management environment, that is the subject of this thesis. Finally assumptions made for the developed environment are presented.

3.1. Problem analysis

As mentioned earlier in this thesis, experiments in the ViroLab virtual laboratory are represented by Ruby scripts. One of the consequences of script approach to experiments is that the experiment state during its execution is equivalent to the current state of the interpreter - it is not persistent. Shall any interpreter error occur during script interpretation, whole experiment computation needs to be manually rescheduled by user. As this is not a big problem for scheduling single experiments with short computation time it becomes a huge matter for complex experiments process - all time and resources spent for failed execution are lost. Dividing such experiments into a group of smaller tasks scheduled in a specified order would of course decrease resource waste as only failed tasks would need rescheduling, however currently the ViroLab virtual laboratory does not provide its users with tolls that would allow user friendly execution and management of such multitask experiments. Moreover manual scheduling and management of such experiments would be a huge challenge for the experimentator.

Therefore, an environment that will allow automatical task scheduling, for non-trivial experiments, and experiment management for the ViroLab users should

be developed in order to eliminate unnecessary resource usage and improve user comfort in using the ViroLab virtual laboratory.

3.2. System requirements

This section lists and describes both functional (section 3.2.1) and non-functional (section 3.2.2) requirements that designed experiment manager needs to fulfill.

3.2.1. Functional requirements

This section focuses on presenting functional requirement that the system needs to comply with. Designed system has to meet following functional requirements:

- Management of executing large number of experiments in virtual laboratory, taking care of all aspects connected with scheduling like for example managing security credential and task dependencies.
- Enable end-users to specify experiment execution workflow, allowing task-to-task dependencies and passing group of tasks output as input data for another.
- Provide user interface for management and monitoring of experiments and their current status.

3.2.2. Non-Functional requirements

This section focuses on presenting non-functional requirement that the system needs to comply with. System's non-functional requirements are as follows:

- User interface provided by the system needs to be intuitive and easy to use, especially for users with basic or without proper computer science educational background. Moreover information concerning experiments tasks current state should be presented in a user-friendly way.
- Both database space and grid resources used by the designed system should be minimized without impacting designed applications performance.
- Experiments current state should be stored persistently so that in case of any system failure minimal amount of tasks would need to be re-executed.
- System should use GSEngine to execute tasks.
- Number of simultaneously executed tasks for each user should be limited. Different users limits should be independent from each other. System should allow administrator to change these limits at any time without stopping the system.
- System should be easily configurable.

3.3. System concepts

This section presents assumptions made upon the designed Environment for Management of Grid Experiments.

- **Web-based user interface**
  Web-based user interface is the most convenient type of interface from users point of view. Only a web browser, which nowadays is available on every computer, is required to use such interface, and it does not require users to install any applications to be able to use EMGE's Web Portal, since all components of EMGE are available on the web server EMGE stands on. Moreover any upgrades made to the application are easy to perform, and are immediately available for all users without any upgrading needed on client side.
- **Database oriented architecture**
  Database used by the designed EMGE environment will be its central point and main module, storing all information regarding the users allowed to use the system together with all information concerning experiments management: experiment structure, job submission information, executed tasks statuses, input data for tasks, task execution logs and naturally results obtained during job execution, allowing fast access to stored information. Information stored in the database should be as accurate as possible and always up to date. In addition storing experiment information in the database will provide the highest error immunity in case of scheduler failure, minimizing resource usage as only tasks requiring rescheduling will be executed again.
- **Independent modules**
  Designed system should be composed of the following modules: user portal, responsible for providing user interface that allows user interaction with the system (schedule new experiments, monitor experiments status, etc.), and experiment manager responsible for experiments execution and storing experiment information into database. Both user portal module and scheduling manager module should be fully independent from each other and exist without any "knowledge" about each other.

3.4. Summary

High level abstraction of environment for the Environment for Management of Grid Experiments have been introduced in this chapter, including the ViroLab experiment management problem analysis followed by solution proposal for identified problems. Functional requirements, identified during problem analysis, for designed system have been listed with their brief description along with non-functional environment requirements. Finally, main concepts for the Environment for Management of Grid Experiments have been presented: usage of Web user interface, independent environment modules concept and database centered architecture.

Chapter 4

Design of the Environment for Management of Grid Experiments

This chapter presents an overview of the Environment for Management of Grid Experiments architecture. First section presents Environment for Management of Grid Experiments architecture with brief description of its components, together with dependencies on external ViroLab components used by EMGE. Second section provides use case diagrams of identified scenarios, with these scenarios brief description. Database model is introduced, together with its motivation, in the last section of this chapter.

4.1. EMGE's architecture overview

Diagram in Figure 4.1 summarizes the Environment for Management of Grid Experiments system architecture, showing correlations between EMGE's components and also their dependencies on external components. EMGE components description is presented in section 4.1.1. Detailed description of the database structure is presented in section 4.3.

4.1.1. Component description

- **User Portal** - provides a web-based interface allowing interaction between users and the EMGE system. It is composed of following components:
  - **Experiment Monitor** - displays experiments current status, providing detailed information about tasks execution: current task status, task input, execution log, results, etc.
  - **Experiment Creator** - allows creation and scheduling of new experiments.

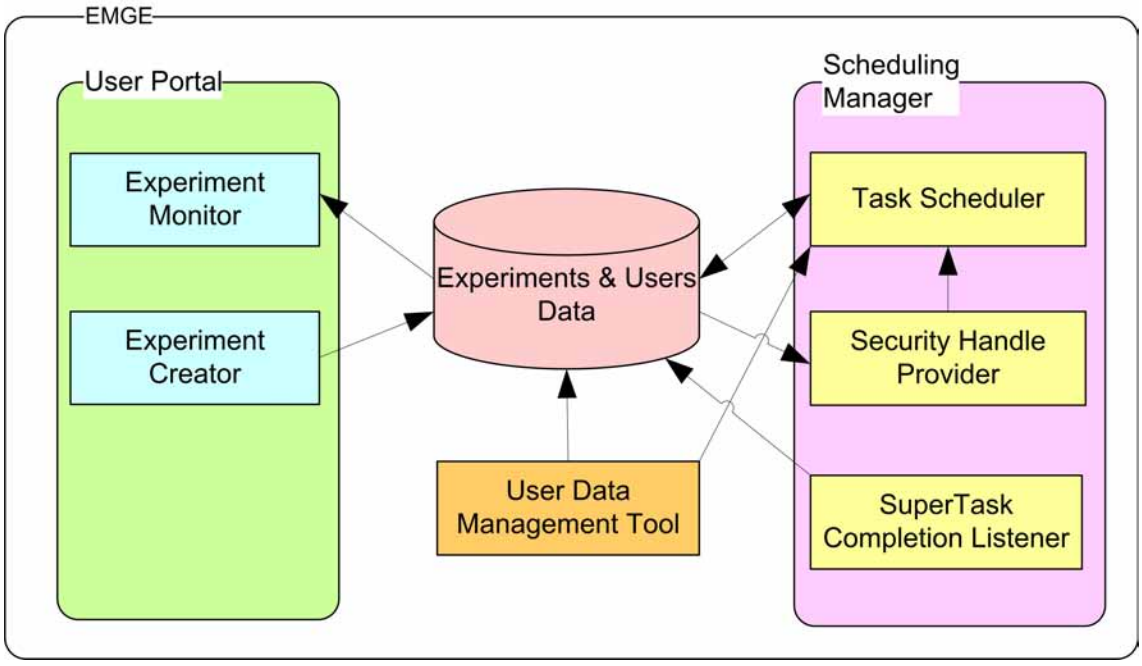

Figure 4.1. **The Environment for Management of Grid Experiments architecture diagram.**
This diagram shows main components of the EMGE system: the User Portal, the Scheduling Manger, the User Data Management Tools and the central point of the EMGE system: database storing all information required by the system to work. This diagram also presents relations between the components of EMGE environment.

- **Scheduling Manager** - entity responsible for managing experiments on-grid execution.
  - **Task Scheduler** - main component of the designed system, responsible for scheduling tasks execution to Grid using the GridSpace Engine. It is also responsible for controlling the amount of concurrently executed tasks.
  - **Security Handle Provider** - this component is responsible for providing valid Shibboleth security handle's with minimizing number of new handle requests send to the Shibboleth Identity Provider.
  - **Task Completion Listener** - after receiving notification about completed task execution it is responsible for notifying the Task Scheduler about freed execution slot. To its responsibilities belongs also performing following actions on detection of super task completion[1]: generating super task results summary and preparing input for super tasks depending on the completed one.

[1]For information regarding experiment structure representation in EMGE system please refer to section 4.3

- **User Data Management Tool** - a tool for updating EMGE's users information in the database and the current tasks limit used by running instance of the Scheduling Manager component.

4.1.2. External ViroLab components used by EMGE

Figure 4.2 shows EMGE's modules usage of external libraries from the ViroLab virtual laboratory environment.

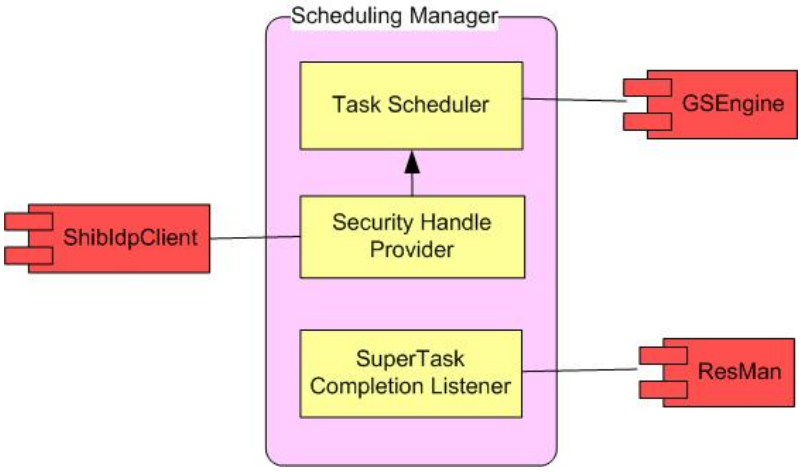

Figure 4.2. **ViroLab components dependencies of the Environment for Management of Grid Experiments diagram.**
This diagram presents dependency of EMGE components on external components provided by the ViroLab environment: the ShibIdpClient, the GSEngine and the ResMan library.

Both the GridSpace Engine client and the Shibboleth Identity Provider Client have been introduced in previous sections of this thesis. Please refer to section 2.1.4 for information about the GSEngine client, and to section 2.1.5 for introduction of the Shibboleth Identity Provider Client.

Result Management library

The Result Management library (ResMan) provides means of interaction with external stores responsible for keeping result-related information. The ResMan library is the only entity allowed to access result data stores, thus keeping it coherent and consistent. Each result entry is composed of two separately stored and accessed entries: the result payload and the result metadata identified by same result identifier.

4.2. Use cases identified for experiment management environment

Presented further in this section use case diagrams feature following actors:

- **Experiment User**
  Person who runs an experiment in order to obtain valuable results that answer important questions .
- **Scheduling Manager**
  Entity responsible for launching experiments' tasks, taking into account number of concurrent tasks that are allowed for each experiment user to be executed simultaneously.
- **Administrator**
  User responsible for deployment and configuration of the EMGE system. Administrator also decides about simultaneously launched tasks limit assigned for each experiment user.

4.2.1. User Portal use cases

Figure 4.3 presents User Portal's use case.

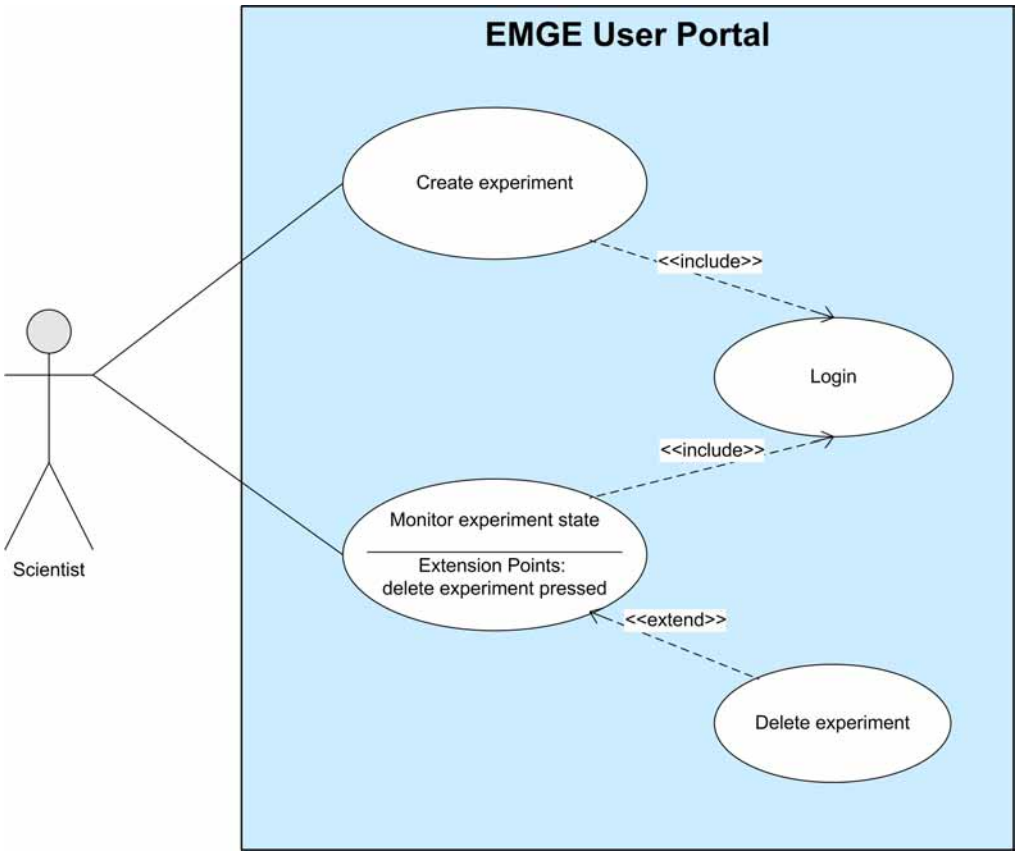

Figure 4.3. **Use case diagram of the User Portal component.**
Diagram presents use cases identified for the User Portal of the EMGE system. The actor performing operations on the User Portal is a user of the ViroLab virtual laboratory. He can schedule new experiments execution to the EMGE environment. The actor can also monitor current state of the experiments he had scheduled and cancel their execution.

Following use cases have been identified for the User Portal:

- **Create experiment**
  In this scenario the actor submits new experiment to EMGE, providing all information needed to define the experiment.
- **Monitor experiment state**
  In this scenario the actor is provided with information about his experiments: scripts used by experiments, current tasks status and data used as tasks input, execution start date (and if the task has finished its execution time), execution log, etc.
- **Delete experiment**
  In this scenario the user decides to cancel experiment execution and remove all information corresponding to it from the EMGE system.

4.2.2. Scheduling Manager use cases

Figure 4.4 presents Scheduling Manager's specific use case.

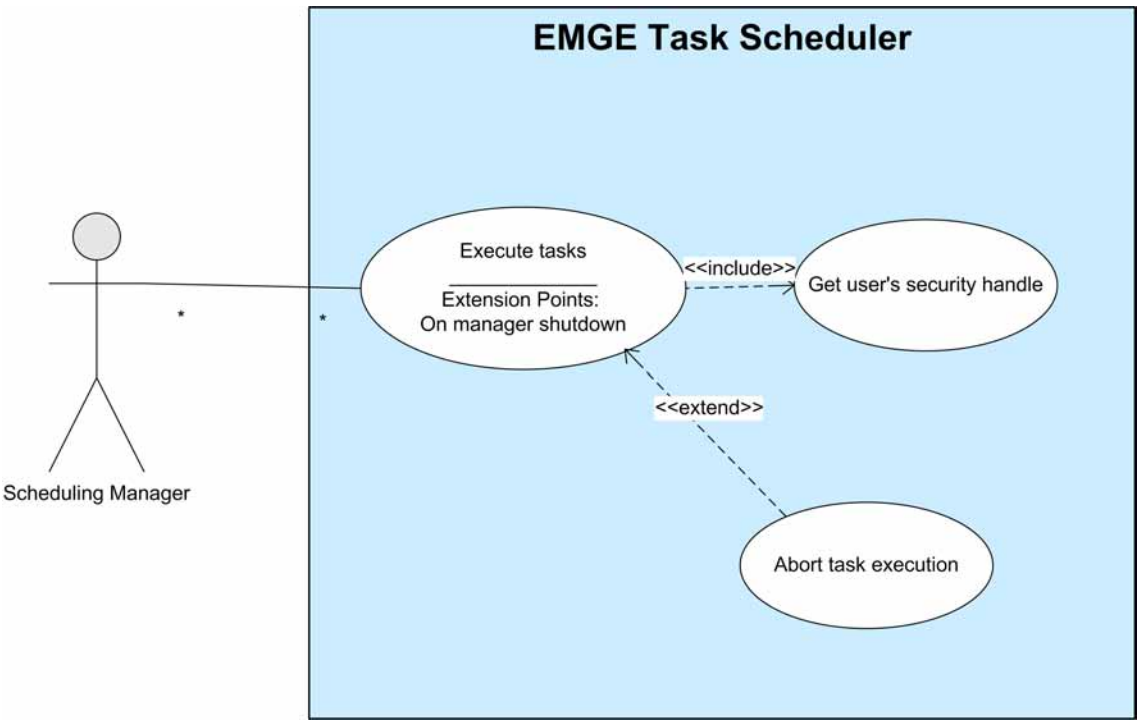

Figure 4.4. **Use cases identified for the Scheduling Manager**
Diagram presents use cases identified for the Scheduling Manager component of the EMGE system. The Scheduling Manager schedules tasks execution to the ViroLab virtual laboratory. Task scheduling process may require to obtain a new security handle for the user owning the task. Moreover the Scheduling Manager may abort tasks execution if the experiment has been aborted by its owner, or on Scheduling Manager exit to prevent unnecessary resource usage.

Following use cases have been identified for the Scheduling Manager:

- **Execute tasks**
  In this scenario the actor periodically schedules new tasks to the Grid using GSEngine. Task definitions are read from the database.
- **Abort tasks**
  In this scenario the actor aborts scheduled tasks execution on the EMGE environment shutdown. This operation is performed because of the callback nature of the GSEngine to release resources used by scheduled tasks and to prevent their unnecessary usage. This scenario also takes place if the experiment owner requests to abort execution of an experiment.
- **Get security handle**
  During this scenario actor requests generation of a new Shibboleth security credentials for the experiment owner.

4.2.3. User Data Management Tool use cases

Figure 4.5 presents use case diagram for User Data Management Tool

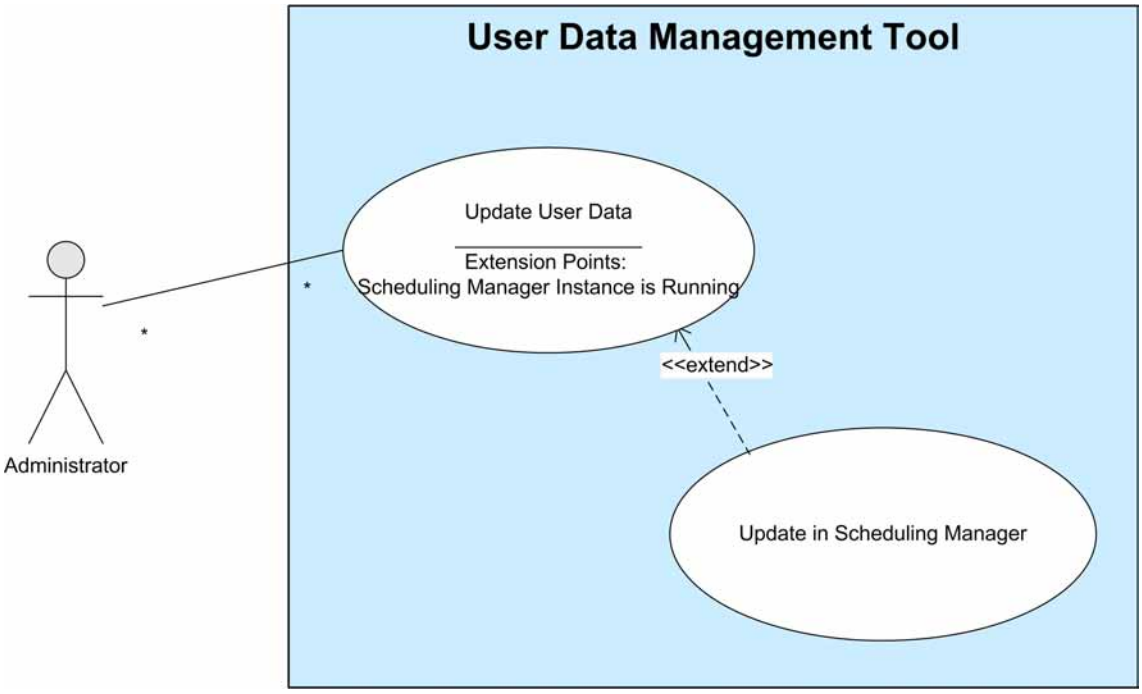

Figure 4.5. **Use cases identified for the User Data Management Tool**
Diagram presents use cases identified for the User Data Management Tool. EMGE system administrator may use the User Data Management Tool to update maximum number of tasks that may be during execution on the same time for each user. The update modifies these values in the database, and if an instance of the Scheduling Manager is launched it is notified about performed tasks limits changes.

Following use cases have been identified for the User Data Management Tool:

- **Update user data**
  In this scenario using the User Data Management Tool the administrator changes user password used by EMGE and/or concurrent task limit, which are both stored in the systems database. Update can be performed for several different users.
- **Update tasks limits in the Scheduling Manager**
  Concurrent tasks limits stored in memory by the Scheduling Manager instance are updated.

4.3. Database Model used by the Environment for Management of Grid Experiments

4.3.1. Rationale for the database model

The database designed for the Environment for Management of Grid Experiments needs to reflect the ViroLab virtual laboratory experiments in a way that will allow efficacious experiment management, and allow to provide users with sufficient information about current state of their experiments. Each ViroLab experiment is composed of tasks that are executed as Ruby scripts. In order to improve clearness and provide hierarchical experiment structure, a so called **Super Task** has been defined, and is nothing more than *a group of independent from each other tasks, defined by identical script, being part of the same experiment.* The table storing information on scripts locations is introduced to remove information redundancy and to decrease database size. User-defined execution dependencies, including dependancy relation type (e.g 1:n etc.), that occur in the experiment are defined between super tasks. Since all of the tasks require data to operate on, a table storing input data for tasks is required.

4.3.2. Table model details

Figure 4.6 presents database tables model used by the Environment for Management of Grid Experiments.

Detailed description of the tables and roles of each table in the database are as follows:

- **USER_DATA table**
  This table stores information regarding users that are allowed to use the EMGE system. It contains following fields
  — **user_name (PK)** - text entry representing user login required for obtaining Shibboleth security handles. It also represents user identifier used by the EMGE environment.

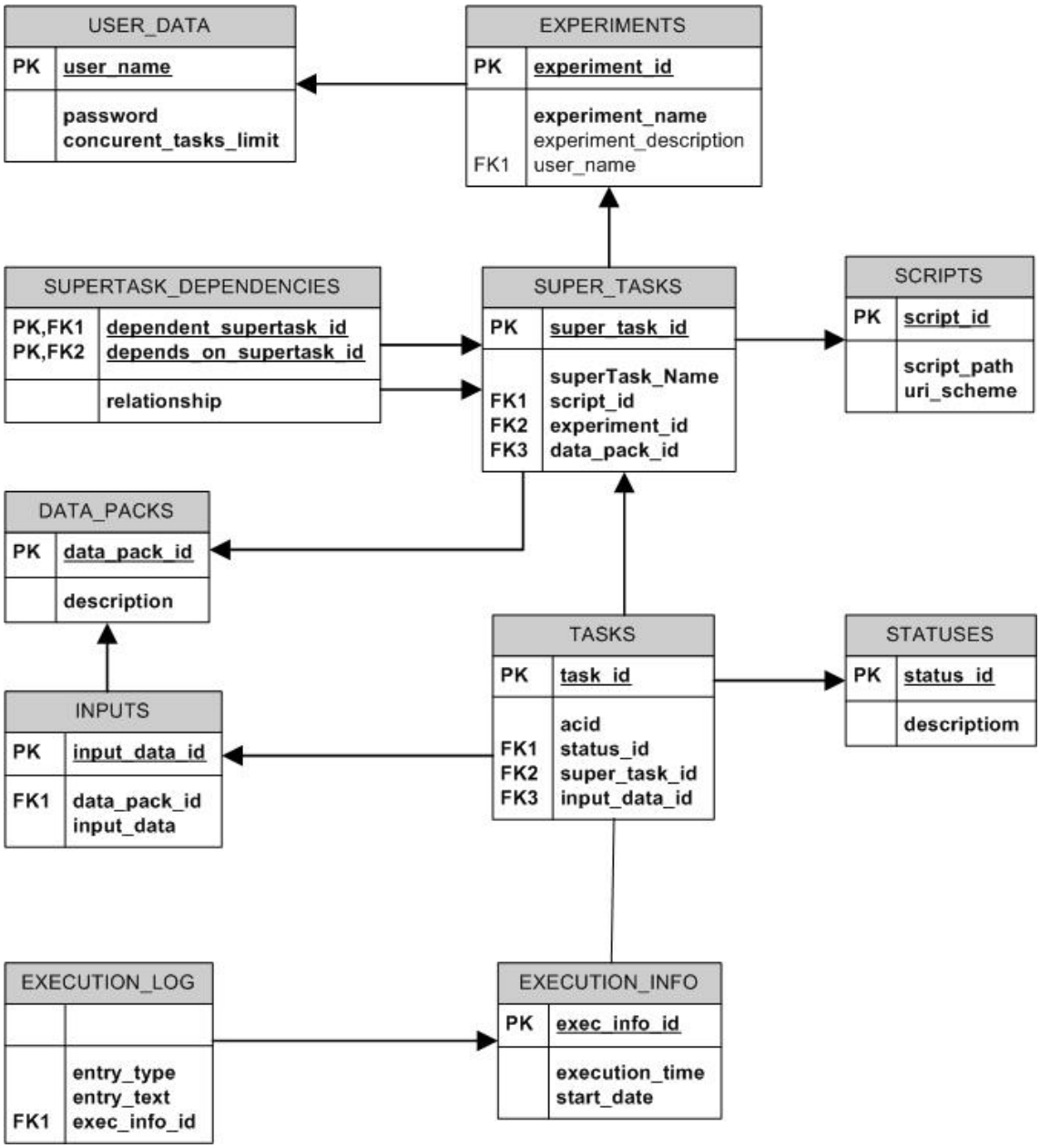

Figure 4.6. **Environment for Management of Grid Experiments database tables model.**

— **password** - text entry representing password required for obtaining Shibboleth security handles. Due to security reasons password is encrypted prior to persisting it into the database.

— **concurrent_tasks_limit** - integer determining the total number of user tasks that may be simultaneously executed on the Grid.

- **EXPERIMENTS table**
  This table stores information representing and identifying each experiment in the EMGE environment.
  — **experiment_id (PK)** - unique identifier of the experiment
  — **experiment_name** - text entry storing short name given to the experiment during experiment submission to the EMGE system by experiment owner. Non-unique, human friendly experiment identifier.
  — **experiment_description** - stores brief description of the experiment purpose.
  — **user_name (FK)** - reference to the entry in USER_DATA table, representing the user owning experiment defined by this table entry.
- **SUPER_TASKS table**
  Each entry in this table represents a logical group of tasks being part of the same experiment and using the same executable. Its purpose in the system is purely organizational to allow easy task grouping for purposes of presenting experiment state using user interface.
  — **superTask_id (PK)** - unique identifier for group of tasks
  — **superTask_name** - text name which identifies group of tasks. Non-unique, human friendly supertask identifier.
  — **script_id (FK)** - reference to the script executed by tasks being part of the defined super task.
  — **experiment_id (FK)** - reference to the experiment entry of which defined super task is part of.
  — **data_pack_id (FK)** - reference to the input data pack used for generation of tasks belonging to defined super task.
- **TASKS table**
  Each entry in this table represents a single task that will be scheduled for execution in the ViroLab virtual laboratory.
  — **task_id (PK)** - unique task identifier
  — **acid** - application correlation identifier - unique identifier of the application run assigned by the GridSpace Engine server.
  — **status_id (FK)** - reference to the status entry in STATUSES table, that represents task current execution status.
  — **superTask_id (FK)** - reference to the entry in SUPER_TASKS table, to which defined tasks belongs to.
  — **input_data_id (FK)** - reference to the input data entry in INPUT_DATAS table that was used as execution arguments on the task execution start.
- **SCRIPTS table**

This table stores information regarding the location of scripts that will be used for the task execution by the EMGE Task Scheduler
— **script_id (PK)** - unique script identifier in the EMGE environment.
— **script_path** - text entry representing location and identification of the script file.
— **uri_scheme** - contains information for the GridSpace Engine interpreter about URI type described by SCRIPTS table entry. Possible values are "local" for user provided files or "apprepo:svn" for experiment scripts stored in the script repository

- **STATUSES table**
This table stores information mapping the task status represented as integer into a text representation of given status.
— **status_id (PK)** - unique status identifier
— **description** - string containing text representation of the defined task status
- **SUPERTASK_DEPENDENCIES table**
This table stores information regarding execution dependencies between super tasks. Both *"depending_id"* field and *"depends_on_id"* field together form the composite primary key of each SUPERTASK_DEPENDENCIES table entry.
— **depending_id** - reference to an entry in the SUPER_TASKS table, representing super task that depends from another super task.
— **depends_on_id** - reference to an entry in the SUPER_TASKS table, representing super task that must be successfully finished in order to execute the depending super task.
— **relationship** - integer representing the number of tasks that will be created for the depending super task. For example, shall the super task one depends on consist of a 100 tasks, and we want our depending task to perform operation on groups of 20 results, the relationship value would be set to 5.
- **DATA_PACKS table**
This table stores information regarding groups of input data that are assigned to the specified super task.
— **data_pack_id (PK)** - unique data pack identifier
— **description** - short textual description of the group of input data that form this data pack.
- **INPUTS table**
This table stores information regarding input data that will be passed to the task script during its execution start.
— **input_id (PK)** - unique identifier of the input data entry.

— **input_data** - text representation of the input data that will be used during task execution start and passed as command line arguments for task script.

— **data_pack_id (FK)** - reference to the a data pack entry in the the DATA_PACKS table, of which the input data described in this table row is part of.

- **EXECUTION_INFO table**
  This table stores metadata regarding task execution. Each entry in this table corresponds to a single entry in TASKS table and is linked by having same identifier value as the task entry it corresponds to.
  — **execution_info_id (PK)** - unique identifier of the task execution information entry.

  — **execution_time** - overall time of task execution represented in milliseconds. It defines the time difference between the time of task schedulation to the Grid, and the time of task execution completion.

  — **start_date** - this field holds a timestamp of the task execution start date.

- **EXECUTION_LOG table**
  This table stores information regarding logs of the task, to which the execution metadata, this log entry is part of, is linked to.
  — **execution_info_id (FK)** - reference to an entry in the EXECUTION_INFO table, of which this execution log entry is part of.

  — **entry_text** - textual representation of this entry payload

  — **entry_type** - determines how the value stored in this entry should be interpreted: as a result, an information log or an error message.

4.4. Scheduling Manager state during experiment execution

The Scheduling Manager, responsible for management of task execution in the ViroLab virtual laboratory, performs actions in a strict order. This section presents possible states and transitions between valid states of the Scheduling Manager during its execution. Figure 4.7 shows state diagram of the Scheduling Manager.

States allowed for the Scheduling Manager during its run are as follows:

- **Created** - state of the Task Scheduler after invocation of its constructor. The Task Scheduler is not yet initialized therefore not ready for usage.
- **Ready** - in this state the Task Scheduler is prepared and ready to perform actions.
- **Scheduling Tasks** - during this state the Task Scheduler performs scheduling of tasks execution to the Grid, using experiment information stored in the database.
- **Aborting Tasks** - in this state the Task Scheduler aborts execution of jobs scheduled to the Grid.

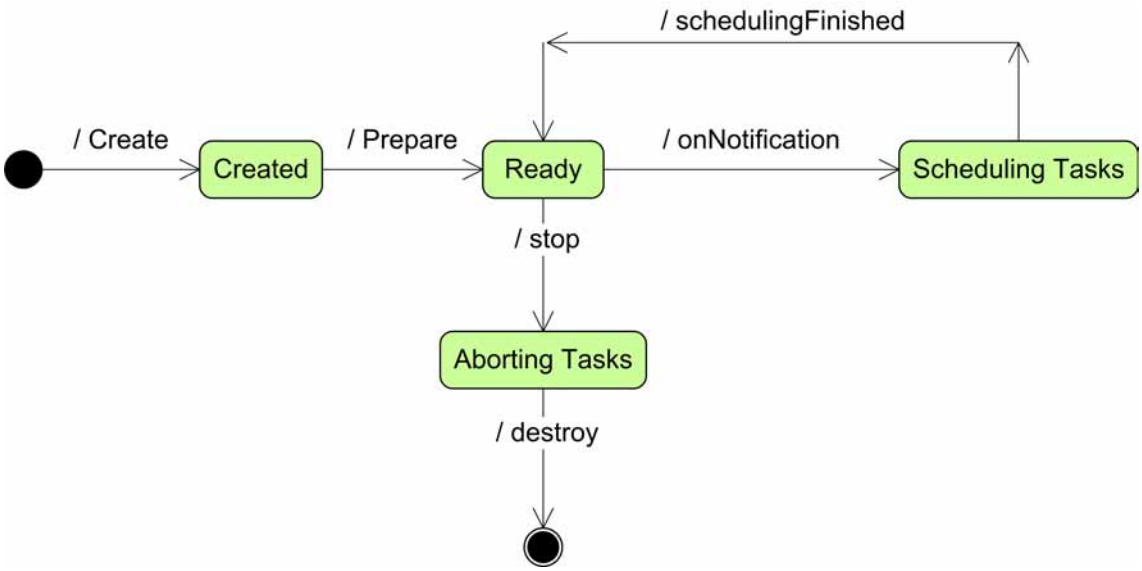

Figure 4.7. **State diagram for the Scheduling Manager.**
This diagram presents valid states for the Scheduling Manager together with transitions between states that may occur during Scheduling Manager execution. In "Created" state the manager has been created, but it is ready to use in the "Ready" state, after passing "preparation" phase. The manager then schedules new tasks execution periodically or after receiving a notification and goes back to "Ready" when scheduling operation is finished. Another possibility for the Scheduling Manager in "Ready" state is to receive stop signal. After receiving such signal, the manager cancels execution of scheduled tasks and finishes execution.

Following transitions between Scheduling Manager states are allowed:

- **Null → Created** - transition triggered on the EMGE Scheduler application launch, during this transition the Scheduling Manager is created.
- **Created → Ready** - initialization of the Task Scheduler and update of task information in the database. In case of the Task Scheduler failure, information about scheduled tasks must be reset in the database and those tasks state set to "awaiting execution".
- **Ready → Scheduling Tasks** - transition triggered by a timer or a notification about task execution end.
- **Scheduling Tasks → Ready** - after scheduling execution of tasks, the Task Scheduler returns to the ready state and awaits further actions.
- **Ready → Aborting Tasks** - transition triggered by a request to stop the Scheduling Manager execution.
- **Aborting Tasks → Null** - execution stop and destruction of the Scheduling Manager instance.

4.5. Summary

The analysis phase of the EMGE environment has been presented in this chapter. It includes discussion considering identified use cases, system architecture and the database model overview. In general EMGE system is divided into following main components: the Scheduling Manager module - daemon process responsible for management od experiments execution, the User Portal - a web interface allowing users to schedule experiments to the EMGE system and monitor their status. The database, which is the central part of EMGE, holds information required for management, scheduling and monitoring of experiments. Since the Scheduling Manager is a stateful entity and its behavior is strictly defined this chapter also contains information considering allowed Scheduling Manager states, together with description of valid transitions between defined application states.

Chapter 5

Implementation of the Environment for Management of Grid Experiments

This chapter presents implementation details and detailed design description of the EMGE system. Each section is devoted to a specific component, and presents its class structure together with control flow diagrams for important actions. List of the technologies used for implementation of the EMGE system, together with their brief description is provided in the last section.

5.1. Implementation details of the Scheduling Manager

5.1.1. Implementation details of the Security Handle Manager

Security Handle Manager is used by the Task Scheduler to obtain security handles needed during task scheduling process. The Security Handle Manager component is responsible for:

- managing and storing valid security handles of experiment owners,
- retrieving new security credential for experiment owners if needed.

Implementation details

Package named *emge.core.Shibboleth* encapsulates Security Handle Manger implementation.
Diagram in figure 5.1 presents detailed architecture of the Security Handle Manger.

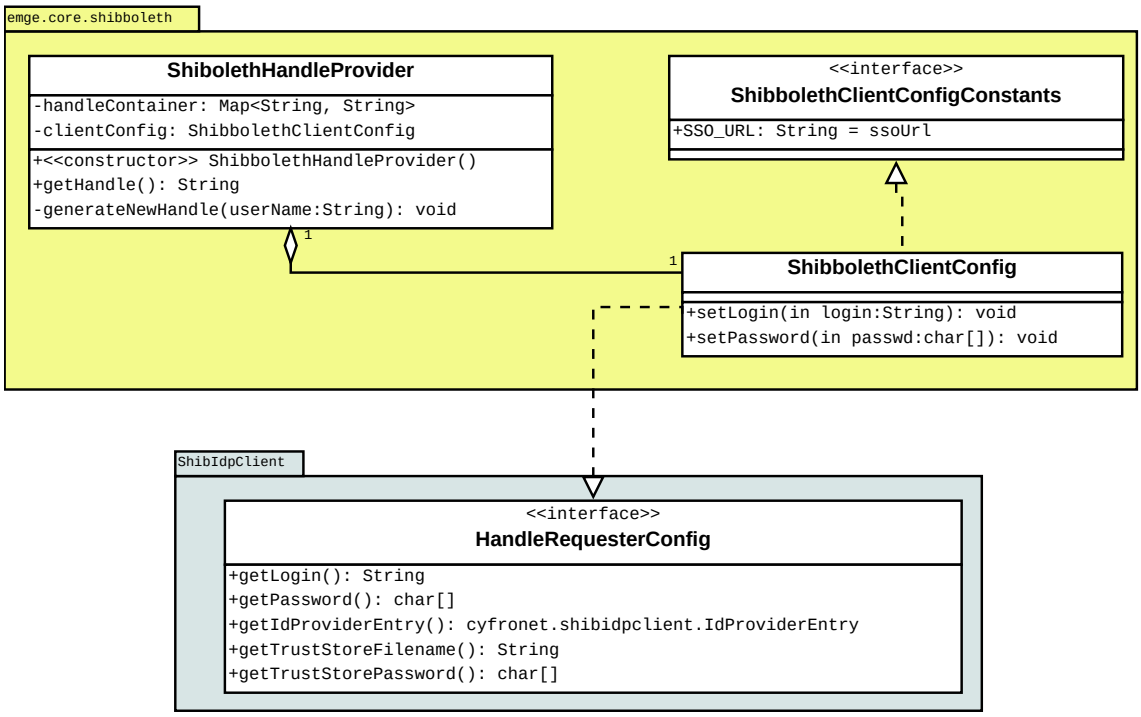

Figure 5.1. **UML class diagram of the Security Handle Manger component.** This diagram shows structure of the Security Handle Manger. It is composed of the ShibbolethClientConfig class implementing HandleRequesterConfig interface. It encapsulates al information needed to obtain a new handle for the user. ShibbolethHandleProvider acts as a handle manager, storing valid handles in memory and updating them if needed.

Obtaining Shibboleth handle for user

Sequence diagram in figure 5.2 shows Security Handle Manger control flow when it is requested for a security handle of the specified user.

The interpretation of calls presented on the diagram 5.2 is given below. Note that actions 2–8 are performed only if a security handle for the specified user is not present in the handle bank or if the requesting entity forced handle refreshment.

1: Task Scheduler requests for a security handle for the specified user. Task Scheduler also provides information shall a new handle be generated, or can the one stored by Security Handle Manager be returned.

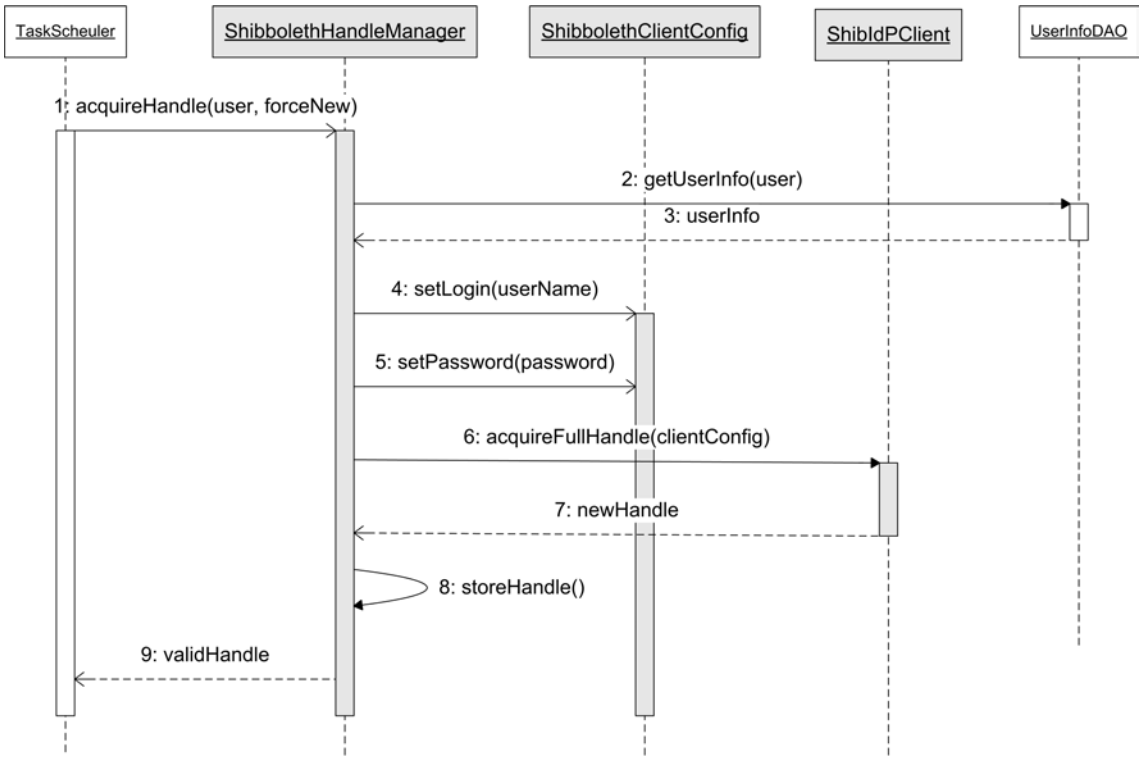

Figure 5.2. **Sequence diagram illustrating Security Handle Manger flow of control during response to handle request.**
This diagram illustrates step-by-step actions taken by the Security Handles Manager after receiving a request to provide valid handle for given user.

2: User information (which contains also user's password) is requested, using dedicated DAO, from the database.

3: User information is returned

4: The Shibboleth Handle Provider sets in the clientConfig object the name of the user on behalf of whom it will request for a new handle generation.

5: The Shibboleth Handle Provider sets in the clientConfig object the password that will be used in the request for a new handle generation.

6: Request for a new security credentials is send using the ShibIdpClient.

7: Newly generated handle is returned

8: Shibboleth Handle Provider stores the generated handle in its local handle bank

9: Security handle is returned to the requester

5.1.2. Implementation details of the Task Scheduler

Task Scheduler is the entity responsible for management of scheduling task execution to the ViroLab virtual laboratory.

Implementation Details

Implementation of the Task Scheduler module is encapsulated in a package named *emge.core*.

Figure 5.3 presents detailed architecture of the Task Scheduler component.

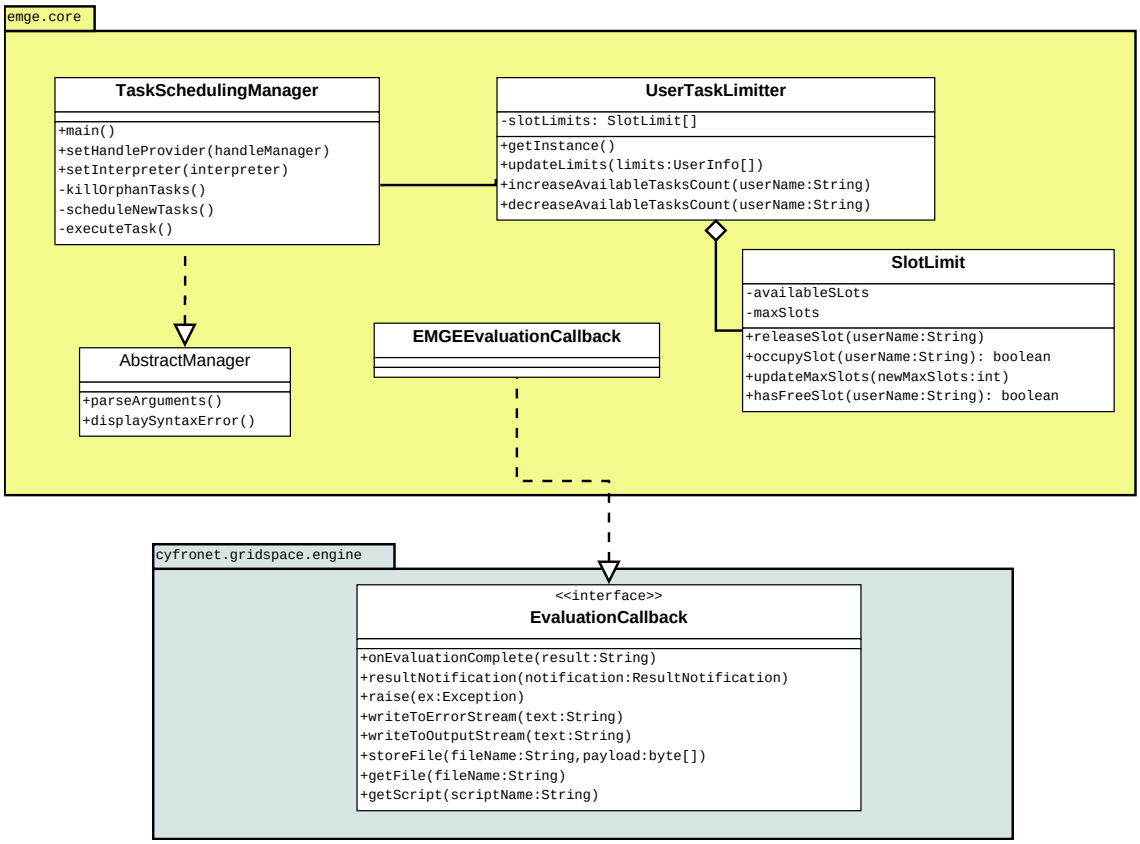

Figure 5.3. **UML class diagram of Task Scheduler component.** This diagram presents structure of the Task Scheduler module. It is composed of the entity responsible for scheduling tasks execution, evaluation callback implementation and UserTaskLimitter, an entity responsible for managing users tasks slots.

Scheduling new task

Sequence diagram in figure 5.4 shows behavior of the Task Scheduler when scheduling execution of a single task.

The interpretation of calls presented on the diagram 5.4 is given below. Note that actions 3-16 are performed only if a user free execution slot was available.

1: Request for occupation of a task execution slot of the user, who owns the experiment of which the scheduled task is part of.

2: Response for the slot occupation request, returns false if there was no free slot available, and true otherwise.

3-4: EvaluationRequest instance is created and its reference is returned.

5-6: Request for user Shibboleth security handle is made. A valid handle is returned.

7: In this operation parameters of evaluation request, such as: security handle, task input data, script location and other required parameters, are set using appropriate setter methods. Task Scheduler reads required parameters values form the database.

8: Evaluation callback is created.

9: Reference to the newly created callback object is returned to the Task Scheduler.

10: Task is scheduled to the GSEngine interpreter for execution using previously created evaluation request and evaluation callback.

11: Application Corelation IDentifier, a unique identifier of scheduled task assigned by interpreter, is returned.

12-13: Execution notification corresponding to the scheduled task is loaded from the database

14: The date of the task execution start is set.

15: Execution notification corresponding to the scheduled task is updated in the database

16-17: Application Corelation IDentifier assigned to the task during its scheduling, is written into the task object.

17: New status is set in the task object.

18: Task information in the database is updated.

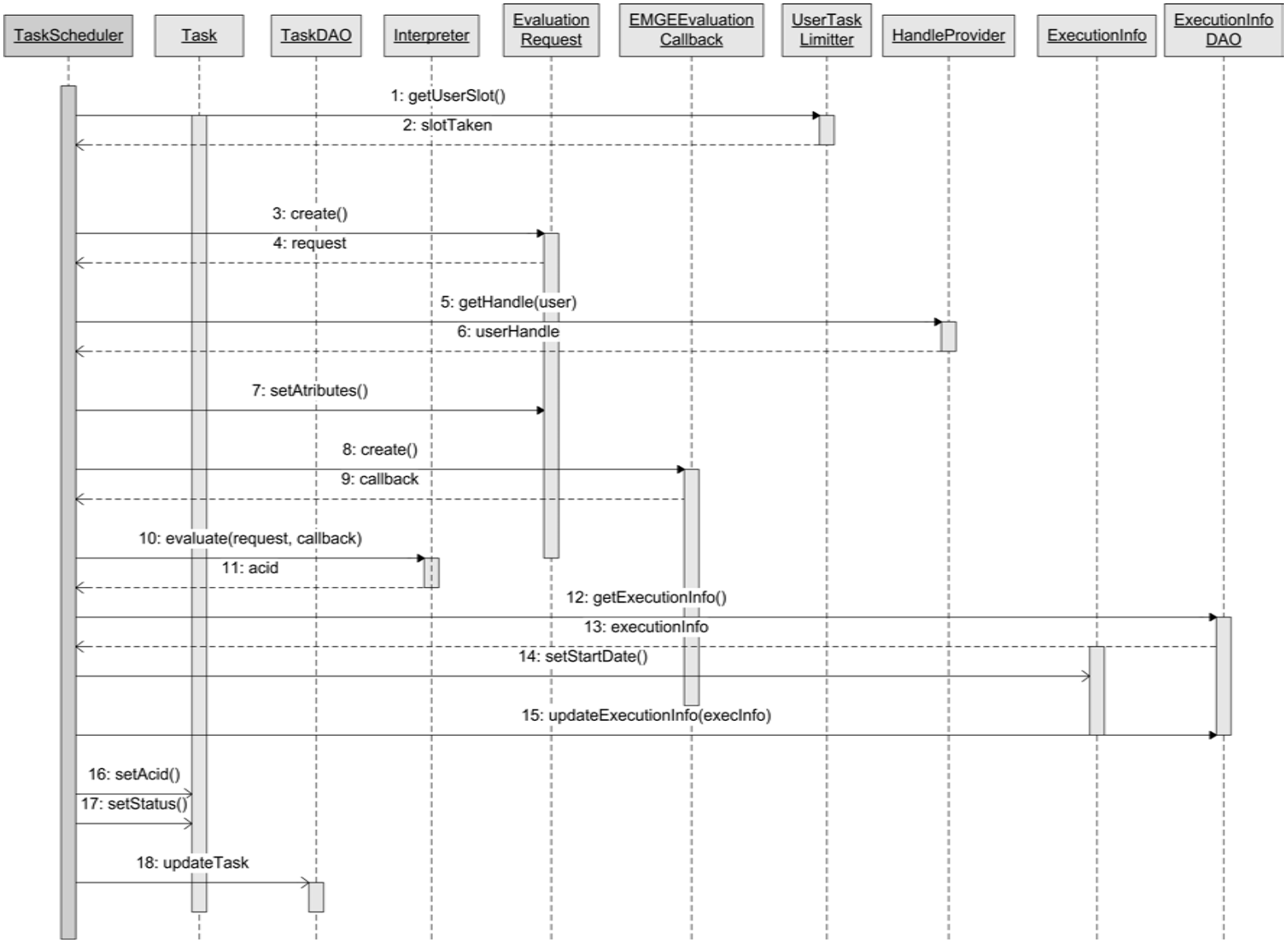

Figure 5.4. **Sequence diagram illustrating control flow in Task Scheduler during submission of task execution to grid** This diagram illustrates step-by-step actions taken by the Task Scheduler to schedule execution of tasks to the ViroLab virtual Laboratory using the GSEngine interpreter.

Canceling tasks execution on manager exit

Sequence diagram in figure 5.5 shows behavior of the Task Scheduler on application exit, when it cancels execution of scheduled tasks.

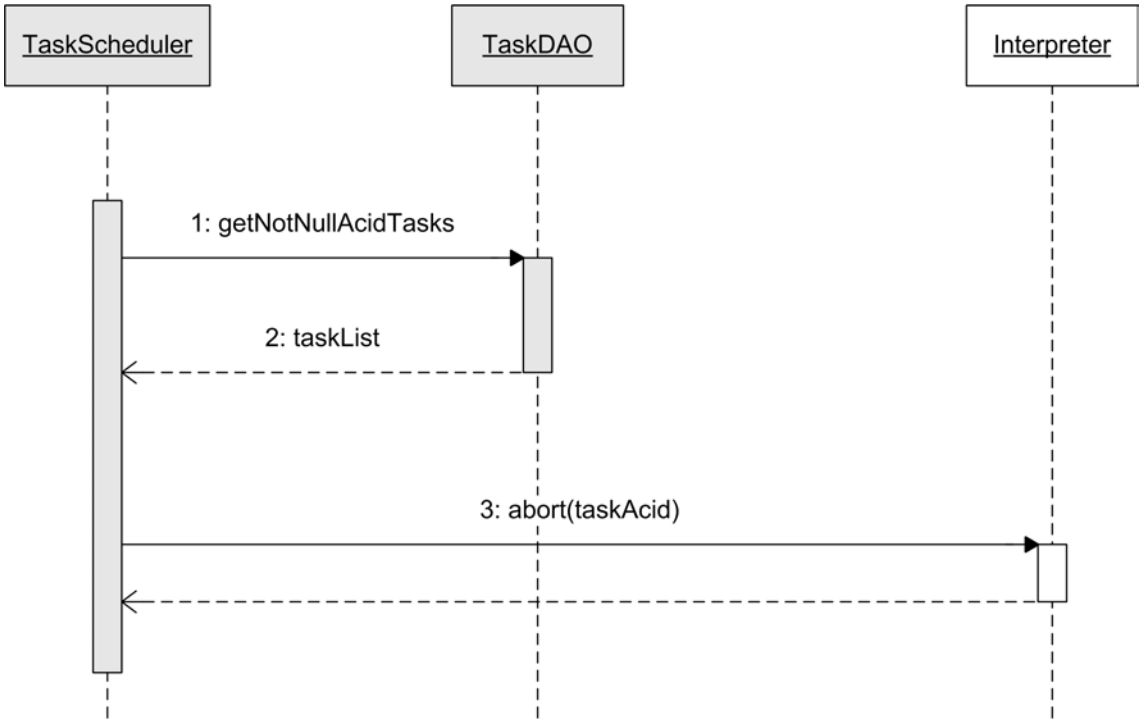

Figure 5.5. **Sequence diagram illustrating control flow in the Task Scheduler while aborting submitted task execution on application exit.**
This diagram illustrates step-by-step actions taken by the Task Scheduler to stop execution of scheduled tasks, after receiving application stop signal.

The interpretation of calls presented on the diagram 5.5 is as follows.

1: Request for a list of tasks scheduled by the Task Manager that are currently executing on the Grid.

2: List of tasks is returned.

3: Scheduled task execution is aborted.

5.1.3. Implementation details of the Task Completion Listener

Task Completion Listener is responsible for generating input data for dependant super tasks on execution finish of super task those tasks depend on.

Implementation Details

Implementation of the Task Completion Listener can be found inside the package named *emge.core*.

Figure 5.6 presents detailed architecture of the Task Completion Listener.

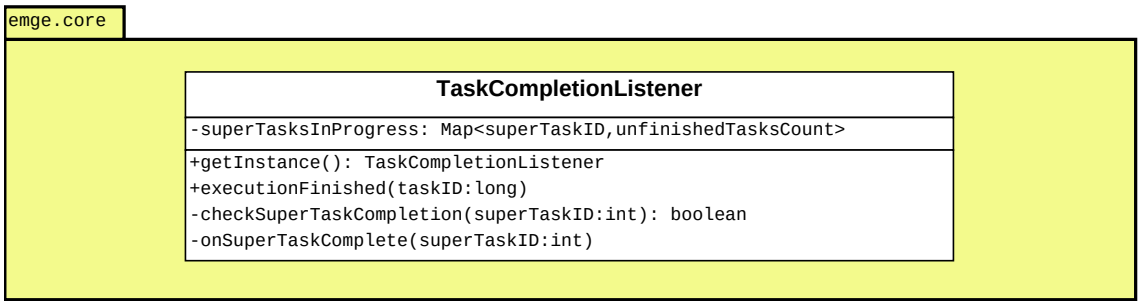

Figure 5.6. **UML class diagram of the Task Completion Listener.**

Behavior of the Task Completion Listener on task execution finish event

Figure 5.7 presents behavior of the Task Completion Listener when a task execution finishes.

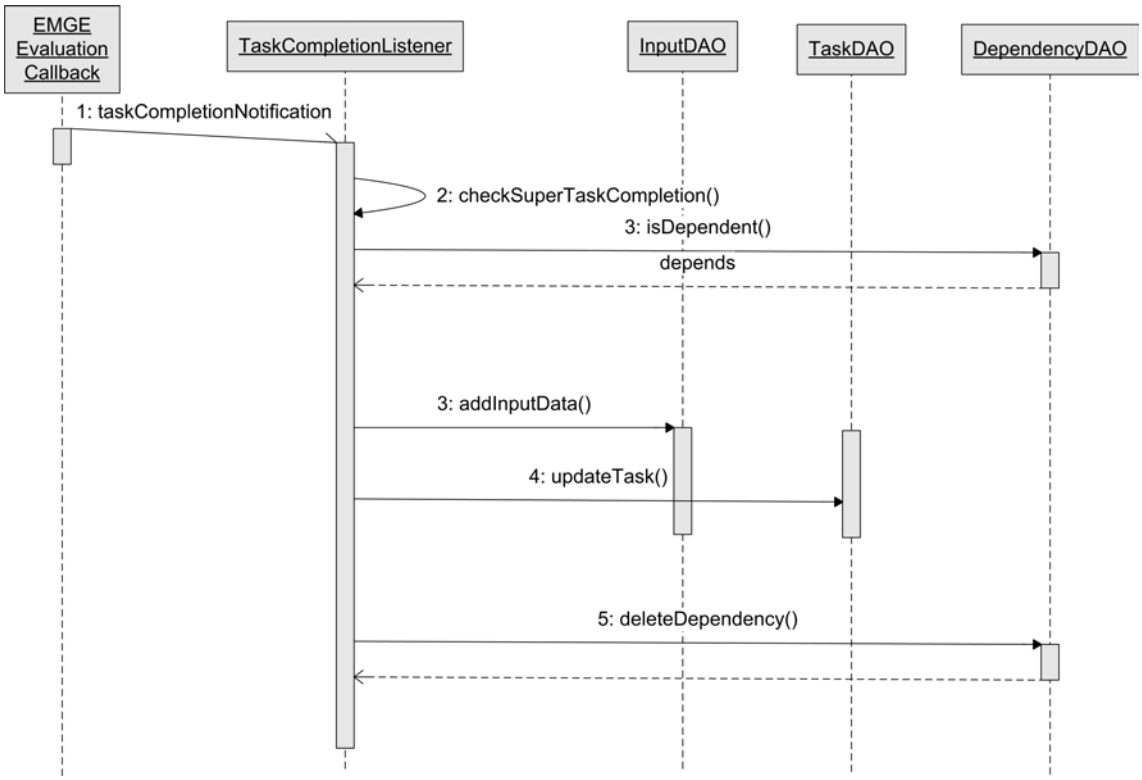

Figure 5.7. **Sequence diagram illustrating control flow in Task Completion Listener when it is informed that task execution finished.**
This diagram illustrates step-by-step actions taken by the Task Completion Listener after receiving a notification about a finished task.

The interpretation of calls presented in the diagram 5.7 is as follows(note that steps **5–7** are executed only if the finished super task had dependencies):

1: Task Completion Listener is notified about execution finish of a task.

2: Check is performed if the super task, to which the finished task belonged, has more tasks awaiting for scheduling or during execution.

3: Task Completion Listener checks if there are any super tasks dependant from the finished superTask.

4: Information about queried dependencies is returned.

5: New input data is added to the database.

6: Tasks, belonging to the dependant super task, information in the database is updated.

7: Information about no longer existing dependency is removed form the database.

5.2. Implementation details of the User Portal

User Portal has been written using the Google Web Toolkit, an open source tool for writing web applications. Please refer to section 5.5 for information on GWT. User Portal provides a web interface for EMGE users, allowing them to schedule and monitor experiments in the ViroLab virtual laboratory. User Portal source code is stored in package *emge.web*.

5.2.1. User Portal server

Server side of EMGE's User Portal acts as a bridge between the client side and the EMGE database, and uses the Database Access Component of the EMGE system to provide the client with required experiment information and submit new experiments data into the database. Implemented server side provides following functionalities for the client:

- **getExperimetns** - which retrieves list of user's experiments from the database.
- **getExperimentStructure** - which retrieves a list of super tasks that compose specified experiment.
- **getSuperTaskInformation** - which provides information about experiments tasks, their current status and execution log to client.
- **uploadExperiment** - during which the server, using information provided by the client, generates database entries representing newly defined experiment structure and uploads necessary files from the user host.

5.3. Implementation details of the Database Access Component

Database Access Component manages database connection together with all database operations, and is the only entity allowed to directly access the database. All other modules of the EMGE system use it to gain access to the database. Database Access Component uses Hibernate for object-relational mapping (ORM) (see section 5.5 for details about Hibernate).

Each database table (see section 4.3 for database tables details) has a class representing an entity stored in that table. That class is a simple Java Bean, with properties corresponding to the columns defined in the database table and is accessed using a specialized data access object (DAO).

Following description presents design details of accessing user information in the database. Because access to other tables is analogously implemented its detailed design will be omitted and only design details of access to USER_DATAS table will be presented in this thesis.

5.3.1. Implementation details

Database Access Component implementation is encapsulated in the *emge.db* package. Diagram in figure 5.8 presents implementation details of the Data Access Component with the UserData DAO.

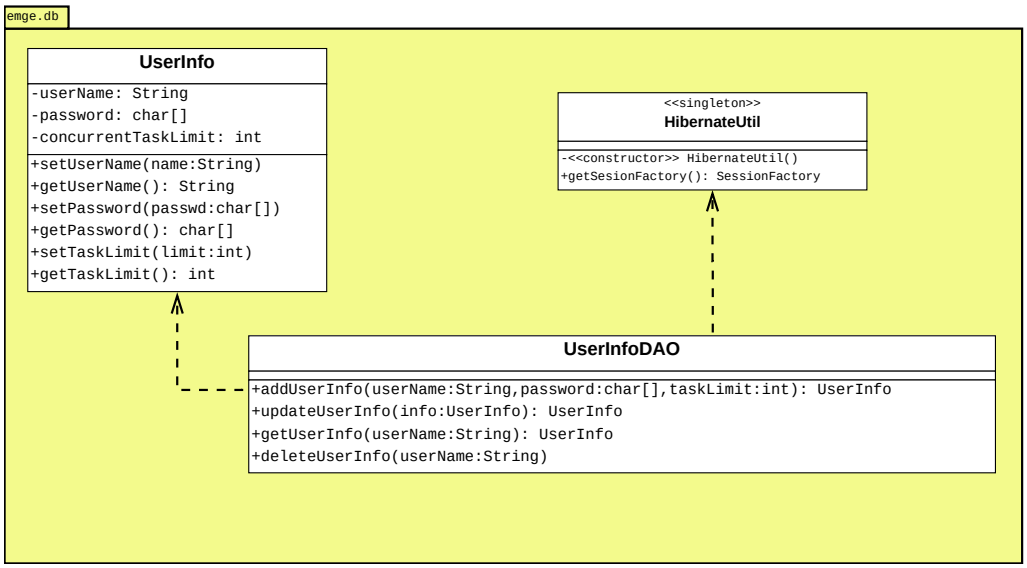

Figure 5.8. **UML class diagram of the Data Access Component.** This diagram shows structure of the Data Access Component. It is composed of the HibernateUtil - responsible for configuring database connection and creating a session factory, JavaBeans corresponding to tables defined in the database - one bean for each table, and the Data Access Objects - one DAO to access each of the database tables.

In order to provide Hibernate with information concerning database connection hibernate configuration file *hibernate.cfg.xml* is required. This file stores information required for connecting with database and a list of mapping files reflecting JavaBeans to the database tables. An example of a mapping file is presented in figure 5.9, and presents the mapping file used for object-relational mapping for the EXECUTION_NOTIFICATION table.

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="emge.db.classes">

    <class name="ExecutionInfo" table="EXECUTION_INFO">
        <id name="id" column="execution_info_id">
            <generator class="assigned"/>
        </id>

        <property name="executionTime" type="long" column="execution_time"/>
        <property name="startDate" type="timestamp" column="start_date"/>

        <bag name="description" inverse="true" cascade="all" order-by="exec_notification_id">
            <key column="execution_info_id"/>
            <one-to-many class="ExecutionNotification"/>
        </bag>

    </class>

</hibernate-mapping>
```

Figure 5.9. **Hibernate mapping file for the EXECUTION_INFO database table.** This diagram shows an example of mapping file required by Hibernate. Presented file is used for object relational mapping of the EXECUTION_INFO table. It contains name of the mapped table, name of Java Bean class the table is mapped to and description of links between table columns and JavaBean attributes columns are mapped to.

5.3.2. Adding user information to the database

Figure 5.10 shows control flow of storing a UserInfo object into the database using the UserInfoDAO. As other operations implemented in UserInfoDAO have analogical control flow to adding user information (they differ on the operation performed on a session object) presented sequence diagram will be the only one presented in this thesis for the Data Access Component.

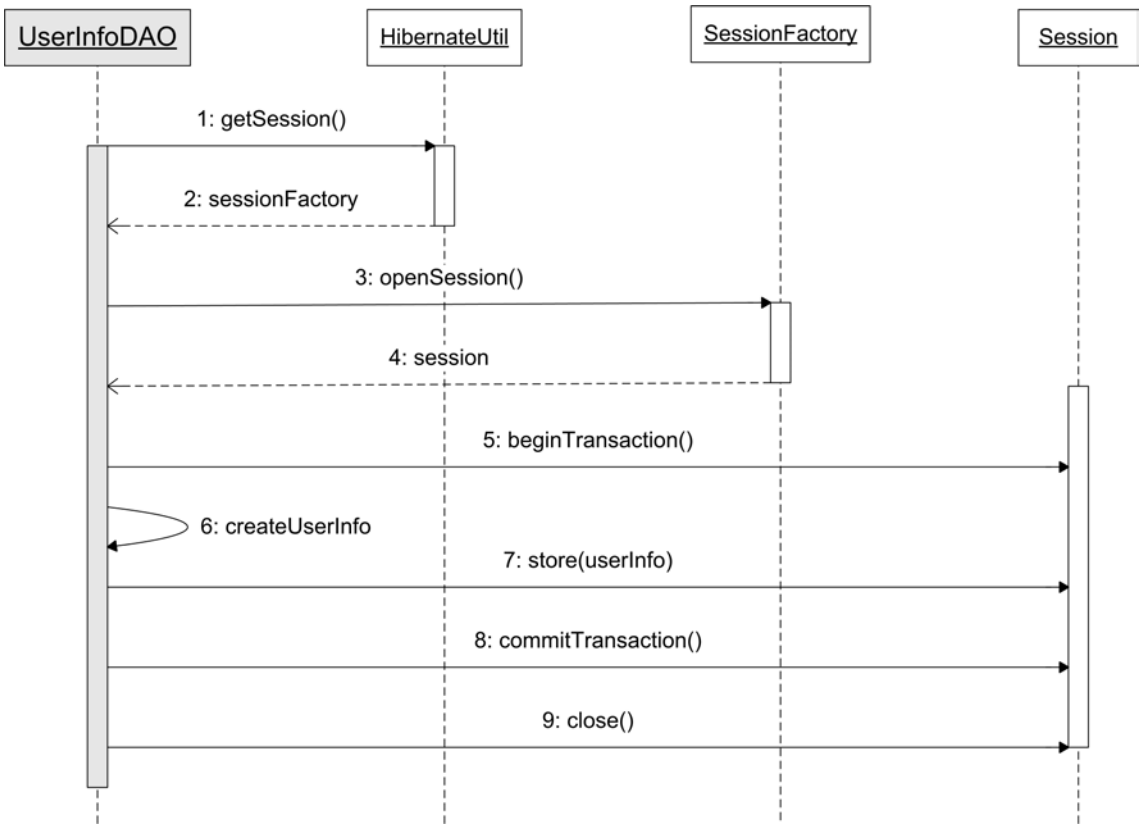

Figure 5.10. **Sequence diagram illustrating process of adding user information to the database.**
This diagram illustrates step-by-step actions taken by the UserData Access Object of the Database Access Component to store a single user data entry into the database.

Interpretation of the calls presented on diagram 5.10 is as follows:

1: UserInfoDAO requests for a session factory from HibernateUtil is made.

2: Reference to the session factory is returned.

3: Hibernate session factory is requested opening of a new database session.

4: Newly created hibernate session instance is returned.

5: UserInfoDAO requests the start of a transaction in the session.

6: UserInfo object that will be stored in the database is created.

7: Request of storing the UserInfo object into database is made.

8: Transaction is committed, and newly stored UserInfo object is persistent from now on.

9: As session is no longer needed it is closed.

5.4. Implementation details of the User Data Management Tool

User Data Management Tool is a command line tool that allows the administrator to update upper limit of simultaneously scheduled tasks by EMGE Task Scheduler for each user. Update user information is written into the database. Moreover if an instance of the Task Scheduler is running the User Data Management Tool also notifies launched Task Scheduler about new limits update.

Implementation details

User Data Management Tool implementation composes of only one class *UserDataManager* placed in package *emge.core.tools*. This class contains *main* method. In order to perform scheduler task limits update user must provide only one command-line argument to UserDataManager - path to file containing user data to be updated. This file can only include lines that match following pattern:

<user_name>:<user_password>:<simultaneous_task_limit>

Both the user_name field and the colons are mandatory, however at most one of the other two values: the user_password or the task_limit can be an empty string, thus indicating that the value will not change.

Updating users simultaneously scheduled tasks limits

Diagram in figure 5.11 show control flow during updating task limits.

Interpretation of the calls presented on diagram 5.11 is interpreted as follows:

1: User Data Management Tool reads update data from a file provided by the administrator. It also performs validation of provided data format

2: New database transaction is opened in order to perform updates.

3: For each provided user information an update is performed in the database.

4: Database transaction is committed, all performed updates become persistent.

5: User Data Management Tool performs lookup for a Task Scheduler instance using RMI Registry.

6: Reference to the Task Scheduler is returned.

7: Task Scheduler is notified about performed simultaneously running tasks limit update.

Following alternative scenarios may take place:

1a: The file containing update data is missing or file format is incorrect. In this case User Data Management Tool execution stops and appropriate error information is printed out. No database update is executed.

3a: If an error occurs during any of user information updates in the database, the whole transaction is rolled-back. User Data Management Tool execution

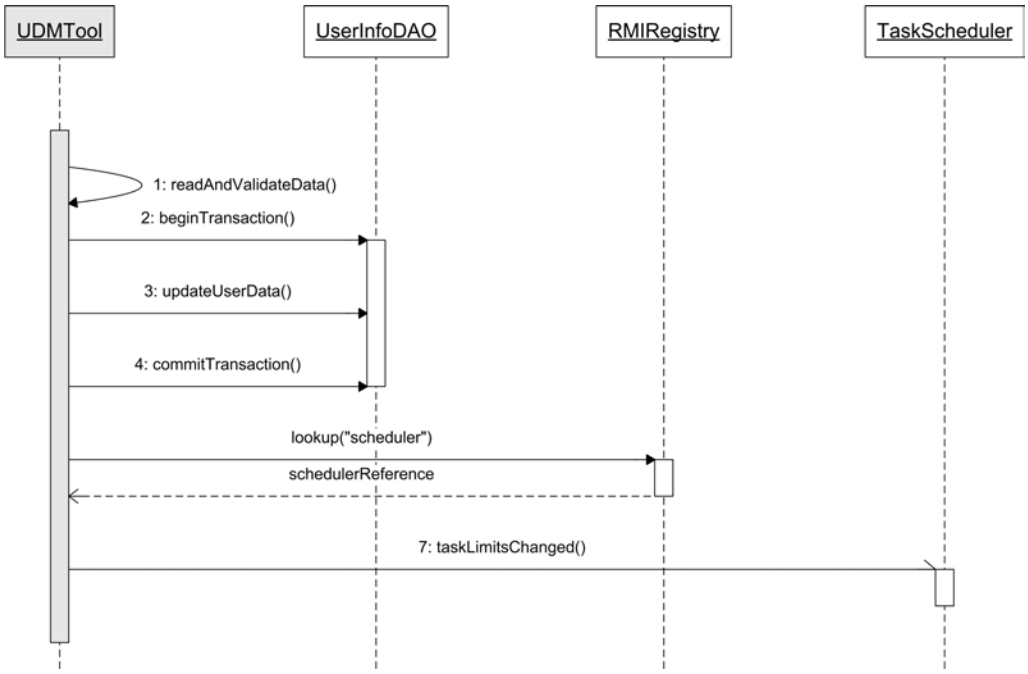

Figure 5.11. **Sequence diagram illustrating simultaneously scheduled tasks limits update performed by the User Data Management Tool.**
This diagram illustrates step-by-step actions taken by the User Data Management Tool to update number of simultaneously scheduled tasks limits.

stops and appropriate error information is printed out. No database update is executed.

7a: If lookup for the Task Scheduler in the RMI Registry did not return a reference to the Task Scheduler (the Task Scheduler is not running or for any reason it is not registered in the registry, although it may be started), step **7:** is not performed. Database entries will contain updated task limits.

5.5. Used technologies and tools

Following tolls were used during the processes of design and development of the Environment for Management of Grid Experiments:

5.5.1. Design

Microsoft Visio 2003 was used for creation of diagrams and figures presented in this thesis in chapters 4th and 5th. It was also used for creation of Figure 2.2 presenting the ViroLab virtual laboratory conceptual layers.

5.5.2. Development

- **Eclipse IDE [21]**
  Eclipse SDK is a software development environment designed for development of

Java applications. Version 3.4 of Eclipse was used for implementing Environment for Management of Grid Experiments using Java programming language.

- **JUnit [27]**
  JUnit is a unit testing framework for the Java programming language. It is used in EMGE project for performing unit tests of EMGE components. It is also used for some integration tests.
- **Google Web Toolkit (GWT) [24]**
  Google Web Toolkit is an open-source tool for support of the development and the debugging process of AJAX applications in the Java language. GWT cross-compiler performs transformation of Java written application to standalone JavaScript files.
- **Apache Tomcat Web Server [17]**
  Tomcat is an open source web server developed by the Apache Foundation. Tomcat implements the Java Servlet and the Java Servlet Pages technologies. EMGE's User Portal is deployed on Tomcat web server.
- **Hibernate 3.0 [25]**
  Hibernate is an object-relational mapping (ORM) library for the Java language. It provides a framework for mapping an object-oriented domain model to a traditional relational database.
- **Spring Framework [29]**
  The Spring Framework is an open source application framework for the Java platform. Core features of the Spring Framework can be used by any Java application, but there are extensions for building web applications on top of the Java Enterprise platform. Spring's Inversion of Control container is used in EMGE's Scheduling Manger to initialize classes manager is composed of, and inject their dependencies.

5.6. Summary

In this chapter implementation details of EMGE have been presented. EMGE has been implemented using Java SE platform 6. The Environment for Management of Grid Experiments has modular architecture that allows customization and flexible future updates mechanism. The Database Access Component, implemented as a standalone Java library, is used by other parts of the EMGE system to communicate with the database. User Portal client side has been implemented using Google Web Toolkit and provides the experiment user with an interface for monitoring and submission of experiments. Spring IoC container is used for initialization and reference settings in the Scheduling Manager on its startup. Scheduling Manager bases on information stored in the EMGE database for managing execution of experiment tasks. Finally a brief description of technologies and tools that were used during the design and implementation process of the EMGE system is provided in the last section of presented chapter.

Chapter 6

Validation of Environment for Management of Grid Experiments

In this chapter validation of the Environment for Management of Grid Experiments is presented. Next, we depict provided functionality and system properties with regard to requirement defined during the analysis phase. Finally description of performed tests is introduced.

6.1. Provided Functionality

Use cases identified during analysis phase of EMGE development (see section 4.2 for details) have resulted in fulfilling of following functionalities:

- Enable monitoring of executing experiments, providing actual and precise information on current status, scripts and data used by executed tasks.
- Allow submission of new experiments to EMGE system.
- Allow cancelation of experiment execution.
- Schedule job execution to Grid using GridSpace Engine.
- Manage security credentials required for task execution.
- Manage limits of tasks simultaneously scheduled for execution.

Providing presented functionalities is equivalent to fulfilling functional requirements defined for EMGE in section 3.2.1. Usage of existing solutions for scheduling single tasks execution (like GSEngine) provides optimal and efficient way of job executing.

6.2. Non-Functional properties of EMGE

Environment for Management of Grid Experiments satisfies all of non-functional requirements introduced in section 3.2.2. Web-interface provided by EMGE is easy to use, thus its users, like clinicians and virologists, will not have any problems to learn and use User Portal.

Presented solution is a lightweight, easy to install and configure (see Appendix A for installation and configuration instructions), standalone, flexible system. EMGE can be installed on any modern host. Operation performed both by EMGE's Task Scheduler and User Portal do not use much amount of processor time neither have high memory consumption. Designed application is platform independent and can be deployed on any machine supporting Java language.

6.3. EMGE Unit Tests

During or before development of EMGE components unit test were implemented for each class of Data Access Component, Scheduling Manager, User Data Management Tool and server side of User Portal. Their intention was to test behavior correctness of each class in situations expected during EMGE execution, including proper behavior in error situations. All of the unit tests had to be passed in order to release EMGE project. The most detailed tests were performed for the Database Access Component (section 5.3) since all other component base on information stored in database and any misbehavior of Database Access Component would be fatal for EMGE.

6.4. EMGE's integration and deployment

Environment for Management of Grid Experiments was successfully integrated with GridSpace Engine client, Shibboleth Identity Provider Client and Result Management library. All of the connection with external components were tested during prototype releases of EMGE. They are fully functional and work correctly. Communication between internal EMGE components works without errors. User Portal prototype has been deployed on Apache Tomcat Server and successfully provides expected functionality.

Example *protein folding* experiment has been scheduled using User Portal and executed correctly by Scheduling Manager giving expected results. Figure 6.1 shows results obtained during described experiment test run.

6.5. Summary

Goal of this chapter was to present validation of Environment for Management of Grid Experiments. EMGe's unit tests description together with integration and deployment results have been presented in this chapter. EMGE supports all use cases described in section 4.2 and satisfies all functional and non-functional requirements presented in sections 3.2.1 and 3.2.2.

Figure 6.1. **Screen-shot of the Experiment Monitor web page showing successful execution of protein folding experiment by the Environment for Management of Grid Experiments.**

Chapter 7

Conclusions and Future Work

This Chapter summarizes goals achieved in this thesis and introduces directions of the Environment for Management of Grid Experiments enhancements that would increase its functionality and performance.

7.1. Conclusions

The main goal of this thesis, providing an environment for management of experiments on the grid has been successfully achieved. A throughout research of existing experiment management environments: ZENTURIO, Askalon, DIANE and Nimrod toolkit was performed, pointing out weak ans strong points of those solutions in Chapter 2. High level abstraction of environment for the Environment for Management of Grid Experiments have been introduced in Chapter 3, including the ViroLab experiment management problem analysis followed by solution proposal for identified problems. Functional requirements, identified during problem analysis, for designed system have been listed with their brief description along with non-functional environment requirements. Main concepts for the Environment for Management of Grid Experiments have also been introduced in Chapter 3, including such concepts as: database oriented architecture, independent EMGE modules and Web-based user interface. The analysis phase of the EMGE environment has been presented in Chapter 4. It includes discussion considering identified use cases, system architecture and the database model overview. The designed EMGE system is divided into following main components: the Scheduling Manager module - daemon process responsible for management od experiments execution, the User Portal - a web interface allowing users to schedule experiments to the EMGE system and monitor their status. The database, which is the central part

of EMGE, holds information required for management, scheduling and monitoring of experiments. Chapter 5 presents details of EMGE implementation. The Environment for Management of Grid Experiments has modular architecture that allows customization and flexible future updates mechanism. The Database Access Component, implemented as a standalone Java library, is used by other parts of the EMGE system to communicate with the database. User Portal client side has been implemented using Google Web Toolkit and provides the experiment user with an interface for monitoring and submission of experiments. Spring IoC container is used for initialization and reference settings in the Scheduling Manager on its startup. Scheduling Manager bases on information stored in the EMGE database for managing execution of experiment tasks. Finally a brief description of technologies and tools that were used during the design and implementation process of the EMGE system is provided in the last section of the 5th Chapter. All off the performed tests (unit tests and integration tests) showed the completeness and correctness of the developed system. Moreover, a test experiment composed of more than 1000 tasks has been successfully executed by EMGE.

7.2. Future Work

Although the work has been successfully completed the following upgrades could be done in the future to EMGE in order to improve its functionality and increase user comfort of using EMGE:

Adaptation of EMGE to use experiment scripts requiring user input during their execution

Many experiment scripts available in the ViroLab virtual laboratory require user input during their execution, and EMGE was not designed to support such experiments. Currently such experiments cannot be used by EMGE users, although those scripts provide valuable, well tested functionality.

Drag&Drop experiment defining mechanism

To make EMGE more user-friendly environment, a drag&drop mechanism for defining and submitting experiments for execution could be created. Currently available scheduling mechanism in user portal is fully functional and allows scheduling of very complex workflow experiments. Unfortunately wrongly composed workflows may be written to EMGE system not because of any implementation errors but due to human mistakes. Using currently available textual workflow specification mechanism for large experiments, composed of more than 20 super tasks, can be very problematic to use, without generating wrong workflows, for humans beings.

Bibliography

[1] D. Abramson, J. Giddy, L. Kotler. High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid? In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 520–528, May 2000. Cancun, Mexico.

[2] D. Abramson, R. Buyya, J. Giddy. Nimrod/G: An Architecture of a Resource Management and Scheduling System in a Global Computational Grid. In *HPC Asia 2000*, pages 283–289, May 2000. Beijing, China.

[3] E. Ciepiela, J. Kocot, T. Gubala, M. Malawski, M. Kasztelnik, M. Bubak. Virtual Laboratory Engine – GridSpace Engine. In *Cracow Grid Workshop 2007 Workshop Proceedings*, pages 53–58. ACC CYFRONET AGH, 2008.

[4] I. Foster. What is the Grid? - a three point checklist. *GRIDtoday*, 1(6), 2002.

[5] I. Foster, C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufman, November 1998.

[6] I. Foster, C. Kesselman, S.Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal Supercomputer Applications*, pages 200–220, 2001.

[7] J.T. Moscicki. Distributed Analysis Environment for HEP and Interdisciplinary Applications. *Nuclear Instruments and Methods in Physics Research*, 502(ISSN 0168-9002), 2003.

[8] J.T. Moscicki. DIANE - Distributed Analysis Environment for GRID-enabled Simulation and Analysis of Physics Data. In *Nuclear Science Symposium Conference Record, 2003 IEEE*, volume 3, pages 1617–1620 Vol.3. NSS IEEE 2004, October 2003.

[9] M. Bubak. Virtual Laboratory for Collaborative Applications. In *Handbook of Research on Computational Grid Technologies for Life Sciences, Biomedicine and Healthcare*, pages 35–40, 2009.

[10] M. Wieczorek, R. Prodan, T. Fahringer. Scheduling of Scientific Workflows in the ASKALON Grid Environment. *ACM SIGMOD Record*, 34(3):56–62, 2005.

[11] R. Prodan, R. Duan, T. Fahringer, J. Qin, A. Villazon, M. Wieczorek. Real World Workflow Applications in the Askalon Grid Environment. In *Advances in Grid Computing – European Grid Conference 2005*, volume 3470 of *Lecture Notes in Computer Science*, pages 454–463, February 2005.

[12] R. Prodan, T. Fahringer. ZEN: A Directive-based Language for Automatic Experiment Management of Parallel and Distributed Programs. In *31st International*

Conference on Parallel Processing, pages 93–100. IEEE Computer Society, August 2002.

[13] R. Prodan, T. Fahringer. ZENTURIO: An Experiment Management System for Cluster and Grid Computing. In *In Proceedings of the 4th International Conference on Cluster Computing (CLUSTER 2002)*. IEEE Computer Society Press, 2002.

[14] R. Prodan, T. Fahringer. ZENTURIO: A Grid Service-based Tool for Optimizing Parallel and Grid Applications. *Journal of Grid Computing*, 2(1):15–29, February 2005.

[15] T. Fahringer, J. Qin, S. Hainzer. Specification of Grid Workflow Applications with AGWL: An Abstract Grid Workflow Language. In *Proceedings of IEEE International Symposium on Cluster Computing and the Grid 2005 (CCGrid 2005)*, Cardiff, UK, May 9-12, 2005. IEEE Computer Society Press.

[16] T. Gubala, B. Balis, M. Malawski, M. Kasztelnik, P. Nowakowski, M. Assel, D. Harezlak, T. Bartynski, J. Kocot, E. Ciepiela, D. Krol, J. Wach, M. Pelczar, W. Funika, M. Bubak. ViroLab Virtual Laboratory. In *Cracow Grid Workshop 2007 Workshop Proceedings*, pages 35–40. ACC CYFRONET AGH, 2008.

[17] Apache Tomcat Web Server.
http://tomcat.apache.org/.

[18] Askalon project official site.
http://www.askalon.org.

[19] Distributed European Infrastructure for Supercomputing Applications official site.
http://www.deisa.eu/.

[20] DIANE: Distributed Analysis Environment project site.
http://it-proj-diane.web.cern.ch/it-proj-diane/.

[21] Eclipse SDK.
http://www.eclipse.org/.

[22] EGEE project official site. http://public.eu-egee.org/.

[23] GridSpace Engine.
http://virolab.cyfronet.pl/trac/vlruntime.

[24] Google Web Toolkit.
http://code.google.com/webtoolkit/.

[25] Hibernate.
https://www.hibernate.org/.

[26] Java Platform 6 SE.
http://java.sun.com/javase/6/.

[27] JUnit testing framework.
http://www.junit.org/.

[28] Nimrod Toolkit project site.
http://messagelab.monash.edu.au/Nimrod.

[29] Spring Framework.
http://www.springsource.org/.

[30] ViroLab Virtual Laboratory site by ACC Cyfronet AGH.
http://virolab.cyfronet.pl.

[31] ViroLab official site.
http://virolab.org.

[32] ZENTURIO Experiment Management System for Cluster and Grid Computing. http://www.dps.uibk.ac.at/projects/zenturio/zenturio.html.

Appendix A

Administrator's Manual

This chapter presents information about installation procedures and information required to successfully deploy the Environment for Management of Grid Experiments. The installation instructions are followed by a brief description of configuration of the Environment for Management of Grid Experiments.

A.1. Installation of the Environment for Management of Grid Experiments

This section provides complete step-by-step installation instructions the for Environment for Management of Grid Experiments.

A.1.1. System requirements

The Environment for Management of Grid Experiments can be set up on any modern computer with any operating system. Successful installation and deployment of EMGE requires the following components on target machine:

- **Java 6 SE [26]** - required for every EMGE component,
- **Apache Tomcat 6.0 [17]** or any other Java Servlet version 2.5 compliant container is required for the User Portal deployment,
- **MySQL Database** is required for every EMGE component (all of EMGE components must use the same database!). EMGE distribution includes a MySQL database connector. EMGE can be used with any other database type that is supported by Hibernate [25] but in such case you must provide it with a valid database connector library.

A.1.2. Installing and running the Scheduling Manager

The following steps need to be taken in order to install and launch the Environment for Management of Grid Experiments Scheduling Manager.

1. **Download EMGE Scheduling Manager distribution**
   Binary distribution of the EMGE Scheduling Manager is available to download from site: *https://gforge.cyfronet.pl/frs/?group_id=80*. Each release archive contains following folders:
   - **bin** - contains executables used to launch the Scheduling Manager,
   - **lib** - contains binaries required to launch the Scheduling Manager,
   - **config** - contains configuration files required for the Scheduling Manger to work correctly.

   Use your favorite web browser to download the EMGE Scheduling Manager distribution from listed previously web site.
2. **Unpack the archive**
   Unpack downloaded archive into a folder where you want EMGE Scheduling Manager to be installed.
3. **Configure**
   For configuration information please refer to the section A.2 of this document.
4. **Launch application**
   Using command line console enter the "bin" directory of downloaded and unpacked distribution. When in the *bin* folder on Linux systems type:

   ```
   ./emge.sh --help
   ```

   and on Microsoft Windows operating system type:

   ```
   emge.bat --help
   ```

   for detailed information how to launch Scheduling Manager.

A.1.3. User Portal installation and deployment

The following steps need to be taken in order to deploy the Environment for Management of Grid Experiments User Portal.

1. **Download the User Portal distribution**
   Binary distribution of the EMGE User Portal is available to download from the site: *https://gforge.cyfronet.pl/frs/?group_id=80*.
   Use your favorite web browser to download EMGE Scheduling Manager distribution from listed previously web site.
2. **Unpack the archive**
   Unpack downloaded archive. After unpacking the archive you will see a *emgeWeb* folder that contains the distribution of User Portal.

3. **Configure**
   For configuration information please refer to the section A.2 of this document.

4. **Deploying the User Portal on a Apache Tomcat web server** In order to deploy the EMGE User Portal to an Apache Tomcat web server you need to perform the following steps:
   - **Stop Apache Tomcat web server**
     In order to stop Tomcat, on Linux operating system execute following commands in system console:

     ```
     cd $CATALINA_HOME
     bin/shutdown.sh
     ```

     On Microsoft Windows operating system execute:

     ```
     cd %CATALINA_HOME%
     bin\shutdown.bat
     ```

     If environment variable *CATALINA_HOME* is not present, enter Apache Tomcat web server installation directory manually.

   - **Copy emgeWeb to Tomcat**
     To copy extracted *emgeWeb* directory to the *webapps* folder in Apache Tomcat home directory, navigate in console to directory containing extracted User Portal distribution. On Linux operating system execute following commands in system console:

     ```
     cp -r ./emgeWeb $CATALINA_HOME/webapps/emgeWeb
     ```

     On Microsoft Windows operating system execute:

     ```
     copy emgeWeb %CATALINA_HOME%\webapps\emgeWeb
     ```

   - **Grant the User Portal allowance to upload files into the file system.**
     Due to security reasons, the Apache Tomcat web server by default restricts deployed on it applications to access local file systems. As the User Portal uploads scripts provided by users to execute during experiments we need to store them on the machine the User Portal is deployed on. To do that we need to add following lines to *CATALINA_HOME/conf/catalina.policy* file:

     ```
     grant codeBase "file:${catalina.home}/webapps/emgeWeb/-" {
       permission java.io.FilePermission "folder_path/*", "read,write,delete";
       permission java.net.SocketPermission "<db_host>:<port>", "connect,resolve";
     };
     ```

   - **Start back the Apache Tomcat web server**
     T start Tomcat, on Linux operating system execute following commands in

system console:

```
cd $CATALINA_HOME
bin/startup.sh
```

On Microsoft Windows operating system execute:

```
cd %CATALINA_HOME%
bin\startup.bat
```

If environment variable *CATALINA_HOME* is not present enter Apache Tomcat web server installation directory manually.

A.2. Environment for Management of Grid Experiments configuration

This section provides information about configuration of all EMGE components.

A.2.1. Configuring Hibernate database connection

Both the User Portal and the Scheduling Manager use Hibernate to maintain and manage the database connection. All settings required for the Hibernate are set in the hibernate configuration file *hiberante.cfg.xml* which is distributed in the releases of the User Portal and the Scheduling Manager. You need to specify information regarding database connection in those files. The administrator needs to specify the database location together with the database access password in provided files. You need to specify values of following properties:

- **connection.url** - url storing the database server hostname, connection port number and name of the database used by EMGE
- **connection.username** - database login
- **connection.password** - database access password

Figure A.1 presents an example of configuration of database connection in hibernate.cfg.xml.

For advanced information about Hibernate configuration please refer to [25].

A.2.2. Configuration of the User Portal

User Portal configuration requires only to specify the name of the folder the User Portal will use for storing uploaded script files. It is a simple property file named *emgeWeb.properties* stored in *config* folder of the User Portal distribution. It holds the following properties:

- **upload_root** - Directory root used to store uploaded scripts for scheduled experiments. Specifying this property is mandatory.

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

    <session-factory>

        <!-- MySQL Database connection settings -->
        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="connection.url">jdbc:mysql://localhost:3306/emge</property>
        <property name="connection.username">user</property>
        <property name="connection.password">password</property>

        <!-- JDBC connection pool (use the built-in) -->
        <property name="connection.pool_size">1</property>
        <!-- SQL dialect -->
        <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
        <!-- Disable the second-level cache  -->
        <property name="cache.provider_class">org.hibernate.cache.NoCacheProvider</property>
        <!-- Drop and re-create the database schema on startup -->
        <property name="hbm2ddl.auto">create</property>

        <mapping resource="emge/db/classes/DataPack.hbm.xml"/>
        <mapping resource="emge/db/classes/Dependency.hbm.xml"/>
        <mapping resource="emge/db/classes/Experiment.hbm.xml"/>
        <mapping resource="emge/db/classes/ExecutionInfo.hbm.xml"/>
        <mapping resource="emge/db/classes/ExecutionNotification.hbm.xml"/>
        <mapping resource="emge/db/classes/Input.hbm.xml"/>
        <mapping resource="emge/db/classes/Script.hbm.xml"/>
        <mapping resource="emge/db/classes/Status.hbm.xml"/>
        <mapping resource="emge/db/classes/SuperTask.hbm.xml"/>
        <mapping resource="emge/db/classes/Task.hbm.xml"/>
        <mapping resource="emge/db/classes/UserInfo.hbm.xml"/>

    </session-factory>

</hibernate-configuration>
```

Figure A.1. **Example of hibernate.cfg.xml configuration file.**

- **tmp_root** - Directory root used for temporary storing of files during scheduling process of experiments.

```
upload_root=C:\\emgeTmp
tmp_root=C:\\emgeTmp\\uploaded
```

Figure A.2. **Example of an emgeWeb.properties configuration file.**

A.2.3. Configuration of the Scheduling Manager

Scheduling Manager configuration requires the following files to be present:

emge.properties

This file stores information about the GSEngine server to be used for scheduling experiments and password needed to decrypt user passwords stored in the EMGE database. It holds the following properties:

- **crypting.password** - password used to encrypt/decrypt user passwords stored in the database,
- **remoteInterpreter.host** - name of host on which the GSEngine server, that will be used by EMGE, resides,
- **remoteInterpreter.port** - number of port on which the GSEngine server listens for incoming connections,
- **interpreter.timeout** - GSEngine server connection timeout.

Figure A.3 presents an example *emge.properties* file.

```
crypting.password=abcdPasswd
remoteInterpreter.host=localhost
remoteInterpreter.port=4655
interpreter.timeout=100
```

Figure A.3. **Example of an emge.properties configuration file.**

truststore.gse

This file stores trusted certificate required to connect to the GridSpace Engine server. Ask your GridSpace Engine server administrator for certificate required to connect to it. After obtaining certificate file it needs to be trusted by executing *dotrust* tool from bin directory of Scheduling Manger distribution. Navigate to bin folder of installed Scheduling Manager and on Linux systems type:

```
./dotrust --help
```

and on Microsoft Windows operating system type:

```
dotrust.bat --help
```

to get detailed information how to make a GSEngine server certificate trusted please refer to the [23].

Appendix B

User Portal user's guide

This manual provides information on the User Portal usage. Extensive description of information presented in the Experiment Monitor together with detailed introduction to fields in the Experiment Creator will provide knowledge how to use the User Portal.

B.1. Using the Experiment Monitor

Figure B.1 presents a screen-shot of the Experiment Monitor web page. Interpretation of the fields marked on the presented figure is as follows:

1. **Experiment name** - Experiment names provided by the user during experiment submission. Clicking button on the left side of the experiment name either expands or hides the information about tasks that form an experiment.
2. **Task name** - Name of the executed group of tasks. Clicking button on the left side of the task name either expands or hides details about task sub-jobs.
3. **Script name** - Name of the script used for execution of sub-jobs belonging to a specified task.
4. **Task identifier** - Unique job identifier.
5. **Job status** - Presents current job status as a text information. Depending on the current status, following colors are used for job information presentation:
   - **GREEN** - used when a job has successfully finished its execution,
   - **RED** - used if an error occurred during job execution, error details can be found in the Execution Log (see further in this section),
   - **BLUE** - marks jobs that are currently during execution on the Grid,

Figure B.1. **Screen-shot of the Experiment Monitor web page with added explanation of shown information.**

- **BLACK** - marks jobs awaiting for scheduling to execution.

6. **Start date** - Shows date and time when the job was scheduled for execution to the Grid. If a job is waiting for scheduling this filed is not present.
7. **Execution time** - Shows the total time between task scheduling to the Grid and task execution completion. Showed only for completed tasks.
8. **Input data** - Input data passed as command line arguments to the specified job.
9. **Execution log** - Provides information about the results stored by executing tasks, error information and all output written by the executed job.

B.2. Using the Experiment Creator

Figure B.2 presents a screen-shot of the Experiment Creator web page. Interpretation of fields marked on presented figure is as follows:

1. **Experiment name Field** - Text field for providing a short experiment name.
2. **Experiment description Field** - Text field for providing description of the created experiment.

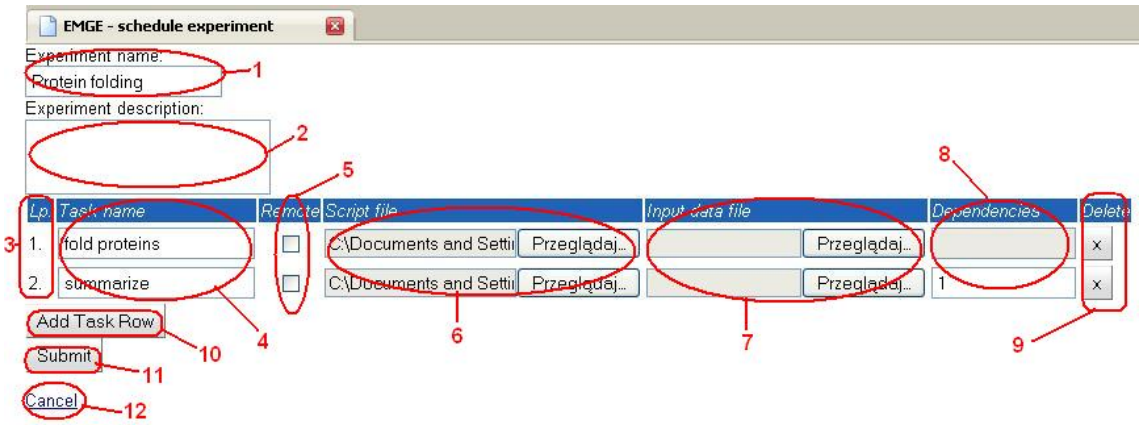

Figure B.2. **Screen-shot of the Experiment Creator web page with added explanation of user input fields.**

3. **Task number column** - This column presents numbers assigned to task defined in each of the table rows. Assigned numbers are used for defining dependencies between tasks.
4. **Task Name column** - This column hold filed for text names of submitted tasks.
5. **Repository Selection column** - Check boxes stored in this column denote if the value stored in "Script file" column point to a locally stored file - when the checkbox is unchecked, or to a remote repository - when the checkbox is checked.
6. **Script file column** - Fields in this column point the location of the script file that will be used for this task execution. It can contain either location of the script file in local file system or script location in a remote repository.
7. **Input data column** - Fields in this column point the location of file containing input data for executed script. Script will be executed number of times corresponding to number of lines in input data file, and information stored in each single line will be passed as command line arguments to executing script. Shall this field be empty, no input data is passed to the script.
8. **Dependencies column** - Fields in this column are used for providing information on dependencies and dependency relations of the defined task. For more detailed information on the dependency line format please refer to section B.2.1.
9. **Delete column** - This column stores buttons used to delete a task definition row from the table. Clicking delete button will remove the task row to which the clicked button belongs.
10. **Add Task Row Button** - After clicking this button new task definition row will appear at the end of task definition table.
11. **Submit Button** - Clicking this button submits new experiment for execution.

If any of required fields value is missing or it has a wrong format experiment is not submitted and proper information about encountered error is displayed.

12. **Cancel option** - Clicking this link cancels submitting of a new experiment to the EMGE system and returns the view to the Experiment Monitor site.

B.2.1. Dependency line format

The Environment for Management of Grid Experiments allows users not only to schedule experiment execution, but also with possibility to specify dependencies between tasks composing an experiment. To define such dependencies the user needs to provide for the task, a list of tasks on which it depends, together with dependency relationship type. Text line used for dependency specification must match the following regular expression or be an empty line (if no dependency is specified):

$$[< task\_number > (relation)?,]^* < task\_number > (relation)? \tag{B.1}$$

In other words, the line specifying dependencies is composed of single task dependency information separated by comas. Single task dependency information is composed of a task number of task, that the defined task depends on, followed by surrounded in brackets number representing the relationship type - number of sub-jobs that will be generated for defined task basing on output provided by task one is dependent to. Relation type can be omitted and it lets EMGE know that one sub-job should be generated for the defined task.

Appendix C

Step-by-step sample experiment execution tutorial

This chapter presents step by step instructions how to use the EMGE User Portal to schedule simple experiment containing task dependencies.

Introduction

Let us assume that we want to calculate the sum of squares of integers from 1 to 200, what can be represented as the following math formula:

$$\sum_{i=1}^{200} n^2.$$

We have got a script that calculates and returns square of the given number: ***square.rb***[1], and a script ***sum_rid.rb***[1] that calculates the sum of numbers stored using the ReMan library and given to the script as the ResMan ID passed as console argument to our script. Since we have access to the ViroLab virtual laboratory we can use EMGE to execute our experiment. Input data for our experiment (integers from 1 to 200) are stored in the ***sample_data.txt***[1] file.

Experiment submission to the EMGE system using User Portal's Experiment Creator portlet

In order to submit our square summing experiment we need to perform the following actions:

1. **Download required script from the EMGE web site**

[1]Both the ***square.rb*** script code and the ***sum_rid.rb*** script code can be downloaded from the EMGE web site, together with ***sample_data.txt*** file.

2. **Navigate to EMGE's Experiment Creator web page**

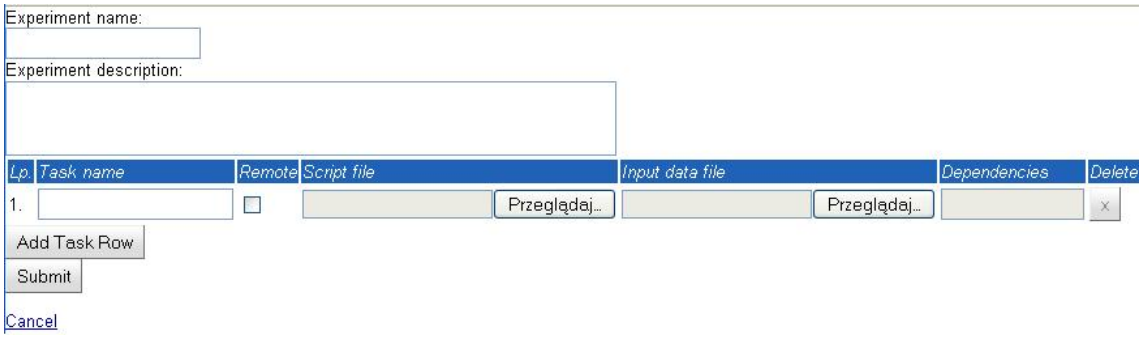

Figure C.1. **Screen-shot of the Experiment Creator portlet.**

3. **Define the workflow of our experiment**

   3.1 Now we have to define the workflow of our experiment. We set the name and the description of our experiment in appropriate fields.

   3.2 Because our experiment will be composed of two super tasks we have to click the button "Add Task Row" in order to add second task definition row to our experiment.

   3.3 We define first super task of our experiment that will be responsible for calculating squares of given integers. We set the task's name, that will be displayed in the Experiment Monitor, set the script file path to ***square.rb*** script and input data file to ***sample_data.txt***.

   3.4 Next we have to define second super task of our experiment, that will sum up calculated squares. We set the task's name and the script file path to the ***sum_rid.rb*** file. As we want the defined super tasks to perform operations on output generated by the first super task we also need to define the dependency. In the "Dependencies" text box we put the following text: **1(1)**. It informs EMGE that our task cannot be executed until the first super task finishes, and that the data generated by the first task should be used to spawn only one job in the second super task.

   3.5 After performing steps 3.1 to 3.4 we should obtain experiment definition presented on Figure C.2.

   3.6 We click on the "Submit" button to submit the defined experiment to EMGE system.

4. **Check experiment status using the Experiment Monitor, and wait until EMGE executes all of tasks of our experiment :)**

Figure C.2. **Screen-shot of the Experiment Creator portlet filled with experiment definition.**