



## **Grid Portal Development**

submitted to the Faculty

of the

Worcester Polytechnic Institute

in partial fulfillment of the requirements for the

Degree of Bachelor of Sciences

by

---

Joshua Nedelka  
Date: 4/22/2008

---

Matthew Reiter  
Date: 4/22/2008

---

Professor Gábor Sárközy, Major Advisor

---

Professor Stanley Selkow, Major Advisor

## **Abstract**

The project consists of the analysis, design and implementation of a user account creation system and a notification system for the P-GRADE Grid Portal. The user account creation system expedites the process of accessing a portal by automating many administrative tasks. The notification system provides a useful feature to users of the Portal by alerting them in real time of the status of their workflows. Both systems serve to enhance a user's experience with the Portal.

## **Acknowledgements**

We would like to thank MTA SZTAKI and WPI for putting together this program and allowing us this experience in Budapest. At SZTAKI, we would like to thank everyone from the Laboratory of Parallel and Distributed Systems (LPDS), especially Prof. Dr. Péter Kacsuk, the head of the lab and Miklós Kozlovszky for his excellent management of the project. In addition, we would like to thank Gergely Sipos for his explanations of the GRID and Portal, Gábor Hermann for his assistance with the user account creation system, Zoltán Farkas and András Schnautigel for their help during integration and development, Károly Göschl and Atilla Marosi for their assistance deploying and testing the account creation system, and Ákos Balaskó for his help in the office. We would also like to thank Ádám Kornafeld for making us feel welcome in Budapest. At WPI, we would like to thank József Patvarczki for giving us a great preparation for our project. Finally, of course, we would like to thank our advisors, Professors Gábor Sárközy and Stanley Selkow, for dedicating themselves to helping us complete our project successfully.

# Table of Contents

<b>ABSTRACT</b> .....	<b>II</b>
<b>ACKNOWLEDGEMENTS</b> .....	<b>III</b>
<b>TABLE OF FIGURES</b> .....	<b>VII</b>
<b>TABLE OF TABLES</b> .....	<b>VIII</b>
<b>1 BACKGROUND</b> .....	<b>1</b>
1.1 HISTORY OF GRID COMPUTING.....	1
1.1.1 <i>Beowulf Clusters</i> .....	1
1.1.2 <i>Origins of Grids and Grid Computing</i> .....	1
1.2 APPLICATIONS OF GRID COMPUTING.....	2
1.3 GRID TECHNOLOGIES.....	2
1.3.1 <i>The Globus Toolkit</i> .....	3
1.3.2 <i>gLite</i> .....	3
1.3.3 <i>Condor</i> .....	3
1.3.4 <i>Parallel Programming</i> .....	4
1.4 MTA SZTAKI AND LPDS .....	5
1.4.1 <i>MTA SZTAKI</i> .....	5
1.4.2 <i>LPDS</i> .....	5
1.5 THE P-GRADE GRID PORTAL .....	6
1.5.1 <i>Using the P-GRADE Portal</i> .....	7
<b>2 PROJECT STATEMENT</b> .....	<b>9</b>
2.1 USER ACCOUNT CREATION SYSTEM .....	9
2.1.1 <i>Current System</i> .....	9
2.1.2 <i>Proposed Implementation</i> .....	9
2.2 EMAIL NOTIFICATION OF WORKFLOW STATUS CHANGES .....	11
2.2.1 <i>Current System</i> .....	11
2.2.2 <i>Proposed Implementation</i> .....	12
<b>3 METHODOLOGY</b> .....	<b>15</b>
3.1 USER ACCOUNT CREATION SYSTEM .....	15
3.1.1 <i>Requirements</i> .....	15
3.1.2 <i>Implementation Considerations</i> .....	17
3.1.2.1 <i>Mostly on Web Server</i> .....	17
3.1.2.2 <i>Combination of Web Server and Portal Servers</i> .....	19
3.1.2.3 <i>Entirely on Portal Servers</i> .....	21
3.1.2.4 <i>Comparison of Implementations</i> .....	23
3.1.3 <i>Technologies Used</i> .....	24
3.1.3.1 <i>PHP: Hypertext Preprocessor</i> .....	24
3.1.3.2 <i>phpBB Confirmation System</i> .....	25
3.2 EMAIL NOTIFICATION SYSTEM .....	25
3.2.1 <i>Requirements</i> .....	25
3.2.2 <i>Implementation Considerations</i> .....	28
3.2.2.1 <i>Polling</i> .....	28
3.2.2.2 <i>Event Driven</i> .....	29

3.2.3	<i>Analysis of Implementation</i> .....	30
3.2.4	<i>Technologies Used</i> .....	31
3.2.4.1	<i>Tomcat</i> .....	31
3.2.4.2	<i>GridSphere</i> .....	32
3.3	<b>COMMON TECHNOLOGIES USED</b> .....	32
3.3.1	<i>Java</i> .....	32
3.3.1.1	<i>JSP &amp; Servlets</i> .....	35
3.3.1.2	<i>Portals &amp; Portlets</i> .....	36
3.3.1.3	<i>NetBeans</i> .....	37
3.3.2	<i>XML</i> .....	38
3.3.3	<i>XHTML</i> .....	39
3.3.4	<i>JavaScript</i> .....	39
<b>4</b>	<b>IMPLEMENTATION</b> .....	<b>41</b>
4.1	<b>USER ACCOUNT CREATION SYSTEM</b> .....	41
4.1.1	<i>Portal Server Interface</i> .....	42
4.1.1.1	<i>Account creator servlet</i> .....	42
4.1.1.2	<i>Information query servlet</i> .....	43
4.1.1.3	<i>Account creator service</i> .....	43
4.1.1.4	<i>Authentication service</i> .....	43
4.1.2	<i>Account Request Form</i> .....	44
4.1.3	<i>Account Request Verification Email</i> .....	45
4.1.4	<i>Administration Console</i> .....	47
4.1.4.1	<i>Login Form</i> .....	47
4.1.4.2	<i>Account Request Verification Form</i> .....	47
4.1.4.3	<i>User Information Exporter</i> .....	49
4.1.4.4	<i>Settings Manager</i> .....	50
4.1.4.5	<i>Portal Information Editor</i> .....	52
4.1.5	<i>Security</i> .....	53
4.2	<b>EMAIL NOTIFICATION SYSTEM</b> .....	57
4.2.1	<i>Portlet Integration</i> .....	57
4.2.1.1	<i>Modifications from WS-PGRADE Portal</i> .....	58
4.2.2	<i>Submit Form</i> .....	59
4.2.3	<i>Backend Integration</i> .....	61
4.2.3.1	<i>Retrieving and Handling Status Change Events</i> .....	61
4.2.3.2	<i>Persistence of Data</i> .....	63
4.2.3.3	<i>Notification Utilities</i> .....	65
<b>5</b>	<b>TESTING</b> .....	<b>66</b>
5.1	<b>USER ACCOUNT CREATION SYSTEM</b> .....	66
5.1.1	<i>Functional Testing</i> .....	66
5.2	<b>EMAIL NOTIFICATION SYSTEM</b> .....	66
5.2.1	<i>Performance Testing</i> .....	66
5.2.1.1	<i>One Workflow</i> .....	67
5.2.1.2	<i>20 Workflows</i> .....	67
5.2.1.3	<i>50 Workflows</i> .....	68
5.2.2	<i>Functionality Testing</i> .....	69
5.2.2.1	<i>Front End Testing</i> .....	70
<b>6</b>	<b>CONCLUSIONS</b> .....	<b>72</b>
6.1	<b>THE FINAL ACCOUNT CREATION SYSTEM</b> .....	72

6.2	THE FINAL EMAIL NOTIFICATION SYSTEM.....	72
<b>7</b>	<b>FUTURE WORK.....</b>	<b>73</b>
7.1	ACCOUNT CREATION SYSTEM.....	73
7.1.1	<i>Bulk Account Creator</i> .....	73
7.1.2	<i>Account Request Viewer</i> .....	73
7.1.3	<i>Testing of Portal Information</i> .....	74
7.1.4	<i>Improved Database Support</i> .....	74
7.2	NOTIFICATION SYSTEM.....	74
7.2.1	<i>Additional Plugins</i> .....	74
7.2.1.1	<i>SMS Messaging</i> .....	74
7.2.1.2	<i>Phone Calls</i> .....	75
7.2.2	<i>More Monitoring</i> .....	75
7.2.2.1	<i>Granularity of Notification</i> .....	75
7.2.2.2	<i>Types of Monitoring</i> .....	76
7.2.2.3	<i>Parameter Study Workflows</i> .....	76
<b>8</b>	<b>REFERENCES .....</b>	<b>78</b>
<b>A.</b>	<b>EMAIL NOTIFICATION SYSTEM PERFORMANCE GRAPHS .....</b>	<b>81</b>
<b>B.</b>	<b>ACCOUNT CREATOR SYSTEM ADMINISTRATOR MANUAL .....</b>	<b>85</b>
B.1.	INTRODUCTION.....	85
B.2.	ACCOUNT CREATION WORKFLOW.....	85
B.3.	INSTALLATION.....	89
B.3.1.	<i>Server Topology</i> .....	89
B.3.2.	<i>Prerequisites</i> .....	90
B.3.3.	<i>Installing web server components</i> .....	91
B.3.3.1.	<i>Configuring Apache</i> .....	92
B.3.3.2.	<i>Configuring the Firewall</i> .....	92
B.3.4.	<i>Installing Portal Server Components</i> .....	93
B.4.	ADMINISTRATION CONSOLE.....	95
B.4.1.	<i>Exporting User Account Information</i> .....	96
B.4.2.	<i>Settings</i> .....	97
B.4.3.	<i>Editing Portal Information</i> .....	99
B.4.4.	<i>Account Request Verification</i> .....	101
B.5.	OTHER FEATURES .....	106
B.5.1.	<i>Logging</i> .....	106
<b>C.</b>	<b>EMAIL NOTIFICATION SYSTEM HELP DOCUMENTATION.....</b>	<b>107</b>

## Table of Figures

Figure 1.1 - P-GRADE Portal Workflow Editor .....	7
Figure 1.2 - P-Grade Portal Certificate Manager.....	8
Figure 1.3 - P-GRADE Portal Information System.....	8
Figure 2.1 - Account creation workflow.....	11
Figure 2.2 - Monitoring workflow statuses in P-GRADE Portal.....	12
Figure 2.3 - WS-PGRADE Notification Portlet .....	13
Figure 3.1 - Web server implementation data flow .....	19
Figure 3.2 - Combination implementation data flow.....	21
Figure 3.3 - Portal server implementation data flow .....	22
Figure 3.4 - Confirmation code images .....	25
Figure 3.5 - Polling Flow of Events.....	28
Figure 3.6 - Event Driven Flow .....	29
Figure 3.7 - The P-GRADE Portal Running Within GridSphere .....	32
Figure 3.8 - JSP Model 2 Architecture [26].....	35
Figure 3.9 - Elements of a Portal Page [23].....	36
Figure 3.10 - The NetBeans IDE .....	38
Figure 4.1 - User account creation components.....	41
Figure 4.2 - Account creation system server topology .....	42
Figure 4.3 - Account request form.....	45
Figure 4.4 - Access request verification email.....	46
Figure 4.5 - Administration console login form .....	47
Figure 4.6 - Account request verification form .....	48
Figure 4.7 - Account request verification success .....	49
Figure 4.8 - Account request verification failure.....	49
Figure 4.9 - User account exporter .....	50
Figure 4.10 - User account creator settings .....	52
Figure 4.11 - Portal information editor .....	53
Figure 4.12 - User account creator vulnerability analysis .....	56
Figure 4.13 - Notify Portlet File Entry.....	58
Figure 4.14 - Notify Portlet Layout Entry .....	58
Figure 4.15 - Notify User Preferences .....	59
Figure 4.16 - Workflow Submission Form.....	60
Figure 4.17 - Event Driven Notification Implementation.....	61
Figure 4.18 - Workflow Job Status File.....	62
Figure A.1 - 1 Workflow with Full Notification.....	81
Figure A.2 - 1 Workflow with No Notification .....	82
Figure A.3 - 20 Workflows with Full Notification.....	82
Figure A.4 - 20 Workflows with No Notification.....	83

Figure A.5 - 50 Workflows with Full Notification ..... 83  
Figure A.6 - 50 Workflows with No Notification..... 84

**Table of Tables**

Table 3.1 - Comparison of account-creation system implementations..... 23



# **1 Background**

## **1.1 History of Grid Computing**

### **1.1.1 Beowulf Clusters**

Before discussing the beginnings of Grid computing, it is important to recognize another important development in distributed computing. In 1993, Donald Becker and Thomas Sterling conceptualized the idea of creating a cluster system from commodity hardware and open source software. This would allow the computational power to solve highly parallelizable problems while offsetting some of the costs associated with a single more powerful machine[18]. The first clusters built contained only a few machines connected with Ethernet and Fast Ethernet, but they demonstrated that powerful computational units were indeed possible with cheap, commodity hardware [6].

### **1.1.2 Origins of Grids and Grid Computing**

Prior to true Grid computing, there was Metacomputing. Metacomputing involved interconnecting supercomputers to achieve superior performance. One implementation of Metacomputing was Information Wide Area Year (I-WAY). I-WAY was developed to connect 17 supercomputing centers together, and this was successfully demonstrated at Supercomputing 1995.

Following the success of I-WAY, Ian Foster of Argonne Lab and Carl Kesselman of the University of Chicago began the Globus Project. Similar to the aims of a Beowulf cluster, a Grid would be a collection of computational resources that could be used to solve a variety of problems[6]. This differed substantially from large computational resources of the past, which could be tailored to perform one and only one task very well. The difference from the cluster, however, is that a Grid can be geographically distributed and substantially more heterogeneous.

The origin of the term “Grid” comes from an analogy of the computational resources with a power grid. The idea is that a user can connect to the Grid and receive as much computing power as needed while ignoring such details as what hardware is being used. This is the same as a person getting electricity from a wall socket – not caring how the electricity is generated or where it is coming from, the only important thing is that it continues to flow[4]. Of course, the analogy falls apart in practice since computing power cannot be drawn in the same way as electricity from a socket; rather jobs must be sent out to the Grid to be processed.

## **1.2 Applications of Grid Computing**

Current uses of grid computing are focused primarily on computation-heavy scientific research. Such research includes weather prediction, high-energy physics, genetics research and financial modeling[14]. The computing power available on individual computers, or even entire grids, is often insufficient for these tasks. Distributed grids offer a shared computing environment usable by anyone in its virtual organization in need of performing large computations that would otherwise be infeasible due to time or memory constraints.

## **1.3 Grid Technologies**

Grids are complex systems comprised of multiple technologies, often from different vendors. At the lowest level are the individual computers and clusters that make up a grid. Grid middleware, such as Globus, provides an interoperability layer that allows the elements of a grid to communicate with each other. Running on top of the middleware are portals such as P-GRADE, which hide the differences between various middleware solutions and allow multiple grids to be treated in a uniform manner by both the users and the jobs running on the grids.

### **1.3.1 The Globus Toolkit**

Globus is composed of three parts: the Globus Toolkit, a community of users, and the web-based infrastructure that supports the community[8]. Of primary interest is the toolkit, as it comprises the software component of Globus.

The Globus Toolkit is “a set of libraries and programs that address common problems that occur when building distributed system services and applications”[8]. It is based on the Open Grid Services Infrastructure (OGSI) and is intended to increase the ability to reuse and extend OSGI technology for new grid applications[25]. The Globus Toolkit consists of a default set of service implements for managing the low-level grid infrastructure, a security infrastructure, tools for building new web services, client interfaces in the form of application programming interfaces and command line tools, and extensive documentation[8].

### **1.3.2 gLite**

gLite is an alternative grid middleware to Globus and was developed as part of the Enabling Grids for E-Science (EGEE) project. gLite is a lightweight framework used to build grid applications that allows them to take advantage of diverse computing and storage elements that may be geographically distributed [9].

### **1.3.3 Condor**

Condor is a system for distributed computing known as a batch execution system[17]. Condor provides various services, including job management, scheduling policy, priority schemes, and resource monitoring and management. Condor’s design philosophy revolves around flexibility and is embodied by the following four statements [17]:

- “Let communities grow naturally.” People have a desire to work together, but they have different needs. Condor permits cooperation, but does not require it; relations will grow according to necessity.

- “Leave the owner in control, whatever the cost.” The owner of a given resource must remain in full control of its policies and may withdraw the resource at any time. If too much control is taken from the owner, people are less likely to join the system.
- “Plan without being picky.” If the community is of a sufficient size, there will always be idle resources available. However, not all resources will work all the time or correctly. As a result, the system should not depend on any given resource being available or continuing to operate correctly. The system should be able to anticipate failures and react accordingly to reassign work to other resources.
- “Lend and borrow.” Knowledge and expertise should be shared among the community. Understanding previous research is the key to future progress; otherwise, the same mistakes and discoveries may be repeated.

#### **1.3.4 Parallel Programming**

Parallel programming, also known as parallel computing, embodies the philosophy that most difficult problems can be broken down into smaller independent tasks that can be executed in parallel. These parallel tasks may be executed simultaneously on a vector processor, multiple processors in a given computer, or even on multiple distributed computers. The main advantage of parallel programming over sequential programming is that it supports much greater scalability. As long as there are more parallel tasks than there are processing units, the rate at which the overall calculation is accomplished can be increased simply by adding more processing units[22].

## **1.4 MTA SZTAKI and LPDS**

### **1.4.1 MTA SZTAKI**

MTA SZTAKI, The Computer and Automation Research Institute of the Hungarian Academy of Sciences, is an application-oriented research institution specializing in computer science and engineering. Among their fields of research and development, as stated by their website, are:

- artificial intelligence methods
- expert- and knowledge-based systems in medicine and process supervisory systems
- robust control, simultaneous identification and in integrated vehicle control system
- computer-integrated manufacturing systems
- distributed information systems and management
- new technologies for local and wide area networks, www-based and multimedia tools
- cluster and grid computing

SZTAKI has cooperation with most of the technical universities in Hungary as well as some within the U.S.A [27].

### **1.4.2 LPDS**

The LPDS, or Laboratory of Parallel and Distributed Systems, is a group within MTA SZTAKI that focuses its research on cluster and grid technologies. Among its products are the P-GRADE Grid Portal, gUSE, and the SZTAKI Desktop Grid. To date, LPDS has been an active participant in many of the European Grid projects, including Enabling Grids for E-Science in Europe (EGEE) and HunGrid. The LPDS also serves as the Central-European Regional Training Center for EGEE [15].

## 1.5 The P-GRADE Grid Portal

The P-GRADE Grid Portal is a web portal based on GridSphere that allows users to access multiple Grids, among other functionalities. A web portal, according to GridSphere, is a gateway to a collection of services and Portlets [1].

P-GRADE Portal is intended to be a high-level access point for users who wish to submit workflows to Grids without becoming bogged down in the technical details typically associated with Grid computing. Not only does the P-GRADE Portal assist with the execution of workflows, it also assists users in building parallel applications for execution on the Grid. Some of the benefits and features of the P-GRADE Portal are [21]:

- Helping to cope with the large variety of the various grid systems and concepts
- Porting applications between Grid systems
- Porting legacy applications to Grid systems
- Allowing observation of application execution in the Grid
- Tackling performance issues
- Executing Grid applications over several Grids in a transparent way
- Interoperability with Globus Toolkit 2, Globus Toolkit 4, LCG and gLite grid middleware
- Providing Grid authentication
- Built-in graphical editor to design and define grid workflows and grid parameter studies
- Integrated workflow manager
- On-line workflow and job monitoring and visualization facilities
- Multi-grid access mechanism
- MPI execution in Globus and gLite grid environments

- Graphical Grid status checking
- Storage management
- Workflow import-export-archive service

### 1.5.1 Using the P-GRADE Portal

The main function of the P-GRADE Portal is to allow users to create and execute workflows and jobs on the Grid. To that end, a comprehensive graphical Workflow Editor is included. The editor is based on Sun's Java Web Start technology. The editor provides capabilities for creating and editing normal workflows and parameter studies. Users build workflows by adding jobs to the editor and then connecting them according to their file dependencies. Dependencies are represented by small boxes attached to jobs. Green boxes are inputs, while gray boxes are outputs. For instance, in the example below the orange boxes represent jobs and it is clear that the "Invert\_A" and "Multip\_B" jobs depend on "Copy\_A".

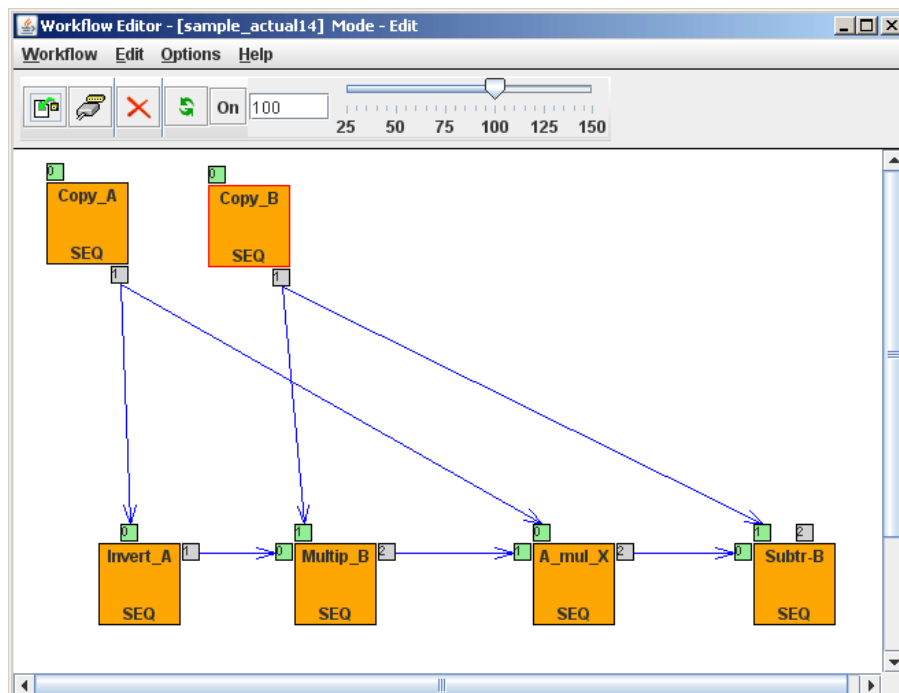


Figure 1.1 - P-GRADE Portal Workflow Editor

In addition, the portal provides a comprehensive certificate management service. After obtaining a certificate from a valid Certificate Authority, a user can upload his or her certificate through the portal and then obtain the proxy necessary to access a specific Grid. The portal will manage the credentials when submitting workflows and jobs so that the user does not have to.

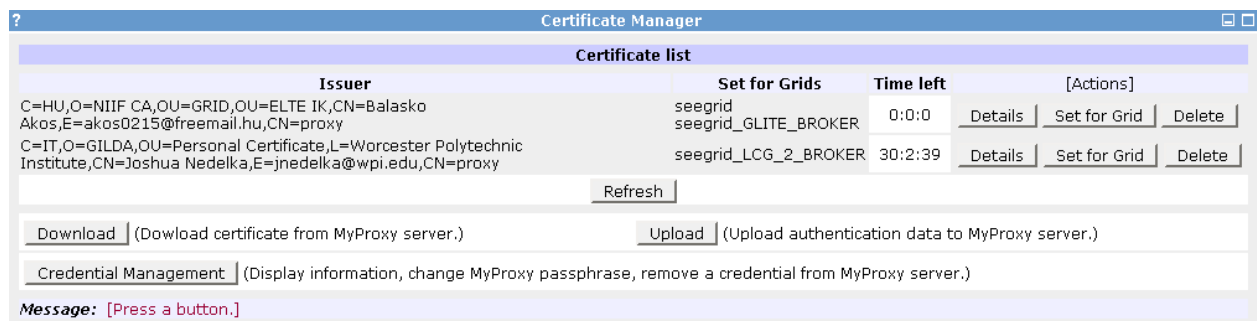


Figure 1.2 - P-Grade Portal Certificate Manager

The portal also provides an information system capable of viewing the statuses of all computing and storage elements on any grid that it is connected to. Results can be filtered according to a specific Virtual Organization or by Grid. The results reported break down each computing element by how many of its processors are currently being utilized. Storage elements simply show how much free space they have available.

Site Name	Sites									
	Computing Element						Storage Element			
	Total	CPU		Running	Job		Space			
	Free	Usage		Waiting	Load	Total	Available	Usage		
aegis01-phy-scl	765	1	100%	788	2465	76%	68.808 TB	68.806 TB	0%	
aegis02-rcub	13	7	46%	6	0	0%	175.498 GB	165.291 GB	6%	
aegis03-elef-leda	3	3	0%	0	0	0%	9.191 GB	8.04 GB	13%	
aegis04-kg	42	37	12%	5	0	0%	871.921 GB	871.811 GB	0%	
aegis05-etfbg	20	20	0%	0	10	100%	40.329 GB	38.473 GB	5%	
alberta-lcg2	140	27	81%	113	35	24%	11.257 TB	11.257 TB	0%	
am-01-iiap-nas-ra	2	2	0%	0	0	0%	1.014 MB	0 B	100%	
amd64.psnc.pl	96	5	95%	9	8	47%	4.675 TB	4.609 TB	1%	
australia-atlas	32	6	81%	26	28	52%	56.027 TB	50.543 TB	10%	
australia-unimelb-lcg2	28	1	96%	27	1	4%	11.217 TB	65.947 GB	99%	

Figure 1.3 - P-GRADE Portal Information System



## **2 Project Statement**

The project that we were given consisted of implementing two features into the P-GRADE Portal: an automatic user account creation system and an email notification system. The two features are common in their purpose to expedite the user's experience with the portal.

### **2.1 User Account Creation System**

In order for people to use an instance of the P-GRADE grid portal, they must first create a user account. The process of creating a new account consists of three high-level tasks: a user requests a user account on one or more portals, a message is sent to the administrators notifying him or her of requests waiting for approval, and finally the administrator creates a user account on each portal requested by the user.

#### **2.1.1 Current System**

Currently, the administrator must conduct most of the account creation process manually. Potential users seeking an account fill out a form on a centralized web server. If the form is filled out correctly, the form then sends an email to the administrator containing the account request information. The administrator then verifies that the request is legitimate, and if so, enters the information into a spreadsheet for later retrieval. For each portal that was requested, the administrator copies the account information into the portal's built-in account creation utility accessible only to administrators. Finally, the administrator sends a pre-formatted email to the user indicating that the account has been successfully created.

#### **2.1.2 Proposed Implementation**

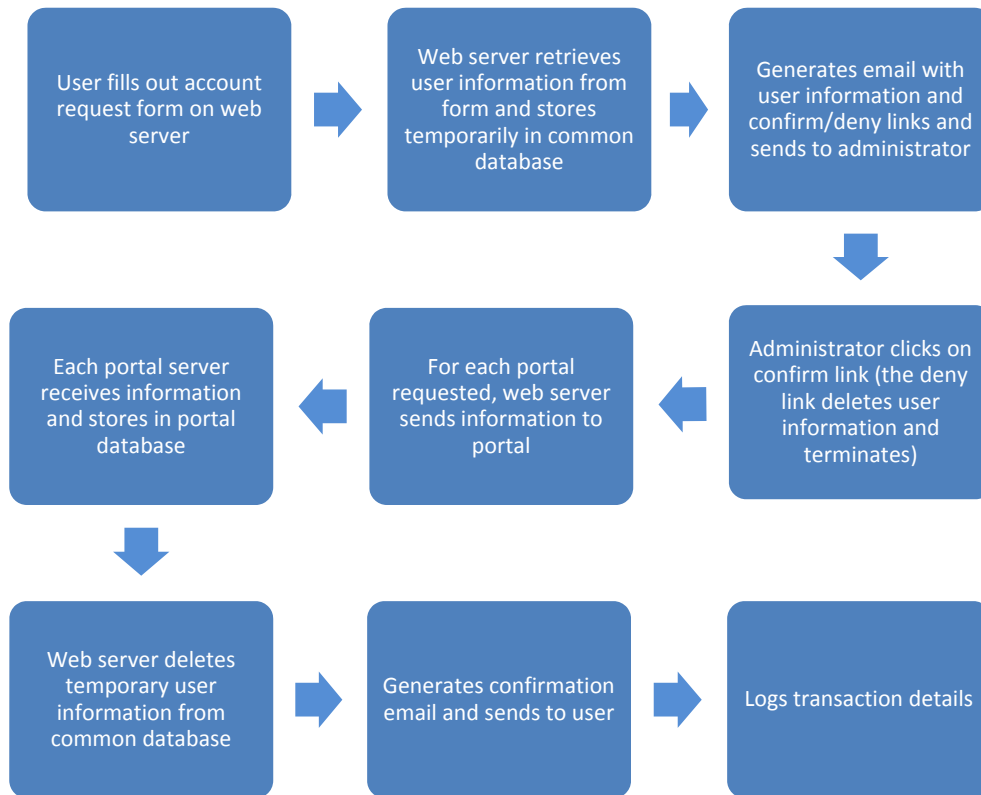
The automated account creation system eases the burden on administrators of processing user account requests. The system accomplishes this goal by automating the majority of the tasks required to create a new user account while retaining an administrator's ability to allow or deny account requests.

The implementation can be broken down into four discrete components:

- extensions to the existing web form to save account requests pending administrator approval,
- a dispatcher to forward account information to each portal requested by the user once an account request is approved,
- a service on each portal to handle account requests and provide error checking, and
- an administrative console to provide a means for an administrator to change settings that affect the account creation process.

The flow of actions that occur when a user wants to create an account is shown in Figure 2.1.

If an action described in the figure does not specify what is performing it, it is the same as the previous action.



**Figure 2.1 - Account creation workflow**

## **2.2 Email Notification of Workflow Status Changes**

The email notification system for workflow status changes is intended to alert users who are not constantly monitoring their workflows that the workflows are indeed being run by the Grid. It can detect and notify any status, from “submitted” to “error” to “finished”.

### **2.2.1 Current System**

In the current implementation of the P-GRADE Portal, there is no functionality for notification of any kind. Users who are interested in determining the status of their workflows and jobs must log into the portal.

Workflow list						
Workflow	Status	Size	Quota (100 Mb)	[ Output ]	[ View ]	[ Action ]
sample_actual1	init	373 KB	0.36%		<a href="#">Details</a>	<a href="#">Submit</a> <a href="#">Attach</a> <a href="#">Delete</a>
sample_actual10	rescue	242 KB	0.24%	N/A	<a href="#">Details</a>	<a href="#">Rescue</a> <a href="#">Abort</a> <a href="#">Attach</a> <a href="#">Delete</a>
sample_actual11	submitted	185 KB	0.18%	N/A	<a href="#">Details</a>	<a href="#">Suspend</a> <a href="#">Abort</a> <a href="#">Attach</a> <a href="#">Delete</a>

**Figure 2.2 - Monitoring workflow statuses in P-GRADE Portal**

This is inconvenient and suboptimal for several reasons. First, while it is possible to know approximately how long a workflow will take to complete, there can also be a very large variance in completion time. A user who believes a workflow will take two hours might be surprised to check the workflow at that time only to see that a few of the total jobs have been run. Conversely, it might be seen that his workflow finished in much less time. In this case, a new workflow could have been started utilizing the same resources as the one that had already finished. This potentially wastes the computing time of the user.

Second, there is no way to tell if a workflow has failed due to an error unless its status is constantly being checked. The same user who checks a workflow after two hours of execution could find that the workflow had stopped running after only a few minutes due to an error in one of its jobs. In this case, a large amount of computing time has been wasted because the user was not made aware that there was a problem in a reasonable amount of time.

### 2.2.2 Proposed Implementation

The proposed solution and implementation to this problem is based on a system that has been developed in cooperation with SZTAKI colleagues for the successor to the P-GRADE Portal, WS-PGRADE. The system provides a flexible infrastructure for notifying users of status changes to their workflows. The current implementation only provides utilities for sending

emails to users, but this could easily be extended to include SMS messages or any other convenient notification.

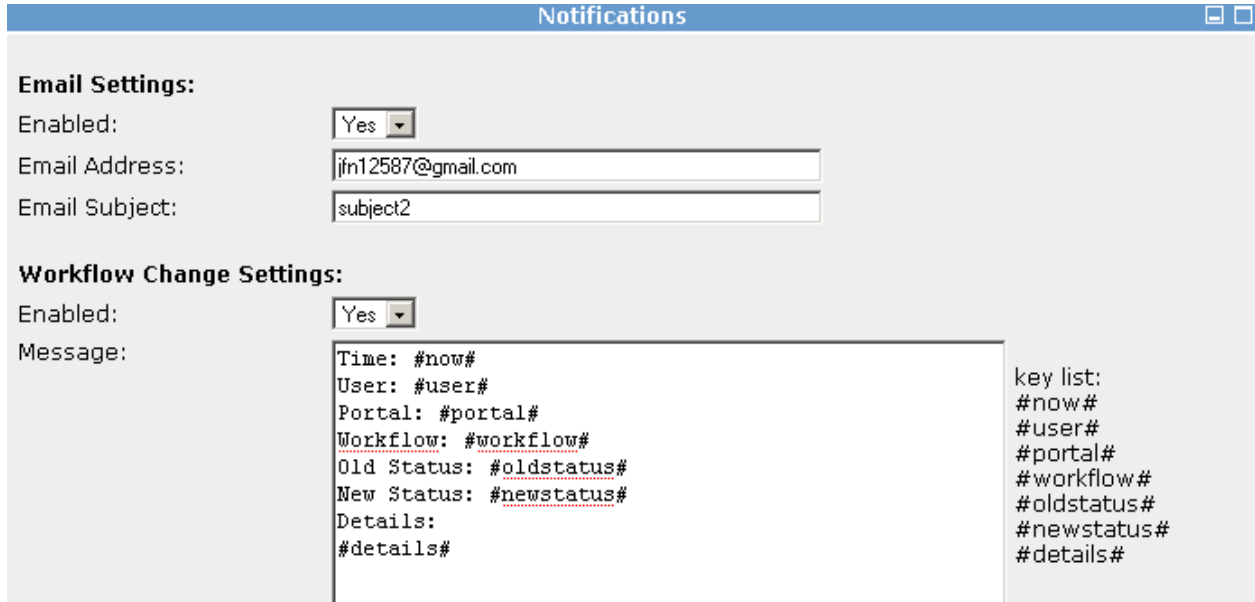


Figure 2.3 - WS-PGRADE Notification Portlet

While the notification system is already robust, it was obviously designed for a system that is vastly different from P-GRADE. The difficulty then is adapting the system to function in an environment that is not necessarily ideally suited to support it. The differences become apparent simply from the names of the two systems: the “WS” in WS-PGRADE stands for Web Services. While the original P-GRADE portal is a monolithic system – that is, its individual components cannot stand alone because of how tightly coupled they are – the new WS-PGRADE is more flexible when it comes to integrating new features. It is much easier to capture the events that are necessary to issue workflow status change notifications.

Without access to the services provided by WS-PGRADE, it becomes necessary to leverage other utilities that the P-GRADE Portal provides. There are several locations that the notification system must be attached to in order to accurately report status changes. By

integrating the system into these key places, the P-GRADE notification system can completely represent all of the possible workflow statuses and also allow users to select the level of notifications that they would like to receive: on any status change, only on workflow completion, or never.

## **3 Methodology**

### **3.1 User Account Creation System**

Implementing the user-account creation system required several steps. The first step was to gather requirements for the system. Based on the requirements, three possible implementations were conceived. Several simple prototypes of these implementations were developed to assess their feasibility. The implementations were then presented to colleagues at SZTAKI in order for them to discuss the merits of each proposed implementation and decide which would best suit their needs. Based on their feedback, a single implementation was selected for further development.

#### **3.1.1 Requirements**

The automated account creation system went through several stages of requirement gathering and prototypes before the design was solidified in its final form. The initial requirement was to reduce the workload of the administrator by performing automatically as many steps of the process as possible. Additional requirements added later are as follows.

- The administrator must retain full control over who is allowed accounts.
  - This is accomplished by providing a mechanism by which the administrator can approve or deny accounts before they are created.
- Unauthorized users should not be able to create accounts without administrator approval, nor should they be able to break into the system to wreak havoc.
  - The security of the account creation system is an important requirement; an insecure system cannot be put into production, as it would place the grid portals in danger of being compromised by an attacker.

- Encryption of sensitive data, storing only hashed or encrypted passwords in databases, and requiring both administrators and servers to authenticate with other servers can minimize the chance of secret information falling into the wrong hands.
- A user should not be able to initiate a modification to a portal server's database without prior administrator approval.
  - Account requests are stored in the web server's database until approved or denied by an administrator. If approved, they are then sent to the portal servers for further processing.
- A user should not be able to spam the administrator by sending large numbers of account requests.
  - Requiring the user to enter a confirmation code displayed in an image before submitting the account creation form prevents bots from quickly sending requests. Although no amount of noise in an image can make it readable by a human but not by a computer, the amount of time required for a computer to process the image is significant enough to prevent spamming.
- The administrator should be able to retrieve information on all registered users in order to send mass emails, etc.
  - A web page allows the administrator to download user information, including email addresses, from one or more portals.
- The account creation system should be able to adapt to various portal configurations.



- An administration console allows the administrator to configure information about the portals on which accounts can be created and change other settings related to the account creation system.

### **3.1.2 Implementation Considerations**

Based on the original requirements provided by SZTAKI, multiple possible implementations of the account creation system emerged. The differences between the various implementations revolved mainly around which tasks are performed by the web server and which are performed by the portal servers.

#### **3.1.2.1 *Mostly on Web Server***

In the first implementation, most of the work of creating a user account is done on the web server and only a small amount is done on each portal server. On the web server are an account request form, an automatic account creator, and a setting manager. On each portal server is an account creator service. Attached to each portal server and the web server are databases for persisting account data and settings. A diagram of the components and the flow of data between them is shown in Figure 3.1. The sequence of events is as follows:

1. A user seeking an account on one or more grid portals fills out the account request form on the web server.
2. The web server retrieves the request information from the form and verifies it with the account creator for each portal listed in the request.
3. The web server stores the request information in the common database for later retrieval.
4. The web server sends an email to the administrator with links to accept or deny the account request.
5. The administrator clicks on one of the links in the email.

- The 'accept' link causes the automatic account creator to accept the request as it is.
  - The 'accept with default roles and groups' link causes the automatic account creator to accept the request, but with the requested roles and groups reset to their default values.
  - The 'deny' link causes the automatic account creator to delete the account request and send the user an email indicating that the account request has been denied.
6. If the administrator accepts the account request, the automatic account creator deletes the request information from the common database and logs the user name, email address, and grid names so that they will be available for future reference.
  7. The request information is dispatched to the account creator of each portal listed in the request.
  8. The account creator on each portal saves the account information in the portal database.
  9. The automatic account creator sends an email to the user indicating that the account request has been accepted.

Separately, the administrator interacts with the setting manager via the administration console.

The setting manager then saves the updated settings in the common database.

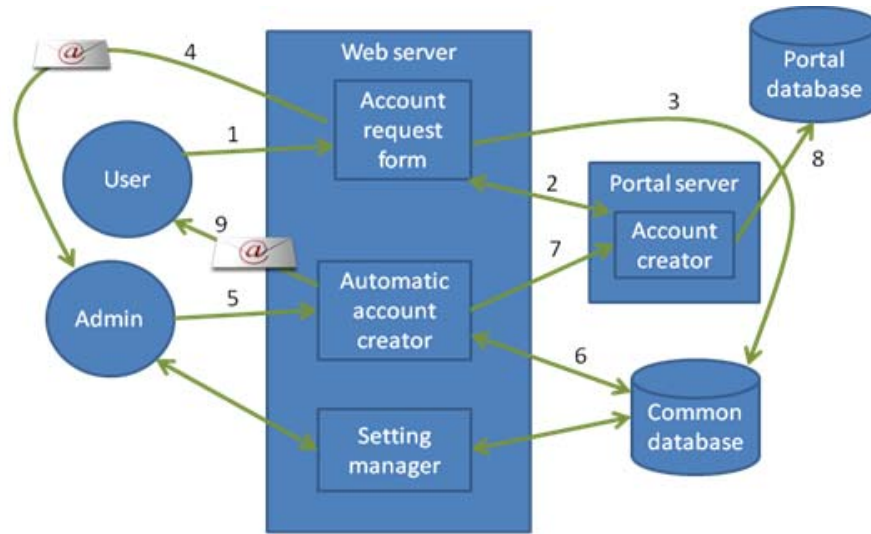


Figure 3.1 - Web server implementation data flow

### 3.1.2.2 *Combination of Web Server and Portal Servers*

In the second implementation, only the account request form is on the web server and the rest of the components are on each portal server. Instead of having the account creation components split over the web server and the portal servers, there is a single automatic account creator for each portal server. A diagram of the components and the flow of data between them is shown in Figure 3.2. The sequence of events is as follows:

1. A user seeking an account on one or more grid portals fills out the account request form on the web server.
2. The web server retrieves the request information from the form and sends it to the automatic account creator for each portal listed in the request.
3. The portal servers store the request information in the portal databases for later retrieval.
4. Each portal server sends an email to the administrator with links to accept or deny the account request.

5. The administrator clicks on one of the links in each email. Since the account creation process for each portal is independent at this point, the administrator may choose to accept the request for some portals by deny it for others.
6. If the administrator accepts the account request for a given portal, the automatic account creator deletes the temporary account request information from the portal database and creates a new account on the portal using the information.
7. The automatic account creator saves the user name, password, and grid name to the common database for future reference. If the information was already added to the database by a different portal, the grid name list in the database is updated to reflect the additional grid.
8. The automatic account creator sends an email to the user indicating that the account request has been accepted.

The setting manager for this implementation is similar to that of the previous implementation, except that there is a separate setting manager and administration console for each portal server.

The settings for each portal are kept synchronized by virtue of residing in the same database.

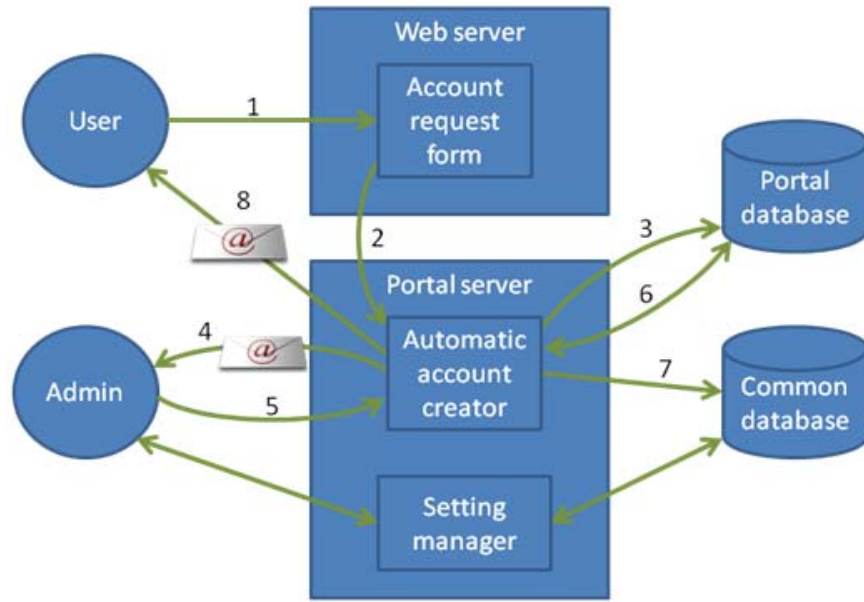


Figure 3.2 - Combination implementation data flow

### 3.1.2.3 *Entirely on Portal Servers*

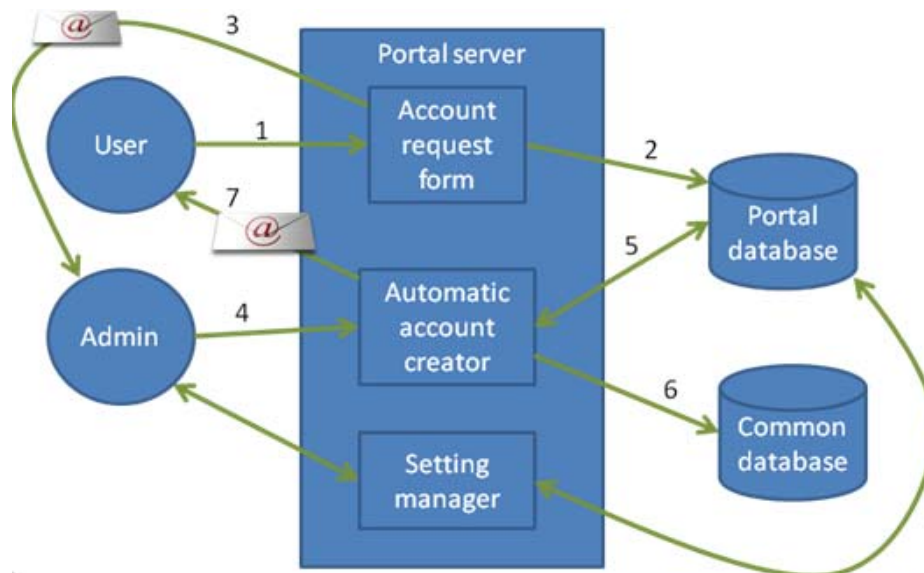
In the third implementation, the entire account creation process is replicated across each portal server. The only common component is the database where user names and email addresses are stored. Shown in Figure 3.3 is a diagram of the components and the flow of data between them.

The sequence of events is as follows:

1. A user seeking an account on one or more grid portals fills out an account request form for each portal desired. The rest of the process is the replicated for each portal.
2. The portal server retrieves the request information from the form and stores it in the portal database for later retrieval.
3. The portal server sends an email to the administrator with links to accept or deny the account request.
4. The administrator clicks on one of the links in the email.

5. If the administrator accepts the account request, the automatic account creator deletes the temporary account request information from the portal database and creates a new account on the portal using the information.
6. The automatic account creator saves the user name, password, and grid name to the common database for future reference. If the information was already added to the database by a different portal, the grid name list in the database is updated to reflect the additional grid.
7. The automatic account creator sends an email to the user indicating that the account request has been accepted.

The setting manager for this implementation is different from previous implementations in that the settings are stored in the portal database instead of the common database. As a result, the settings must be updated individually for each portal. The purpose of this design is that the individual portals are kept as independent of each other as possible.



**Figure 3.3 - Portal server implementation data flow**

### 3.1.2.4 *Comparison of Implementations*

Each of the possible implementations of the account creation system has benefits and drawbacks.

Table 3.1 shows a comparison of the three implementations described earlier.

Implementation	Benefits	Drawbacks
Mostly on web server	<ul style="list-style-type: none"> <li>• The administrator only receives a single email per account request.</li> <li>• A user may be able to request accounts on multiple portals with a single request.</li> <li>• The portal servers are protected from flooding attacks because they do not become involved in the account creation process until the administrator has approved a given account request.</li> </ul>	<ul style="list-style-type: none"> <li>• Since most of the functionality is in the web server, the account creator cannot take advantage of the existing portal environment.</li> <li>• Communications between the web server and the portal server add complexity to the system.</li> </ul>
Combination	<ul style="list-style-type: none"> <li>• Uses existing code on the web server for handling account requests while also taking advantage of the portal environment.</li> <li>• A user may be able to request accounts on multiple portals with a single request.</li> </ul>	<ul style="list-style-type: none"> <li>• Communications between the web server and the portal server add complexity to the system.</li> <li>• Each account request generates multiple emails to the administrator.</li> <li>• The portal servers are more vulnerable to flooding attacks because requests are stored on them before being sent to the administrator for verification.</li> </ul>
Entirely on portal servers	<ul style="list-style-type: none"> <li>• Takes advantage of the existing portal environment and account creation code already written for the portal.</li> <li>• Each portal is kept as independent as possible so that they may be upgraded separately.</li> </ul>	<ul style="list-style-type: none"> <li>• A user must submit a separate account request for each portal, generating a separate email to the administrator for each request.</li> <li>• Since each request may contain different information, the portals must be able to reconcile conflicts when updating the common database.</li> <li>• The portal servers are more vulnerable to flooding attacks because requests are stored on them before being sent to the administrator for verification.</li> </ul>

**Table 3.1 - Comparison of account-creation system implementations**

After all of the benefits and drawbacks were taken into consideration, the first implementation, in which most of the account creation process is done on the web server, was chosen. The benefits of a single email to the administrator per account request and protection of the portal servers against flooding attacks outweigh the drawbacks of additional complexity and the inability to reuse significant amounts of code.

### **3.1.3 Technologies Used**

Implementing the account creation system required the use of multiple technologies. The web server uses PHP: Hypertext Preprocessor (PHP) to generate Extensible Hypertext Markup Language (XHTML) and sends it to the user's web browser to be displayed. The web server also employs a confirmation system written by the phpBB Group in order to reduce spamming of account requests. The portal server, written in Java, uses servlets to process requests for information and account creation from the web server. Technologies specific to the account creation system are listed below; those common to both the account creation system and the email notification system are described in Section 3.3.

#### **3.1.3.1 *PHP: Hypertext Preprocessor***

PHP is a server-side scripting language used to transform documents requested by a web server. Any text-based document type may contain PHP tags. When a user requests a document of a type configured to use PHP, any PHP tags inside the document are executed and their output is merged with the rest of the document. Like most server-side scripting languages, PHP empowers websites to support a wide range of features such as session support, customized pages and database connectivity.



### 3.1.3.2 *phpBB Confirmation System*

The confirmation system used in phpBB uses non-solid characters, random colors and character rotation, and customizable foreground and background noise to increase the difficulty of automated character recognition. This system is advantageous compared to others because it is free (released under the General Public License (GPL)) and because it was written in PHP and thus did not need to be ported to a different language in order to be integrated into the account request website. Examples of the images generated by the confirmation system, both with and without foreground noise, are shown in Figure 3.4.

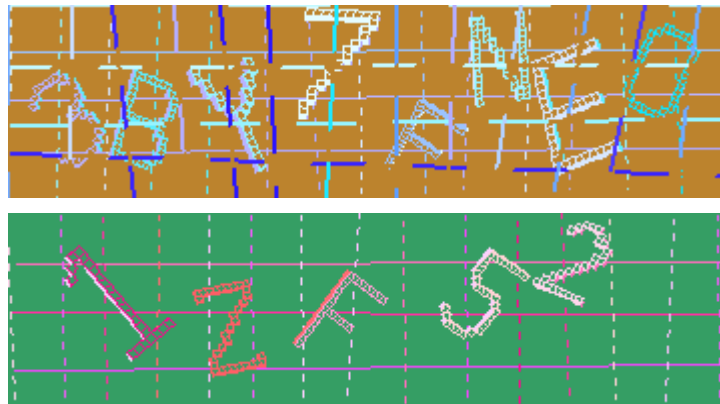


Figure 3.4 - Confirmation code images

## 3.2 Email Notification System

### 3.2.1 Requirements

As with any engineering project, the first step was to gather and understand all requirements to make sure that a functionally complete and useful system was designed. A number of people in the LPDS were able to provide input regarding the functionality that was necessary for a successful implementation. Requirements will be categorized as either “implementation” or “integration”, so requirements that are specific to the integration and not development of the notification system will be marked as such. The requirements are:

- Create a flexible set of notification utilities (implementation)
  - This requires creating portal-independent functionality to handle notification events once they have been sent from the system. This includes starting any plugins that will actually generate the notifications and handling the logic to determine if a notification should actually be sent.
- There should be a way to add notification plugins (implementation)
  - Through a simple interface, the system provides the flexibility necessary to add additional plugins (i.e. SMS notifications).
- There should be a way for the user to globally disable notifications (integration)
  - This must be an overriding property for all the other plugins. It provides an easy way for users to turn notifications on and off, as they need them. Most importantly, the system must persist and use this value.
- There should be a way for the user to disable a single plugin (implementation)
  - This switch will be the second in the enable/disable hierarchy (after the global one). It should allow users to turn off a single notify plugin (Email, SMS, etc.) based on their current needs.
- There should be a way for the user to specify the contact information necessary for each plugin (implementation/integration)
  - Since each plugin notifies in a unique way, each plugin must have its own set of properties to allow it to function. For example, an Email plugin needs an address, while an SMS plugin needs a valid mobile phone number.
- There should be a way for the user to format the notification to be received (implementation)

- Each plugin should be able to send its own uniquely formatted message based on the destination of the notification and the user's needs. The system should provide various pieces of information (statuses, time stamp, etc.) that the user can include in the messages, if they choose.
- There should be a way for the user to specify when to be notified for a given workflow (integration)
  - This final switch is third in the enable/disable hierarchy (after plugin and global). It should allow a user to turn off notifications for a specific workflow when it is being submitted. However, it should also allow the user to specify the granularity with which they would like to be notified (i.e. on completion of a workflow, on any status change, etc.)
- The system should accurately reflect the calculated status shown in the portal (integration)
  - For this to be possible, the system needs to receive notifications of status changes in real time which means it must plug into the portal at several key points. It is also necessary to detect some statuses that are not immediately apparent.
- The system should handle rescue/restart of a workflow correctly (integration)
  - Whatever data files the system creates must be updated when a user intervenes and changes the normal running pattern of a workflow. The system must reinitialize this data to prevent incorrect statuses from being sent.
- The system should log all actions taken (integration)
  - For administrative and debugging reasons, all actions taken by the system should be written either to a specific or global log file.

### 3.2.2 Implementation Considerations

After investigating the innards of the portal and discussing them with members of the LPDS, two different solutions seemed possible: status change detection by either polling or detection through events.

#### 3.2.2.1 Polling

At its core, polling involves occasionally checking a piece of data to see if it has changed, and then taking the appropriate action when it has. Typically, this is implemented by starting a separate thread to monitor the data in question. The thread will track the last status it reads, check for new statuses, and then sleep. For status change notifications, the system can be shown with the following diagram.

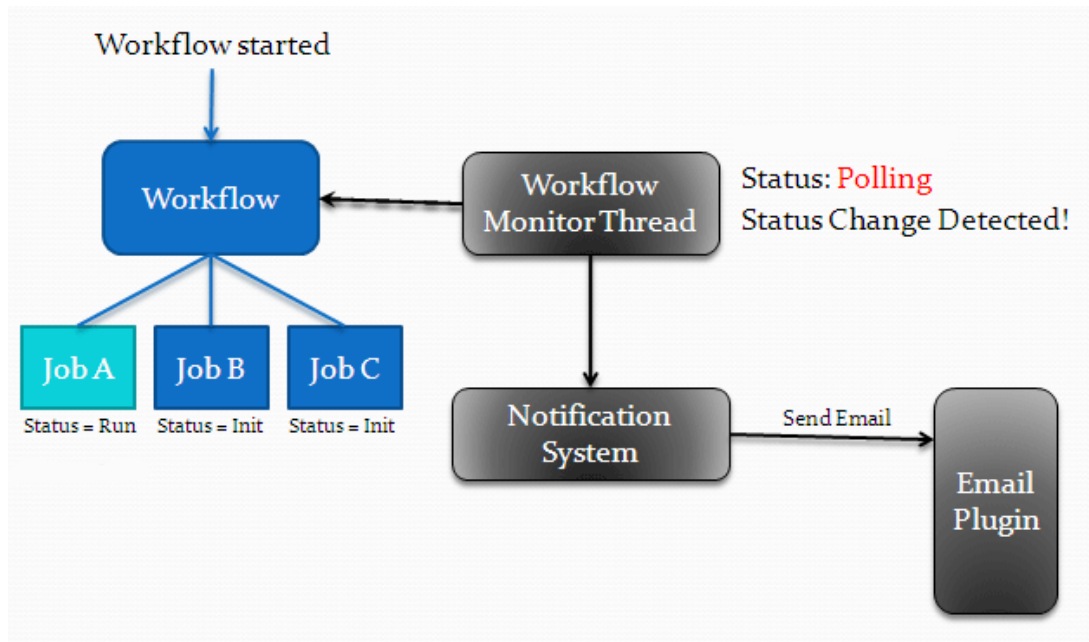


Figure 3.5 - Polling Flow of Events

In this case, when a job status changes, this can be reflected in a change in the entire workflow status. The monitor thread polls for this, and if it detects a change then it initializes the notification system that handles actually notifying the user.

### 3.2.2.2 Event Driven

An event driven system is slightly less complicated because it does not need to first determine if it should act; rather it knows that simply by being invoked the system is telling it that in fact a change has occurred that it needs to respond to. The event driven system would have to receive either job or workflow statuses from another location in the system, and it could then store and process them in similar ways to the polling system. The event driven system is reflected below.

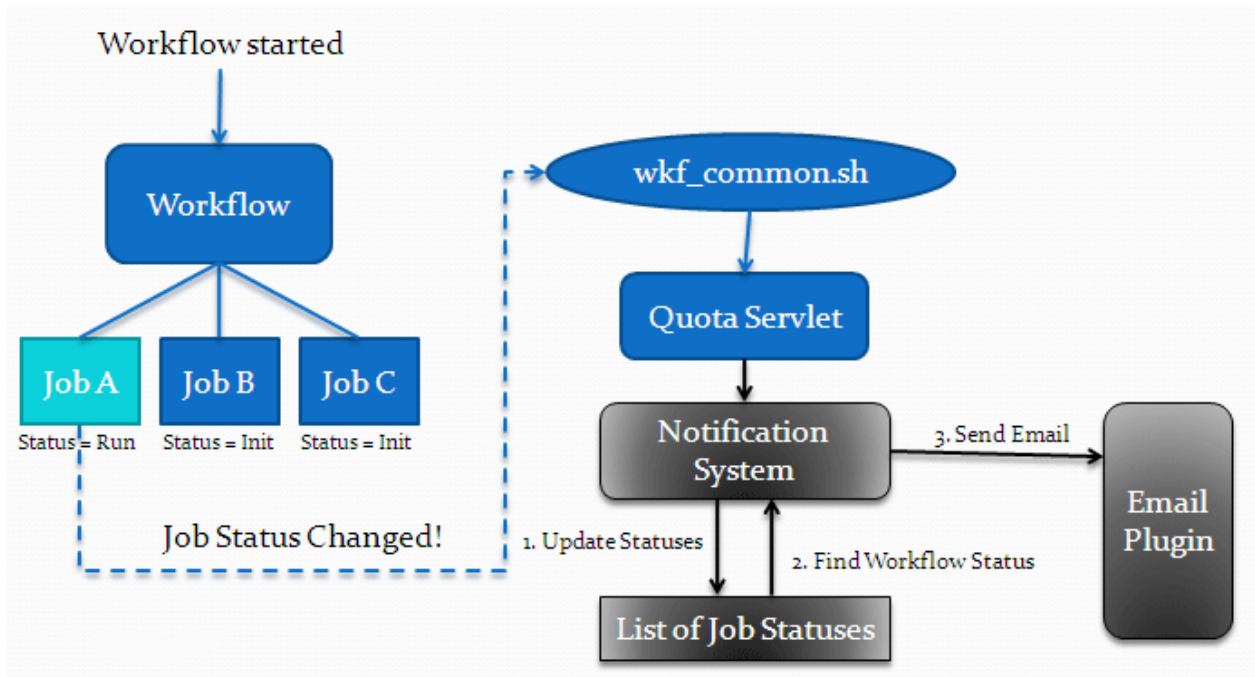


Figure 3.6 - Event Driven Flow

This diagram reflects some of the more implementation specific portions of the system, but the principles remain the same. Instead of reading that a job status has changed, this system actually receives an event indicating that it has happened. This event is handled and eventually initializes the notification system like in the polling example. The notification system stores the status changes it has read in a different way than polling, and then sends an email.

### 3.2.3 Analysis of Implementation

Each system presents its own set of advantages and disadvantages. The polling system seems to integrate more readily into the existing P-GRADE Portal while remaining loosely coupled. It leverages the current utilities built into the portal (like getting workflow statuses) and just adds another level on top of them. Moreover, since it runs completely separately from the workflow, it is less likely to interfere with current code. However, polling does have some serious drawbacks. First, it is very expensive in terms of resource usage compared to the event driven system. Starting a separate thread for each workflow has the potential to put a very large burden on the server, and it is something that would have to be tested extensively. The LPDS thought it possible that hundreds of workflows might be executing at any one given time. Additionally, the polling system also has a tradeoff with the accuracy of its reporting. Because it is not polling constantly (it polls at random intervals to avoid monopolizing the processor), it can never be as accurate as the event driven system. It will never be off by more than a few seconds, but this could be important. Another major disadvantage appears when considering the premature termination of a workflow. The thread must be initialized every time a workflow is submitted and unfortunately, because of recovery from an error state, there are many places that submission can happen within the code. This means that the monitor thread will be tightly coupled with the workflow submission process.

The event driven system manages to avoid many of the problems plaguing the polling system. It eliminates the excess overhead by only running when an event is actually fired. This also allows it to report with a greater degree of accuracy and eliminates the problem with rescuing and restarting workflows. Since the notification system would only run when invoked, the event driven system must make sure that all data it calculates is stored in a more permanent way than as a variable in a class. If a workflow enters into a “rescue” status the data remains even when

the workflow is restarted. In practice it is slightly more complicated, though certainly not as much so as restarting a workflow when polling. The problems with the event driven system are mostly related to coupling. In order to receive the events it needs to accurately compute statuses, the system needs to be invoked from several completely different locations in the portal. Additionally, since it will be using statuses that it receives and stores, there is going to be some code overlap with existing utilities for calculating workflow statuses.

While there are certainly tradeoffs with each system, the advantages of the event driven system seem to win out against the polling system. The main concern is that some of the problems with the polling system could bring the server to a crawl. A problem this serious cannot be offset by saying the system is more loosely coupled. After discussing the potential solutions and analysis with the rest of the lab team, the consensus was that an event driven system would work quite well for implementing the feature.

### **3.2.4 Technologies Used**

What follows is a brief overview of the technologies that were used exclusively by the notification system. First, an introduction of the technology will be given followed by its uses in the system.

#### **3.2.4.1 Tomcat**

Apache Tomcat is an application server and servlet container. It is used to host and serve JavaServer Pages (JSP). Tomcat is an open source project distributed under the Apache Software License.



Initially developed as a reference servlet implementation by Sun Microsystems, Tomcat was eventually donated to the Apache Software Foundation for further development [5].

Tomcat is used as the application server to host a number of web applications on the portal server. The two main applications hosted are GridSphere (the actual portal) and the Grid portal itself.

### 3.2.4.2 *GridSphere*

GridSphere is a portlet container based



on the JSR 168 portlet API standard. It allows for the easy integration of new portlets into the system, thereby extending its functionality [1].

On the portal server, GridSphere is used to expose all of the functionality of the portal to the user. Using a vast array of portlets, functionality that would otherwise only be available over the command line is delivered via a web interface. GridSphere also provides some of the portlets itself, including the user authentication portlet.

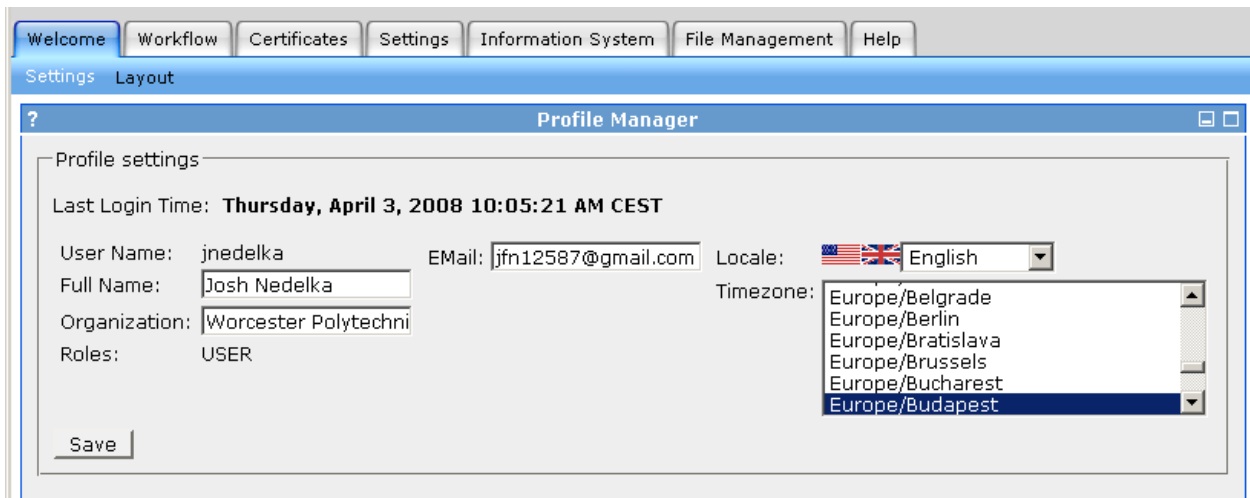


Figure 3.7 - The P-GRADE Portal Running Within GridSphere

## 3.3 Common Technologies Used

### 3.3.1 Java



Java is a high-level, object oriented programming language developed by Sun Microsystems [2]. Originally developed as a language for use on Sun's \*7 multimedia controller, Java soon found itself more at home with internet applications. In stark contrast to the completely static and non-interactive content that existed on the Internet at the time, Java provided the ability to add user controllable animations [7]. Currently in its sixth major version, Java continues to be a popular development platform, enabling developers to [16]:



- Write software on one platform and run it on practically any other platform
- Create programs to run within a web browser and web services
- Develop server-side applications for online forums, stores, polls, HTML forms processing, etc.
- Combine Java technology-based applications or services to create highly customized applications or services
- Write powerful and efficient applications for mobile phones, remote processors, low-cost consumer products, etc.

Java has proved to be so useful because of the goals the developers had when creating it:

- Simple, Object Oriented and Familiar
  - When dealing only with the core functionality of Java, it is in fact a relatively simple language that can be understood by someone with limited programming experience. This in conjunction with its object-oriented design makes it a powerful tool for beginners. In addition, the developers of the language decided

to base it around C++ syntax, while abandoning some of the complications that C++ carried. This made it more accessible to already experienced programmers.

- Robust and Secure
  - With extensive error checking and built-in memory management, Java avoids many of the problems typically found in C and C++. Errors in Java code are more frequently logic based, rather than a result of unexpected memory allocation behavior. In addition, the Java Virtual Machine (JVM) protects code from malicious attacks.
- Architecture Neutral and Portable
  - Java code is not directly compiled; rather it is first compiled into an intermediate bytecode. This bytecode can be executed on any machine for which there exists a JVM – the architecture specific implementation that allows Java to be executed.
- High Performance
  - While not as fast as compiled code, Java makes up for this by allowing its interpreter to run without needing to run-time check the environment. In addition, when speed is absolutely an issue, Java can interface with native compiled code.
- Interpreted, Threaded and Dynamic
  - As with any modern language, Java is multi-threaded, allowing it to run multiple applications simultaneously. Its dynamic nature comes from its class linking (since the language itself is statically typed). Java can dynamically link in any library, allowing for greater flexibility when delivering distributed and networked applications.

The core portal is built entirely in Java, including many other Java technologies (see below) [28].

### 3.3.1.1 JSP & Servlets

JavaServer Pages (JSP) technology allows easy integration of dynamic and static web content. Much like Java, JSP is platform independent and can be deployed using a number of servers including Apache Web Server or Internet Information Services with third party servlet containers and WebSphere, GlassFish, Tomcat and others. JSP is designed so that the presentation layer is separated from the content generation, enabling someone without extensive knowledge of Java to easily change presentation without affecting the content. JSP allows for embedded Java using scriptlets and provides its own expression language: the JavaServer Pages Standard Tag Library (JSTL) [13].

JSP is an extension of Java Servlet Technology. In many cases, JSP is directly compiled into a servlet for execution on the server. However, the more powerful combination of JSP and servlets comes from the so-called Model 2 architecture. While Model 1 architecture uses the JSP page to handle both request and response, Model 2 separates this and moves all logic into a servlet that handles the request. The job of the JSP page then becomes simply to render the content; all logic and processing is removed [26].

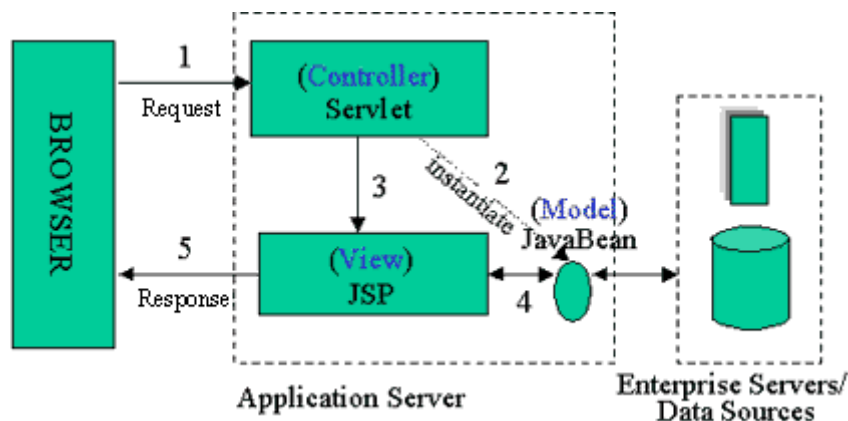


Figure 3.8 - JSP Model 2 Architecture [26]

### 3.3.1.2 *Portals & Portlets*

At their core, portlets are “web components -like Servlets- specifically designed to be aggregated in the context of a composite (portal) page” [10]. Put more simply, a portlet is an application designed to produce a fragment of the markup necessary to render a page. It is only responsible for the rendering of its own small view, and the portal page handles joining the various portals together to form a single web page.

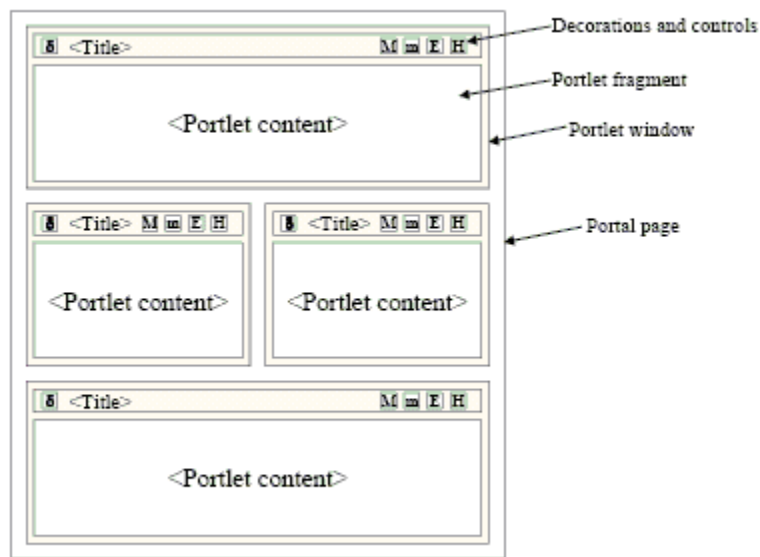


Figure 3.9 - Elements of a Portal Page [23]

The portal however, is responsible for more than simply content and portlet aggregation. The portal also serves to control the look and feel of the view presented to the user – the menus, styles, etc. In terms of functionality, the portal also typically provides a set of common services that can be applied to all portlets. Examples include a common login, where a user can login to the portal and then access all of its portlets, and personalization, where a user can change style settings and even which portlets they want displayed on their page.

Since the P-GRADE Portal is just that – a portal – it follows that all user interface development comes in the form of portlets. A number of portlets are necessary for the features being implemented. Primarily these portlets will be used for editing settings.

### 3.3.1.3 *NetBeans*

The NetBeans Integrated Development Environment (IDE) is a



free, open-source tool for working with a variety of languages and technologies, including Java, C/C++, Ruby, etc. [20]. Originally developed by Sun Microsystems, the IDE was made open-source in 2000 to further its development and bring in more community support [29].

For the purposes of this project, NetBeans provides the best support for the features that are needed. It is integrated tightly with Java Enterprise Edition (J2EE) and allows the entire P-GRADE Portal to be hosted locally, if necessary [11]. It also provides all of the features expected in a modern IDE, including auto-completion, automated building and support for multiple technologies (in our case, JSP, Java, and HTML, just to name a few).

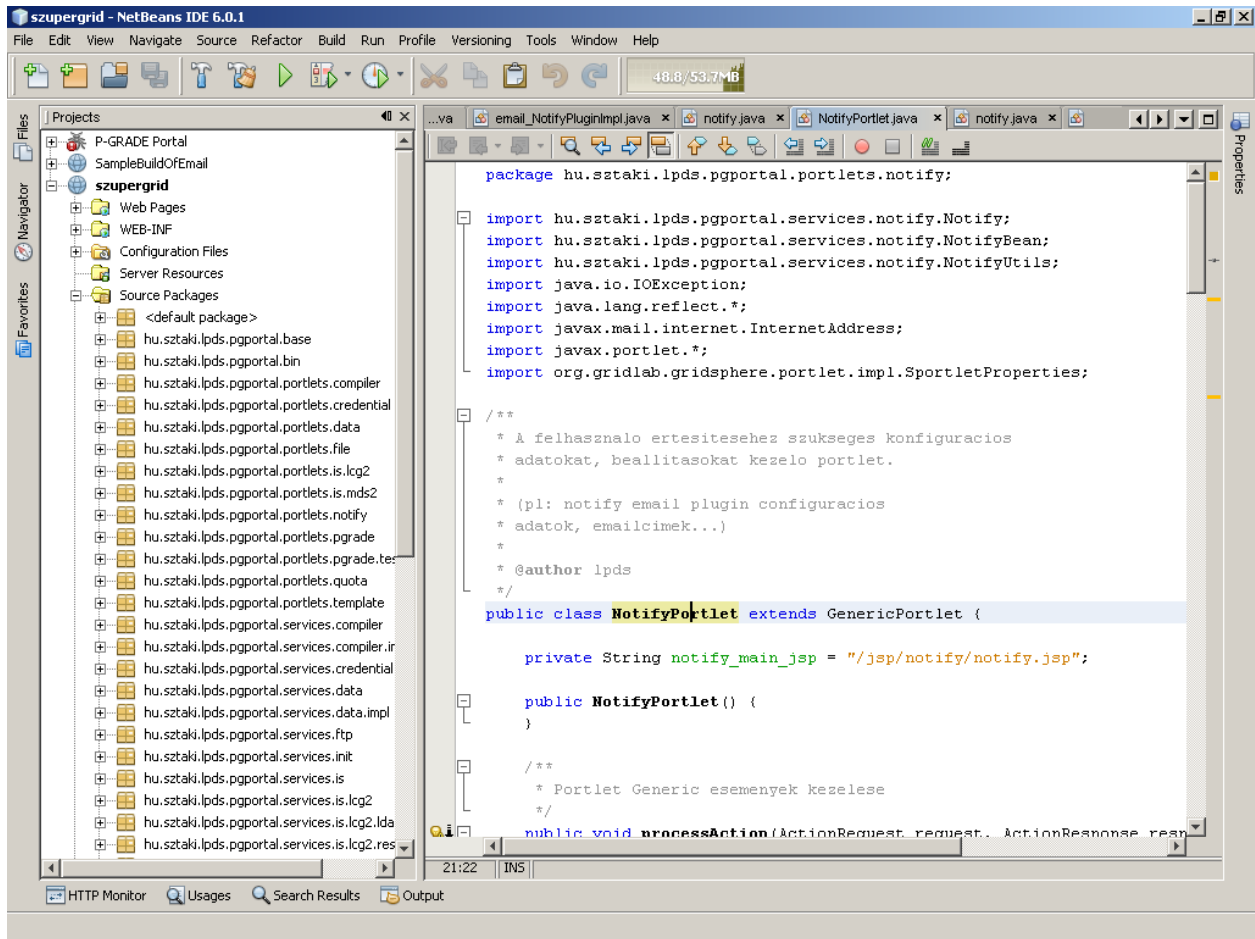


Figure 3.10 - The NetBeans IDE

### 3.3.2 XML

Extensible Markup Language (XML), developed in the late 1990s, is



a specification for writing custom markup languages. XML is used primarily to store and transfer structured data. Ready access to standardized XML libraries on most platforms allows XML to be used to transfer data between many platforms, thus significantly increasing the potential for interoperability[3].

For this project, XML is used to transfer data between the portal servers and the web server, particularly for the exportation of user information.

### 3.3.3 XHTML

Extensible Hypertext Markup Language (XHTML) is a document model based



on Hypertext Markup Language (HTML) that is conformant to XML standards. As such, all valid XHTML documents are also valid XML documents and thus can be parsed using standard XML libraries.

XHTML and HTML documents are rendered by web browsers and are used by the majority of websites to display textual and graphical content. HTML may be sent unmodified by a web server or generated on the fly by server-side scripts. Although originally designed with static content in mind, HTML has been extended to support dynamic content via JavaScript and other client-side scripting languages.

For this project, the user interfaces for both the account creation system and the notification system are entirely written in XHTML. The account creation system uses PHP running on a web server to generate XHTML, while the notification system uses JSP running on an application server.

### 3.3.4 JavaScript

JavaScript is a scripting language typically used by web browsers to enable client-side scripting within HTML pages. Written by Brendan Eich[30], JavaScript was first released as part of the Netscape web browser in December 1995. JavaScript, although named after Java, is a fundamentally different language. JavaScript has dynamic typing, weakly typed variables, and prototype-based classes. Functions are first-class, which means that they may be manipulated like normal objects and can be called dynamically. These features make JavaScript more accessible to non-programmers than typical programming languages[12].

For this project, JavaScript is used to display a form when submitting a workflow and to provide dynamic checking of fields in the account request form before they are sent to the web server. Additionally, JavaScript is used in the administrator manual to give users the ability to expand or collapse individual sections of the manual.



## 4 Implementation

### 4.1 User Account Creation System

The account creation system consists of several components on multiple servers. Shown in Figure 4.1 are the most significant components that make up the account creation system, connected by data flow. The blue components reside on the web server and are implemented using PHP scripts, and the green components are part of the portal server and are implemented as Java servlets and services. The arrows indicate the direction data flows between the various components.

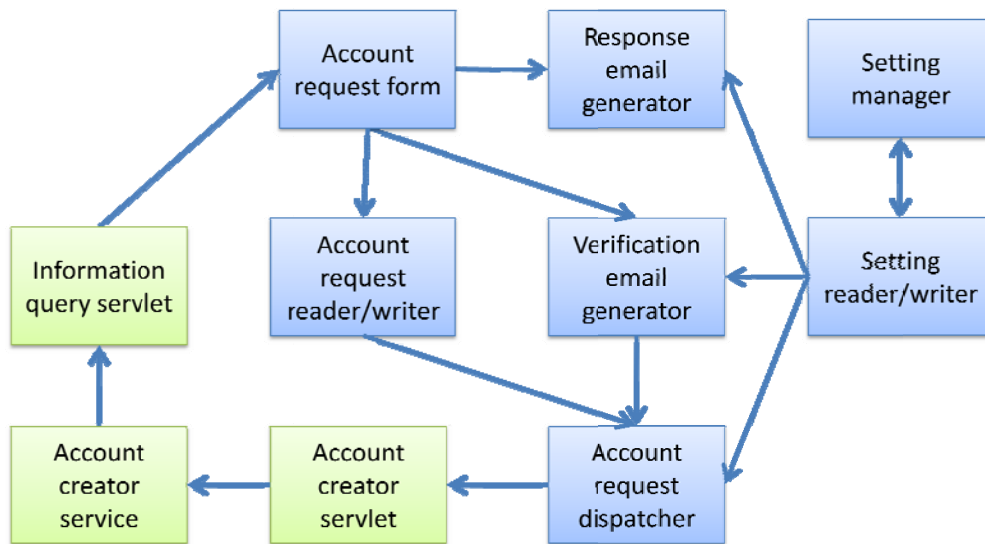


Figure 4.1 - User account creation components

The account creation system will work with a wide variety of server topologies. Described in Figure 4.2 is one possible topology; other configurations will work as well. In the topology shown, each server is in a separate physical location. There is a single web server hosting a common account creation website connected to one or more servers running P-GRADE portal. The web server and portal servers each have their own database, which can be on either the same computer or a separate computer. For additional security, each web server is located behind a firewall and only the specific ports needed are left open.

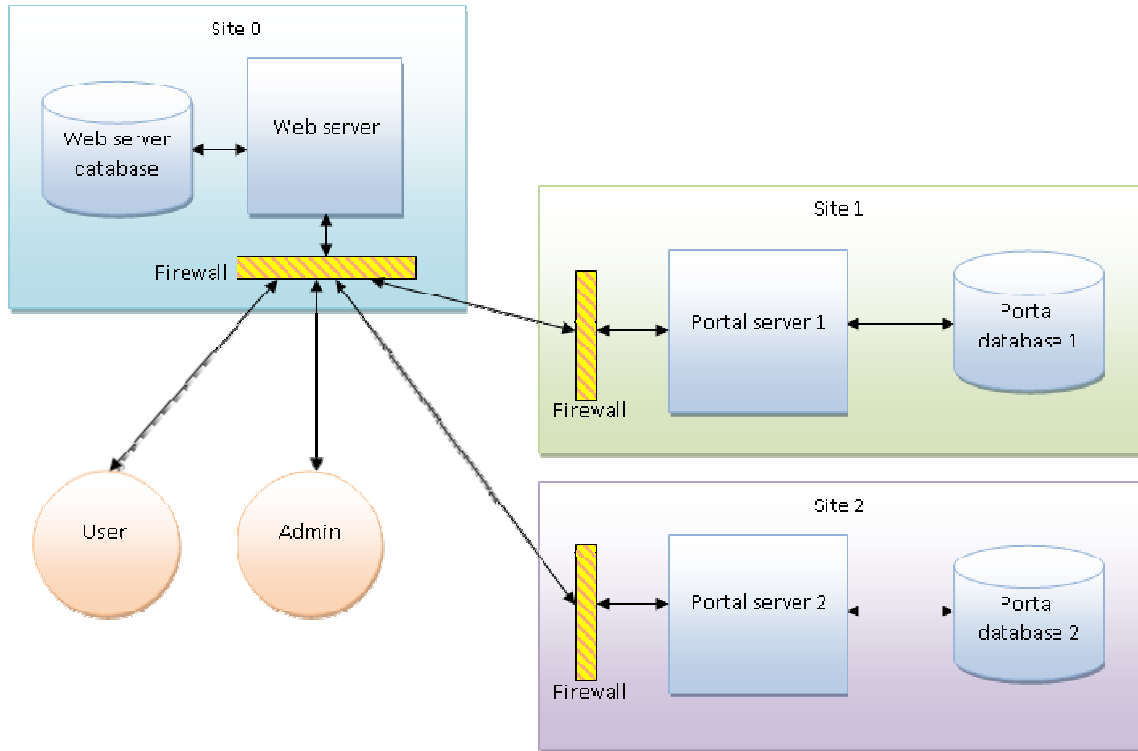


Figure 4.2 - Account creation system server topology

#### 4.1.1 Portal Server Interface

The portal server is implemented as two web applications, GridSphere and Szupergrid, running on a Java application server such as Apache Tomcat. So that the web server could interface with the portal server to submit account requests and retrieve information, several portlets were created along with services that implement account creation and authentication. The servlets and services are described in the following sections.

##### 4.1.1.1 Account creator servlet

The account creator servlet processes requests from the web server to create new user accounts and to verify account request information. The servlet uses the authentication service to check

that the web server has the correct permissions and then calls methods exposed by the account creator service to do the actual work of creating or verifying a user account.

#### 4.1.1.2 ***Information query servlet***

The information query servlet processes requests from the web server to retrieve information pertaining to user accounts. After checking permissions with the authentication service, the servlet retrieves the appropriate information and outputs it in an XML format that can be read by the web server. The following information may be requested:

- All of the roles and groups that a user may request to join as part of an account request.
- The user name, full name, email address, and organization of each user with an account on the portal.

#### 4.1.1.3 ***Account creator service***

The account creator service provides methods for creating new user accounts, verifying user account information, and verifying that users have the appropriate permissions to create user accounts. The service integrates with built-in user management services to create users on the portal and retrieve information about existing users.

#### 4.1.1.4 ***Authentication service***

The authentication service provides a password-based authentication mechanism for verifying that a web server connecting to a servlet on the portal server has the appropriate permissions. Beforehand, the portal administrator must create a user account with a specific role on the portal. When the web server attempts to connect to a servlet, it posts the user name and password for the account that the portal administrator had created. The servlet then invokes the authentication service, which ensures that the password is correct and that the user has the correct role. If

authentication fails, an error message is sent back to the web server indicating the authentication failure.

#### **4.1.2 Account Request Form**

The account request form, shown in Figure 4.3, provides a means for a user to submit an account request for approval by an administrator. The user fills out his or her name, desired user name, password, email address, and other contact information. Additionally, the user selects which grid (or grids) he or she wishes to use and optionally may request membership in roles or groups defined on the grid portals. Roles and groups give users special permissions or access to restricted parts of the portals.

To prevent users from spamming the administrator with account requests, an image confirmation system is employed. A user must be able to match a code displayed in an image before the account request may be processed by the web server. Since a high-quality image confirmation system is difficult to design and implement, an existing system written for the phpBB forum[24] software was chosen.

Home  
» How to get access  
Go to admin console

**P-GRADE** | portal

P-GRADE Grid Portal access request form

Full name: A Physicist

User name: sci

Password: .....

Confirm password: .....

Address: Nowhere

Phone number: 12345

Institute: Physics Institute

E-mail address: mreiter@127.0.0.1

Grid to use:

- Biomed
- Compchem
- Gilde
- HunGrid
- SEE-GRID
- VOCE

Scientific case and planned usage : Physics research.

Roles (optional):  Optional Role Used for account creation testing

Groups (optional):  Optional Group Used for account creation testing

To prevent automated access requests, you are required to enter a confirmation code. The code is displayed in the image you should see below. If you are visually impaired or cannot otherwise read this code please contact [portalreq@lpds.sztaki.hu](mailto:portalreq@lpds.sztaki.hu).

Confirmation code:

1q43bec

Enter the code exactly as it appears. All letters are case insensitive, and there is no zero.

Figure 4.3 - Account request form

### 4.1.3 Account Request Verification Email

The account-request verification system allows the administrator to retain full control over who is allowed to create an account. When a user submits an account request, an email containing the details of the request is sent to the administrator. In the email, shown in Figure 4.4, there is a link

to the account request dispatcher, which provides accept/deny options to the administrator, sends the account request to each portal listed in the request, and then notifies the administrator if there were any errors. If one or more portals returned error messages, the administrator is provided an option to resubmit the account request to each of the portals that were in error.



Figure 4.4 - Access request verification email

#### 4.1.4 Administration Console

The administration console provides a common login for administrative tasks pertaining to the account creation system. The components that make up the administration console are described in the following sections.

##### 4.1.4.1 *Login Form*

The login form, shown in Figure 4.5, provides a mechanism for the administrator to authenticate with the administration console. If the administrator attempts to navigate to a restricted page, he or she is presented with the login form. After logging in, the administrator is taken directly to the page that was originally requested. Once the administrator has logged in, he or she remains logged in for the duration of the session or until explicitly logging out using the login form.

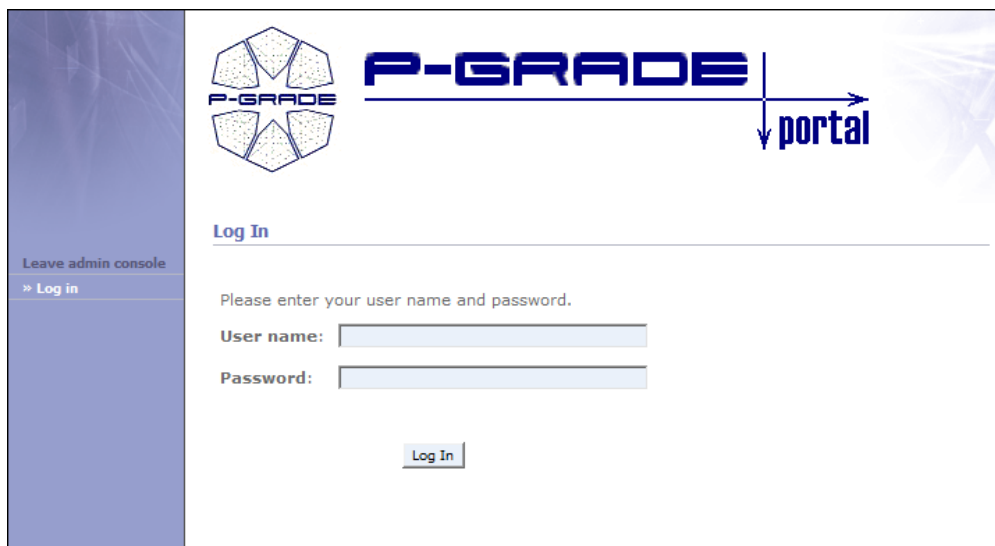
The image shows a web-based login form for the P-Grade portal. On the left is a vertical blue sidebar with the text "Leave admin console" and a "Log in" link. The main content area features the P-Grade logo (a hexagonal grid) and the text "P-GRADE portal" with a blue arrow pointing right. Below this is a "Log In" heading, a horizontal line, and the instruction "Please enter your user name and password." There are two input fields: "User name:" and "Password:". A "Log In" button is centered below the fields.

Figure 4.5 - Administration console login form

##### 4.1.4.2 *Account Request Verification Form*

The account request verification form, shown in Figure 4.7, is activated when the administrator clicks on the link in the verification email sent when a user submits an account request. A form is displayed which allows the administrator to choose whether to accept or deny an account request. A field is provided which allows the administrator to send additional comments to the

user who requested the account. The administrator has four options for how to handle an account request: accept the request, deny the request, accept the request but with the roles and groups reset to their default values, or ignore the request. The third option is provided in case a user requests roles that he or she should not have, but the administrator feels that the request is otherwise valid. A request should be ignored if the administrator does not wish to send any notification to the user or if the email address provided by the user is invalid. Once the administrator submits the form, the server forwards the account request to each portal listed in the request and displays any error messages that were returned. If the account request succeeds on all portals, a success message, shown in Figure 4.7, is displayed. However, if the account request fails on one or more portals, the administrator is presented with an option to resend the account requests to each portal that returned an error message, as shown in Figure 4.8

Welcome, admin!

Leave admin console

Log out

Home

Export user accounts

Change settings

Edit portal information

### Account Request Verification

**Account request information:**  
Full name: *Joe Shmoe*  
User name: *Joe*  
Password: *password*  
Address: *Nowhere*  
Phone number: *12345*  
Institute: *College of Testing*  
E-mail address: *mreiter@127.0.0.1*  
Roles: *Optional Role*  
Groups: *Optional Group*  
Portals: *SEE-GRID*  
Scientific case and planned usage:  
*Grid will not be used.*

**Accept the request?**

Accept

Deny

Accept with default roles and groups

Ignore (no email will be sent to the user)

**Message to user (optional):** Membership in "Optional Group" is restricted to members of SZTAKI.

Continue

Figure 4.6 - Account request verification form





Figure 4.7 - Account request verification success

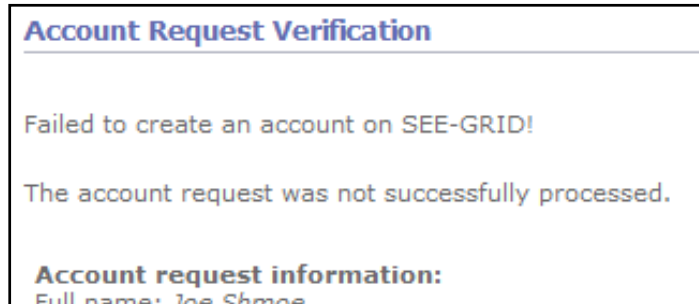


Figure 4.8 - Account request verification failure

#### 4.1.4.3 *User Information Exporter*

The user information exporter, shown in Figure 4.9, allows the administrator to download user information (user name, full name, organization, email address, portals) for all of the users on each grid managed by the account creation system. The format chosen was a comma-separated-value (CSV) file. The CSV file format was chosen because it is compatible with most spreadsheet software and requires significantly less work to implement than other formats such as Excel spreadsheets.

The main purpose of the exporter is to allow the administrator to send mass emails to all of the users on the grids. As such, it was important that each email address only be listed once. To accomplish this, the user information is stored in an associative array keyed by a lower-case version of the email address. If another user has the same email address (excluding case) as a user already listed in the array, the user information is merged using the following rule: for each field (user name, full name, organization), the first non-empty value encountered is used. For

example, if there are two users { “JDoe”, “John Doe”, “”, “jdoe@sztaki.hu”, “SEE-GRID” } and { “JDoe2”, “John Doe II”, “SZTAKI”, “jdoe@sztaki.hu”, “HunGrid” }, the result of merging them would be { “JDoe”, “John Doe”, “SZTAKI”, “jdoe@sztaki.hu”, “SEE-GRID, HunGrid” }.

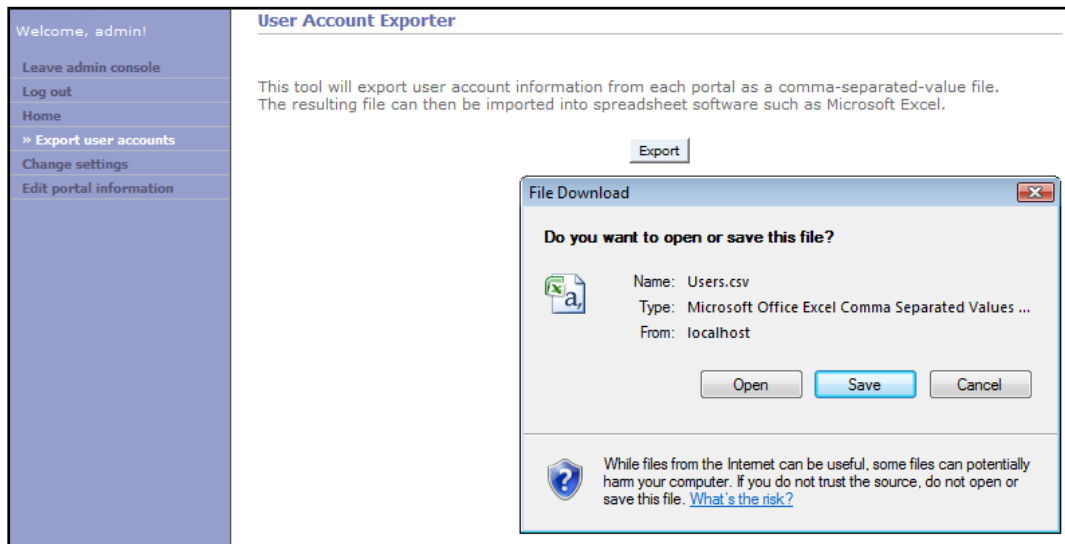


Figure 4.9 - User account exporter

#### 4.1.4.4 *Settings Manager*

The settings manager provides an interface for changing settings that affect the operation of the account creation system. The following settings are available:

- **Require confirmation:** If set to true, the user must enter a confirmation code when requesting an account.
- **Use GD confirmation:** If set to true, the GD version of the confirmation image is used. This version is better than the non-GD version, but requires that the GD2 extension be enabled.
- **GD confirmation foreground noise:** If set to true, foreground noise is used to make the GD-based confirmation harder.

- GD confirmation X grid: The average number of pixels between horizontal grid lines in the GD-based confirmation.
- GD confirmation Y grid: The average number of pixels between vertical grid lines in the GD-based confirmation.
- Session lifetime: The lifetime of a session, in hours. This only applies to the account request page.
- Servlet authentication key: The key used to encrypt the servlet user name and password; should be a unique (preferably random) string that is difficult to guess.
- Admin user name: The administrator's user name.
- Admin password: The administrator's password. This value cannot be retrieved, as it is stored in a hashed form in the database.
- Admin email address: The administrator's email address.

**Figure 4.10 - User account creator settings**

#### 4.1.4.5 *Portal Information Editor*

The portal information editor, shown in Figure 4.11, is used to add or remove portals from the account creation system or edit information about an individual portal. The following fields are available for each portal:

- Portal ID: The name that is used internally by the account creation system when referring to a given portal.
- Portal name: The name is displayed to users.
- Portal URL: The URL that points to the portal's root directory (for example, `https://portal.organization.com/szupergrid`).
- Servlet user name: The user name of a portal user that has been given the ACCOUNT\_CREATOR role.
- Servlet password: The password corresponding to the servlet user name.

- Account creator user name: The user name of a portal user that has been given the `SERVLET_CLIENT` role.
- Account creator password: The password corresponding to the account creator user name.

**Figure 4.11 - Portal information editor**

#### 4.1.5 Security

Prevention of unauthorized user account creation is a key requirement of the account creation system. To prevent an attacker from masquerading as the web server and communicating directly with a portal, each portal requires the web server to authenticate with a “servlet” user name and password. These credentials are stored in an encrypted form on the web server’s database. To construct the encryption key, a password stored in the database (and modifiable by the administrator) is concatenated with a ‘salt’ string and then sent through a one-way hash. The salt, whose purpose is to increase the difficulty of password-guessing attacks[19], is a random string stored as a constant in the account request system’s source code. In order for an attacker to

decrypt the servlet credentials, an attacker will need access to the salt and the administrator's user name and password. As the salt should be different for each instance of the account request system, an attacker would need to gain access to both the web server and its database in order to obtain the user name and password used to communicate with the portals. Creating an account on a portal requires an additional "account creator" user name and password. As with the servlet credentials, the account creator credentials are stored in the web server's database. However, the account creator credentials are encrypted using the administrator's user name and password as the key. The result of this is that only the administrator is capable of retrieving the information required to connect to a portal and create a user account.

There are many ways an attacker can gain access to a system, deny others access to the system, or cause problems. Shown in Figure 4.12 is an analysis of the damage that could be caused by intercepting various transmissions between computers. An additional attack vector is SQL injection. SQL injection is accomplished by entering information into a form in such a way that it will cause arbitrary SQL statements to execute on the database. For example, a form could ask for a user name and then insert it into table using the following PHP code:

```
mysql_query("INSERT INTO user_table (name) VALUES (' . name . "')");
```

If an attacker typed "Joe ' ; DELETE FROM user\_table;" then all of the users would be deleted. The attacker was able to inject an SQL statement by using the string terminator (an apostrophe) to escape out of the user name string and then a semicolon to terminate the current command. At this point, the rest of the text can be any SQL command an attacker chooses.

The account creator uses two methods to prevent SQL injection attacks. Built-into the MySQL library for PHP is a limit that an SQL query may contain only a single SQL command. If multiple commands are issued simultaneously, only the first is actually transmitted to the

MySQL server for execution. As a result, the particular form of SQL injection described above would not work, as the “DELETE FROM user\_table;” statement will never be executed. The second method used to prevent attacks is to escape all strings before inserting them into an SQL query. Apostrophes and other potentially problematic characters are replaced by escape sequences so that they are recognized by MySQL as part of a string instead of being given a special meaning.

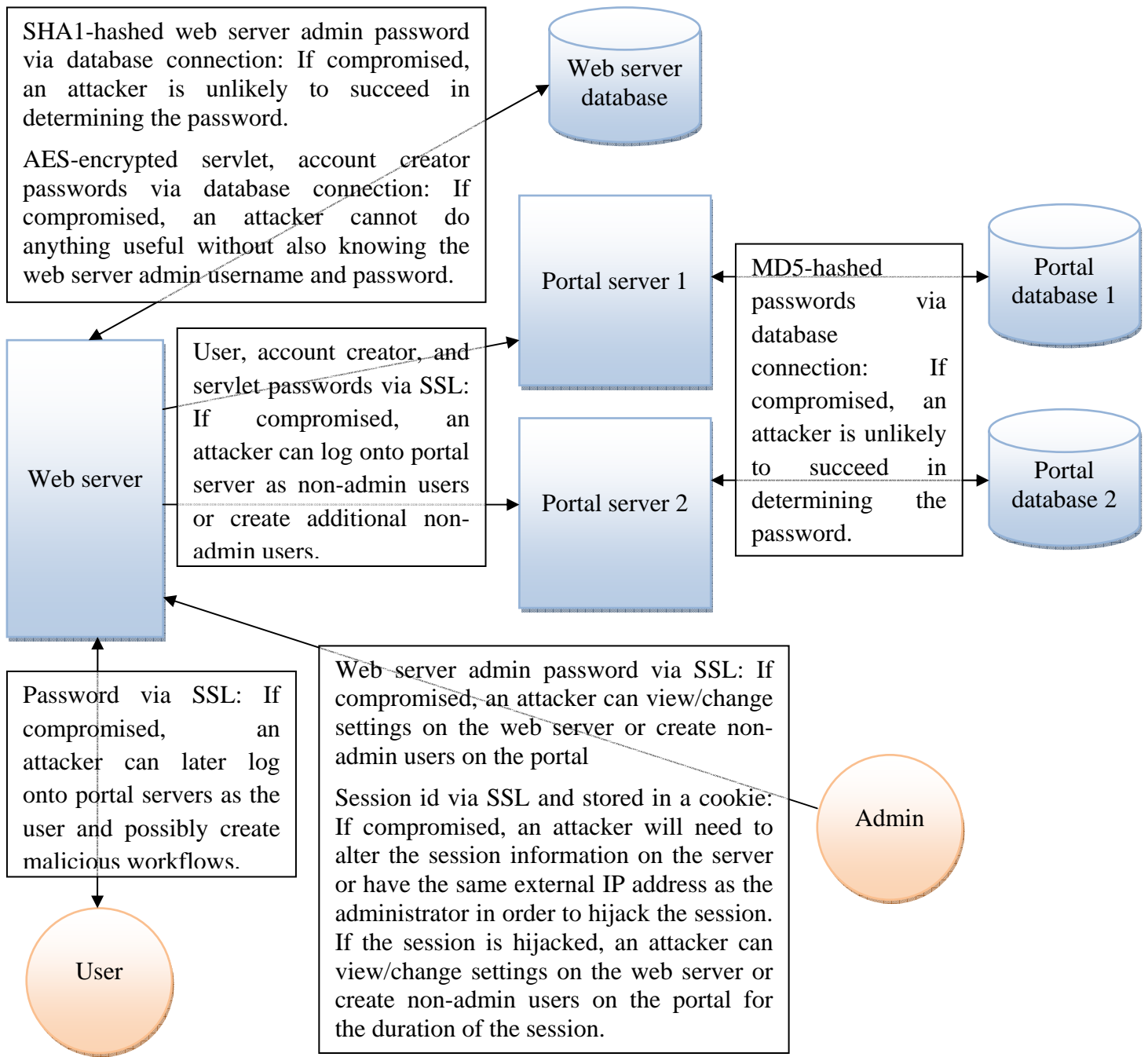


Figure 4.12 - User account creator vulnerability analysis



## 4.2 Email Notification System

The base for the email notification system is a module developed with colleagues at SZTAKI that is currently in use in WS-PGRADE portal and serves the same purpose. That base consisted of several Java packages and a single portlet. The packages were:

- `hu.sztaki.lpds.pgportal.services.notify` – The main package including facilities for setting and retrieving the user’s notify settings as well as a set of utilities associated with notifications. Lastly, it contains the thread that actually performs the notifications.
- `hu.sztaki.lpds.pgportal.services.notify.EventHandlers` – Contains the classes that responds to a status change event and determine if a notification is necessary.
- `hu.sztaki.lpds.pgportal.services.notify.Plugins` – Contains the interface that a notification plugin must implement in order to be used in the system, as well as a concrete example (email).

The portlet itself is a JSP and Java file pair. The JSP page is responsible for displaying to the user all the preferences that they currently have set and allow them to make any necessary changes. The Java file is the logic behind the JSP page that handles updating and displaying of preferences.

### 4.2.1 Portlet Integration

Before any logic could be inserted into the notification system, it had to actually be added to the portal. Because all portlets are based on the JSR-168 specification, it is a straightforward process to add them. After inserting the class and JSP files into their appropriate places in the server, the file “portlet.xml” had to be modified to include an entry for the notification portlet. The file simply defines the portlets to be used in the system. The entry for the notification portlet is:

```

<portlet>
  <description xml:lang="en">Notify</description>
  <portlet-name>notify</portlet-name>
  <display-name xml:lang="en">Notify Portlet</display-name>
  <portlet-
class>hu.sztaki.lpds.pgportal.portlets.notify.NotifyPortlet</portlet-class>
  <expiration-cache>0</expiration-cache>
  <supports>
    <mime-type>text/html</mime-type>
  </supports>
  <supported-locale>en</supported-locale>
  <portlet-info>
    <title>Notifications</title>
    <short-title>Notify</short-title>
    <keywords>workflow</keywords>
  </portlet-info>
</portlet>

```

**Figure 4.13 - Notify Portlet File Entry**

Next, the portlet had to be added to the menu structure of the portal so that it could actually be accessed by a user. This was done by adding an entry to another file, “layout.xml”. This file defines how the menu structure will appear on the portal. The entry for the notification portlet is:

```

<portlet-tab label="notify">
  <title lang="en">Notify</title>
  <table-layout>
    <row-layout>
      <column-layout>
        <portlet-frame>
          <portlet-class>szupergrid#notify</portlet-
class>
          </portlet-frame>
        </column-layout>
      </row-layout>
    </table-layout>
  </portlet-tab>

```

**Figure 4.14 - Notify Portlet Layout Entry**

After all of these changes were made and the portal was restarted, the portlet was (following some other modifications, described below) usable.

#### 4.2.1.1 **Modifications from WS-PGRADE Portal**

Even though it was possible to plug the portlet into the P-GRADE Portal quite easily, some modifications needed to be made before the full functionality of the portlet could be established.

First, several errors had to be addressed. The new portlet included several packages that were

not currently available in P-GRADE, so their respective JAR files had to be added to the classpath. The JSP page also included a tag library that was unavailable in the P-GRADE Portal. To fix this, all calls to the tag library were removed and replaced with the appropriate markup. Second, the JSP page had extra functionality for defining a storage quota notification, although this functionality was never implemented in WS-PGRADE. Therefore, the entire form was dropped from the portlet in the P-GRADE Portal. With these changes made, the portlet was able to read and write its settings as expected. The persistent storage of these settings was in a file called `.notify.xml`. The file itself resides in each user's directory. An example of this file is:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<notify>
  <prop key="wfchg_mess" value="Time: #now##13;##10;User:
#user##13;##10;Portal: #portal##13;##10;Workflow: #workflow##13;##10;Old
Status: #oldstatus##13;##10;New Status: #newstatus##13;##10;Details:
##13;##10;#details#" />
  <prop key="email_addr" value="josh.nedelka@gmail.com" />
  <prop key="wfchg_enab" value="1" />
  <prop key="email_enab" value="1" />
  <prop key="email_subj" value="subject2" />
</notify>
```

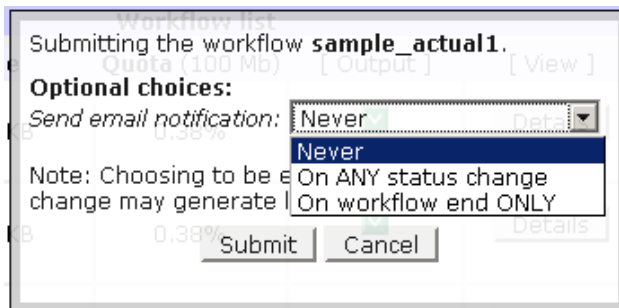
**Figure 4.15 - Notify User Preferences**

The only particularly interesting field is the “wfchg\_mess”. It includes several keys (delimited as “#keyname#”, though not to be confused with entries of the form “#number;” that are effectively line breaks) that are replaced by their appropriate values when the notification message is generated.

## 4.2.2 Submit Form

When choosing to submit a workflow, the user should be allowed to choose when they would like to be notified of status changes. This is one of the requirements of the notification system, and the submission form was designed to satisfy it. It is also functionality that is included in WS-PGRADE, but since the mechanisms for submitting workflows are so different in the two

portals, it was necessary to rewrite the submission form rather than migrate it from WS-PGRADE.



**Figure 4.16 - Workflow Submission Form**

The workflow submission form allows a user three options for notifications (shown above). It also notes that notification on any status change will result in a large amount of emails (proportional to the number of jobs and amount of parallelism). After the user submits their choice, the selection is written to the file “.notify” in the workflow’s directory so that it can later be accessed by the notification system.

One important consideration here is the handling of the “Submit All” button. The functionality is exactly as described: it submits all the workflows currently in the user’s workflow manager. However, since it is possible to specify a level of notification for each workflow, it was necessary to choose one of two possibilities for the submit all:

1. When submit all is pressed, provide functionality that will allow the user to choose a different level of notification for each workflow, or
2. When submit all is pressed, provide allow the user to choose one level of notification and have it apply to all workflows being submitted

For the purposes of simplicity, programmatically and from a user’s perspective, the latter choice was implemented.

### 4.2.3 Backend Integration

Compared to the front-end pieces, the backend was significantly more complex. The largest difficulty was that the P-GRADE Portal was not designed in such a way to make notifications easy, though WS-PGRADE was. As an initial overview, a more detailed view of the event driven system that was implemented is shown below.

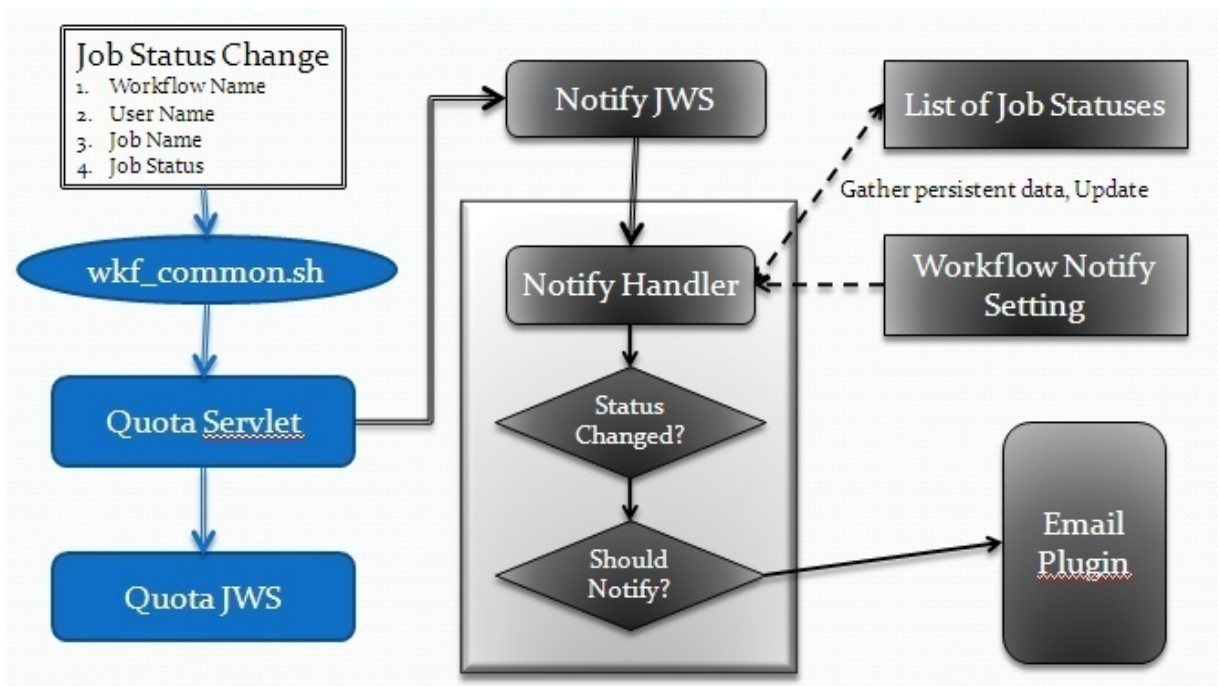


Figure 4.17 - Event Driven Notification Implementation

#### 4.2.3.1 Retrieving and Handling Status Change Events

Initially, there was only one point of capture for the status change events. The function getStatus() inside of wkf\_commons.sh is invoked each time that a job within a workflow changes its status. The function then invokes a servlet that passes the event along to the notification system. The status change event consists of four parts: username, workflow name, job name and new job status. The username and workflow name are necessary so the correct location of the

notify preference files can be found (main notify preferences are stored in the user's directory, while workflow notify preferences are stored in the workflow's directory). The job name is used as the key when storing and retrieving the statuses. And of course, the new job status is the reason that the event has been fired.

After the notification system has been invoked, it proceeds to look up the previous job statuses that it had stored. These are stored in the file ".notify\_jobstatuses", located in the workflow's directory. An example of this file is:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<workflow>
  <job jobname="Job1" status="3" />
  <job jobname="Job2" status="1" />
  <job jobname="Job3" status="5" />
  <job jobname="Job4" status="2" />
</workflow>
```

**Figure 4.18 - Workflow Job Status File**

The new job status is inserted into this list, by either adding a new entry or updating the appropriate job. With all the current job statuses, it is now possible to determine a workflow status. If all job statuses show finished, then the workflow itself is complete. If any one job shows running, then the workflow is running. If any one job shows an error, then the workflow is in error, though this does not mean that the workflow has stopped running. The workflow can continue until it is no longer possible to execute any other jobs without resolving the error.

However, not all possible workflow statuses can be established simply by looking at job statuses. Specifically, a rescue workflow status can never be determined, since it depends on two things: the workflow being stopped and jobs being in an error state. The job statuses can determine the error, but not the termination of the workflow. To get around this problem, the notification system is called from another location as well. The script watchWorkflow.sh runs until a

workflow terminates (due to completion, error, etc.). When it detects the termination of a workflow, it alerts the notification system but does not pass in any other information. The notification system looks at the previously calculated workflow status, and if it determines that a workflow is in error, then it notifies the user of the appropriate rescue status.

After having established the new workflow status, the system looks up the last previous status that it had calculated. This is stored in the file “.notify\_status” in the workflow’s directory. The new status can be compared with the old, and then a series of checks occurs to see if a notification should be sent. They are (in order from first to last):

1. Are the two statuses different?
2. Did a user choose to be notified of this status when they submitted the workflow?
3. Is the current plugin enabled to send notifications?
4. Is the entire system enabled to send notifications?

If any of these questions is answered negatively, then no notification is sent and the process is complete. If they are all answered affirmatively, then the notification is sent. The first and second questions only apply once, before starting the specific notification plugins. The third and fourth questions are handled individually by the respective plugins.

#### 4.2.3.2 *Persistence of Data*

For the three files that the notify system uses for storing preferences and data, it is important to recognize for how long these should be maintained, and when they should be deleted, if at all. This is relevant only within the context of a workflow that terminates prematurely. Workflows that finish successfully can safely have their notification files deleted with no ill effects. The first file, “.notify”, which stores the user’s preference for notification should technically never

need to be cleared. Since the choice is only given to the user when they submit (and not when they resume or rescue a workflow), the system assumes that this preference will persist until completion or abortion and resubmission of the workflow. If the user does resubmit, they will once again be given the option to choose a notification preference, which will overwrite the old one.

The second file, “.notify\_status”, which stores the workflows last recorded status, should also only be cleared on workflow termination. However, unlike “.notify”, “.notify\_status” *needs* to be cleared. The reason for this can be illustrated with a simple example: if a status of “running” is recorded and then the workflow is aborted and resubmitted, then when the notification system goes to look up the last known status it will still see “running”. This is clearly inaccurate since the workflow has just been started. Therefore, termination of a workflow must clear this file. Since there are many ways that termination might occur and since some of them are difficult to detect, it is easier to say that the “.notify\_status” file should in fact be cleared only on workflow submission. This guarantees that when a workflow is started, it has no last known status.

The final file, “.notify\_jobstatuses”, which stores a list of all current job statuses, has slightly different behavior than the previous too. When a workflow is first submitted, it is completely cleared just like the other files. However, when a workflow is rescued from an error state, the file has to be modified slightly. Any jobs not in the completed state when the workflow is rescued have their statuses reset since they are going to be run again. Consider the case where several jobs were in error when the workflow was rescued. If the statuses were not reset, then when a job status change event was passed into the system, one of those jobs might no longer be in error, but the rest certainly would. This would register the workflow with an “error” status, when this is certainly not the case.



#### 4.2.3.3 *Notification Utilities*

The last major portion of the implementation was designing a set of utilities to expedite the process of performing the notifications. Since most of the persistent notification data is stored as XML, many of the utilities are focused on setting or retrieving portions of a document. Specifically, they perform some of the functions described above, including resetting all job statuses, setting job statuses after a rescue, clearing persistent workflow statuses, setting notification frequency and handling user preference updates. The utilities also provide some helper functions for building the notification messages that can be sent out, including adding a timestamp and a human-readable description of the current job statuses.

## **5 Testing**

### **5.1 User Account Creation System**

The user-account creation system is largely user-driven and involves communication between multiple servers. As such, automated testing would be difficult to implement and thus all testing was done manually.

#### **5.1.1 Functional Testing**

The initial stage of functional testing was accomplished by going through the steps of requesting and then approving or denying a user account, and finally logging into the portal to verify that the account was created successfully and that all of its attributes were correct. Additionally, the administration console was tested by changing settings and then verifying that the database was updated correctly and that the settings had the desired effect on the account creation process.

The purpose of the second stage of functional testing was to ensure that the account creation system would work in a production environment as opposed to the sandbox-like environment in which it was developed. The source code for the account creation system was sent out to the web master and the portal administrator to be installed on the production web server and portal server. Next, the person normally responsible for creating user accounts conducted manual tests of the system to ensure that it worked correctly and exposed the desired functionality.

### **5.2 Email Notification System**

The notification system test consisted of two main parts (performance and functionality), each of which was broken up into more manageable and measurable pieces.

#### **5.2.1 Performance Testing**

The two types of performance testing that the system was put through were testing of the system under various loads, and testing the system compared to a base implementation. 1, 20 and 50 workflows were run, first with the notification system sending emails after every workflow status

change. Next, the entire notification system was disabled (none of its code was run) and the same number of workflows were executed again. To measure performance, the Linux application “top” was used in batch mode, recording statistics for the java process on the system at intervals of one second.

#### 5.2.1.1 *One Workflow*

The processor and memory usage for the java process with and without the notification system appear in Figure A.1 - 1 Workflow with Full Notification and Figure A.2 - 1 Workflow with No Notification, respectively. It is clear that the memory usage is the same for both graphs (and indeed, it changes little even up to the 50 workflows). However, there is certainly a discrepancy where processor usage is concerned. It is important not to look at the absolute heights of the processor spikes (justified in the section describing 20 workflows, below), but rather to look at the changes in heights of the spikes across the lifetime of the workflow. When no notification is enabled, processor usage typically falls between 5 and 10 percent. In general, the spikes towards the end of the workflow are higher, though this is not always true. The “standard” spike in the graph with notifications enabled is between 8 and 10 percent usage, though there are many more spikes spaced throughout the graph that greatly exceed that. Ignoring the 100 percent spike, the processor usage occasionally reaches between 20 and 50 percent. These are indicative that the notification system is more CPU intensive, though as the next cases show, one workflow is not necessarily characteristic of the performance graphs.

#### 5.2.1.2 *20 Workflows*

The two graphs showing java process statistics with 20 workflows executing (Figure A.3 - 20 Workflows with Full Notification and Figure A.4) show something a bit more interesting than the graphs of only one workflow. In this particular set of graphs, the typical absolute CPU usage of the system with notification enabled is in fact less than that of the system without any

notification. The reason for this is that the readings were taken quite separately. Whereas most of the graph pairs were taken back to back, leaving no time in between, the graphs of 20 workflows were separated by several days, so the condition that the server was in must have changed substantially. This is the reason that it is unwise to simply compare the absolute height of the various CPU spikes: they are too dependent on the state of the server. However, the difference in heights within a single graph continues to be a strong indicator of performance.

With 20 workflows executing without notification, there is a very standard amount of CPU usage being used that typically falls between 7 and 8 percent. There are several spikes as well, most of which are focused during the submission phase of the workflows. It is clear that there is more overhead with 20 workflows than with only the single one. However, when compared to the 20 workflows with notification, the system without notification is apparently more efficient. The base for the system with notification is only around four percent, but it consistently reaches to nearly ten percent. This is in stark contrast to the system without notifications, which only has the sparsely placed large spikes that the system with notification shares.

#### 5.2.1.3 **50 Workflows**

The two graphs showing java process statistics with 50 workflows executing (Figure A.5 and Figure A.6) combine the features of the single and 20 workflows. Like the single workflow graphs, the 50-workflow graph with notification has consistently more processor usage. And like the 20 workflows graphs, the 50-workflow graph shows the same basic patterns of processor spikes. As more workflows are added to the sample, the graphs showing with and without notification begin to seem more and more similar. However, it is still apparent that the frequency and size of the processor spikes relative to the baseline are more substantial with the notification system, rather than without.

### **5.2.2 Functionality Testing**

Since the portal is such a complex and user-driven piece of software, it proved difficult to provide any automated testing facilities to verify functionality of the notification system. Obviously, the notification system needs a workflow to be submitted and running before it can even be called, so it was necessary to set this up manually. The two main types of workflows were tested: normal and parameter study. Each piece of the system was tested thoroughly until it was considered working (criteria for each individual piece are discussed below), at which point another piece of the system was added to the tests, and everything was run again. Functionality was verified by output logs and notifications received.

Each layer was tested under the following scenarios:

- Single normal workflow
- Multiple copies of the same normal workflow
- Multiple different normal workflows
- Single parameter study workflow
- Multiple copies of the same parameter study workflow
- Multiple different parameter study workflows
- Mix of normal and parameter study workflows

Each of these scenarios was run in several ways. First, they were allowed to go to completion. Next, they were randomly terminated and rescued or aborted and resubmitted. Finally, errors were introduced, fixed, and the workflows were restarted. All of the workflows used are the samples that come with P-GRADE Portal.

First, a discussion on testing the main use case of the system: receiving notifications. The most fundamental piece of the system that was tested was the ability to accurately capture the job and workflow status change events. These occur in two places (watchWorkflow.sh and wkf\_commons.sh) and each triggers the same servlet. This was the logical first layer to test. In order to verify the data, Tomcat's log files were used to read from Java's standard out. Once it was established that each entry point was sending the appropriate data, the next layer was added.

This layer consisted of a Java Web Start file that was invoked by the servlet as well as the logic to determine if a notification should be sent and update all statuses. This was a fairly involved test, verifying that data files were being written and read correctly. The workflows were suspended periodically so that the data files could be checked manually against what the log files indicated they should contain. The notification logic itself wrote its decision to send a notification, as well as the calculated status of the workflow, to the log file.

The final layer to be tested in the system was the actual emailing. The individual components were the send thread and the email plugin. Initially, the email plugin was causing problems. All other tests and logs indicated that an email was in fact being sent, despite the fact that none appeared. Eventually, this issue was tracked to a classpath conflict, and was resolved.

#### **5.2.2.1 *Front End Testing***

Two more parts of the whole system needed to be tested, both on the front end: the preferences portlet and the workflow submission form. The preference portlet itself was straightforward to test: it only needed to update and read statuses. A quick check of the preference file that it wrote was enough to verify it, though bad input was tested as well.

The submission form had to be tested at two points: on the main workflow manager (single and “Submit All”) and on an individual workflow’s details. Three data files had to be updated in both cases: the .notify preference file for notification frequency, the .notify\_status file for tracking a last status, and the .notify\_jobstatuses file for tracking job statuses. All had to be wiped out on every submission. After establishing that this was the case, all functionality testing was complete.

## **6 Conclusions**

### **6.1 The Final Account Creation System**

The account creation system achieves all of the initial requirements in addition to a multitude of requirements added later. The account creation processes has been automated to the point that once the initial configuration has been completed, the only thing an administrator must do when a user submits an account request is to indicate whether the request should be accepted or denied and optionally enter a comment to be sent along with the acceptance/denial notification to the user.

The back-end portal components of the account creation system are already available for download and will be included in a future release of the P-GRADE Portal. The front-end web server components are distributed separately from the portal and may be downloaded from SourceForge.net.

### **6.2 The Final Email Notification System**

The notification system in its final form achieves all of the requirements initially set forth. It provides a useful piece of functionality for end users of the P-GRADE Portal and provides portal developers with a way to expand this functionality. The system will appear in a future release of the P-GRADE Portal, which is freely available on SourceForge.net.



## **7 Future Work**

### **7.1 Account Creation System**

Given the short time frame of the project, implementing all of the desired features was not possible. As a result, one of the original requested features, the bulk account creator, has been left as a stub to be finished in later versions of the account creation system. Additionally, parts of the system could be improved in future versions to reduce overall complexity and to improve compatibility.

#### **7.1.1 Bulk Account Creator**

The bulk account creator, part of the administration console, provides a means for the administrator to create a large number of user accounts simultaneously. For example, an instructor may need to create thirty user accounts to use for a training exercise. With both the current system and the automated account creation system, the creation of so many users would take a considerable amount of time. The bulk account creator can cut down on that time by generating user names, and possibly other fields, dynamically based on a template and a counter that increments for each user created. The administrator would simply fill in the user information once, specify how many users to create, and submit the form. At this point, all of the users are created automatically.

#### **7.1.2 Account Request Viewer**

The account request viewer, which would be part of the administration console, would display a list of pending account requests. Buttons would be provided to accept or deny request. There are two ways to implement this: there could be individual buttons for each request in the list, or there could be check boxes next to each request and a few buttons at the bottom (or top) of the page that would accept or deny all of the selected account requests.

### **7.1.3 Testing of Portal Information**

The portal information editor allows the administrator to enter the URL and credentials for connecting to each portal. Currently, the administrator must attempt to create an account or otherwise attempt to connect to a portal in order to verify that the information entered is correct. The portal information editor could be extended with a feature that would allow the administrator to click on a button and then the web server would attempt to contact each portal and then inform the administrator of any failures. This way, the administrator would know immediately if any information is incorrect.

### **7.1.4 Improved Database Support**

Currently, the only type of database supported is MySQL. As future work, a developer could modify the code so that it uses database-independent libraries and thus support other database types such as Oracle or Microsoft SQL Server.

## **7.2 Notification System**

The implemented notification system, while extremely useful to users, presents only a bare bones set of features. While it would have been impossible to implement everything, what follows is a list of features that would be well suited for release in future versions.

### **7.2.1 Additional Plugins**

The first point of extension for the system concerns the actual plugin implementations. The current system only allows emails to be sent, but there is a simple interface to implement in order to develop new plugins.

#### **7.2.1.1 SMS Messaging**

SMS (text) messaging is likely the next logical means of notification following email. It is useful for users of the portal who do not have constant access to email, but who do have a cell phone. The difficulty in implementation is there are no free libraries that expose the necessary

functionality. A library would need to be licensed from an outside company or developed internally, costing either time or money.

#### **7.2.1.2 *Phone Calls***

Notifying users through direct phone calls, rather than via some kind of text-based format, presents another attractive, yet problematic, option. It makes the application much more accessible, both in terms of how it could be used (to a cell phone, landline, etc.) and who could use it (blind users would no longer need assistance). However, actually calling a user to report a status change presents a new set of problems. From a usability standpoint, it would get very frustrating if the system was attempting to call someone every few seconds. There would have to be more control over when a notification would be sent. This approach is also different from a technical standpoint. Now, instead of simply sending text in some way, it would have to first be converted to speech. There would also likely be options to allow the user to replay the notifications, if they desired, so there would have to be interaction with a touch-tone phone as well. Like the SMS option, calling users would certainly require external libraries.

### **7.2.2 *More Monitoring***

Now, only workflow statuses are being monitored, and even they are only being monitored in very specific ways. Both of these limitations present interesting expansion options.

#### **7.2.2.1 *Granularity of Notification***

The user is given two options for how they would like to be notified: on any workflow status change and only on workflow completion. This could be extended to allow a user to choose specific statuses that they would like to be notified of (there are currently nearly 10 statuses that a workflow can be in).

There could also be options to allow certain events to automatically trigger a notification, regardless of whether or not the workflow status has changed. For instance, completion of a specific job could send a notification to the user if they so chooses.

#### **7.2.2.2 *Types of Monitoring***

Initially, there was a form to let users set up a notification to alert them to the amount of storage that they had remaining, whenever it changed. This feature was dropped from the current P-GRADE Portal version of the notification system, but it would likely prove a useful feature to add in the future. Continuing with that basic idea, there are many things that a user might like to be notified of that the Portal currently tracks. Notifications could be tied into the Information System to alert users when various elements change states. There could also be administrative notifications sending statistics for the entire server. Realistically, any portion of the portal could be given its own highly customizable set of notifications. While that would require a lot of work designing the front end portlets and back end integration, at least some work could be reused. The actual notification mechanisms could easily be modified to support notifying from multiple different sources.

#### **7.2.2.3 *Parameter Study Workflows***

In the current system, a parameter study workflow is treated just like a normal workflow. That is, notifications are sent whenever a job triggers them. However, parameter studies are unique in that they contain and monitor element workflows and only have a few special jobs that they can run. What ends up happening is a parameter study will report no status changes (it is almost always in the “Submitted” state”) until all of its element workflows have terminated. It would be convenient to allow the user to receive notifications for these element workflows, if they choose, so that the actual progress of the parameter study workflow could be monitored.



## 8 References

- [1] About Gridsphere. *Gridsphere Project*. [Online] [Cited: March 31, 2008.] <http://www.gridsphere.org/gridsphere/gridsphere/guest/about/r/>.
- [2] About the Java Technology. *Sun Microsystems*. [Online] [Cited: April 3, 2008.] <http://java.sun.com/docs/books/tutorial/getStarted/intro/definition.html>.
- [3] **Anderson, Tim**. Introducing XML. *ITWriting*. [Online] 2006. [Cited: April 21, 2008.] <http://www.itwriting.com/xmlintro.php>.
- [4] **Antognini, Christian**. Is Oracle Database Moving Towards Grid Computing? *Trivadis*. [Online] [Cited: March 30, 2008.] [http://www.trivadis.com/Images/grid\\_computing\\_en\\_tcm17-11759.pdf](http://www.trivadis.com/Images/grid_computing_en_tcm17-11759.pdf).
- [5] Apache Tomcat. *Apache Tomcat*. [Online] [Cited: April 2, 2008.] <http://tomcat.apache.org/>.
- [6] **Berlich, Rüdiger**. Grid Computing - Roots, Motivations and Implementation. *EGEE*. [Online] July 6, 2004. [Cited: April 1, 2008.] <http://www.egee.nesc.ac.uk/trgmat/events/040920GridKa/talks/slides/whatIsGrid.pdf>.
- [7] **Byous, Jon**. Java Technology: The Early Years. *Sun Microsystems*. [Online] April 2003. [Cited: April 3, 2008.] <http://java.sun.com/features/1998/05/birthday.html>.
- [8] **Foster, Ian**. Globus Toolkit Version 4: Software for Service-Oriented Systems. 2005. [http://books.google.com/books?hl=en&lr=&id=YN72O\\_14JzkC&oi=fnd&pg=PA2&dq=Globus+toolkit&ots=1QxQAT0QAL&sig=ysH\\_kOP8q1O\\_Ispi0KFnnlxeYqA#PPA2,M1](http://books.google.com/books?hl=en&lr=&id=YN72O_14JzkC&oi=fnd&pg=PA2&dq=Globus+toolkit&ots=1QxQAT0QAL&sig=ysH_kOP8q1O_Ispi0KFnnlxeYqA#PPA2,M1).
- [9] gLite: Lightweight Middleware for Grid Computing. [Online] [Cited: April 21, 2008.] <http://glite.web.cern.ch/glite/>.
- [10] JSR 168: Portlet Specification. *Java Community Process*. [Online] October 27, 2003. [Cited: April 4, 2008.] <http://www.jcp.org/en/jsr/detail?id=168>.
- [11] Java Platform, Enterprise Edition (Java EE) Support in NetBeans IDE. *NetBeans*. [Online] [Cited: April 4, 2008.] <http://j2ee.netbeans.org/>.
- [12] JavaScript. *Wikipedia*. [Online] [Cited: April 21, 2008.] <http://en.wikipedia.org/wiki/JavaScript>.
- [13] JavaServer Pages Overview. *Sun Developer Network*. [Online] [Cited: April 2, 2008.] <http://java.sun.com/products/jsp/overview.html>.

- [14] **Joseph, Joshy and Fellenstein, Craig.** *Grid Computing*. s.l.: IBM Press. [http://books.google.com/books?hl=en&lr=&id=2e73K\\_jXdfcC&oi=fnd&pg=PR21&dq=grid+computing+uses&ots=fQAmgz0f\\_2&sig=YoqsNaxLvCNxZ0ToG0pz8FfGNCs#PPA44,M1](http://books.google.com/books?hl=en&lr=&id=2e73K_jXdfcC&oi=fnd&pg=PR21&dq=grid+computing+uses&ots=fQAmgz0f_2&sig=YoqsNaxLvCNxZ0ToG0pz8FfGNCs#PPA44,M1).
- [15] Laboratory of Parallel and Distributed Systems . *MTA SZTAKI LPDS*. [Online] [Cited: March 30, 2008.] <http://www.lpds.sztaki.hu/>.
- [16] Learn About Java Technology. *Java.com*. [Online] [Cited: April 1, 2008.] <http://www.java.com/en/about/>.
- [17] **Livny, Miron, Tannenbaum, Todd and Thain, Douglas.** Distributed computing in practice: the Condor experience. s.l. : Wiley InterScience, 2005. 17, pp. 323–356.
- [18] **Merkey, Phil.** Beowul History. *Beowulf Project* . [Online] 2007. [Cited: March 29, 2008.] <http://www.beowulf.org/overview/history.html>.
- [19] **Morris, Robert and Thompson, Ken.** *Password Security: A Case History*. Murray Hill : Bell Laboratories, 1978.
- [20] NetBeans IDE 6.0 Features. *NetBeans*. [Online] [Cited: April 4, 2008.] <http://www.netbeans.org/features/>.
- [21] P-GRADE Grid Portal. *P-GRADE Grid Portal*. [Online] [Cited: March 31, 2008.] <http://www.lpds.sztaki.hu/pgportal/?m=0&s=0>.
- [22] Parellel computing. *Wikipedia*. [Online] [Cited: April 21, 2008.] [http://en.wikipedia.org/wiki/Parallel\\_computing](http://en.wikipedia.org/wiki/Parallel_computing).
- [23] **Patil, Sunil.** What is a Portlet. *OnJava*. [Online] September 15, 2005. [Cited: April 4, 2008.] <http://www.onjava.com/pub/a/onjava/2005/09/14/what-is-a-portlet.html>.
- [24] **phpBB Group.** *phpBB web site*. [Online] [Cited: March 20, 2008.] <http://www.phpbb.com>.
- [25] **Sandholm, Thomas and Gawor, Jarek.** *Globus Toolkit 3 Core – A Grid Service Container Framework*. 2003. [http://66.102.1.104/scholar?hl=en&lr=&q=cache:vw2CSgO4DmUJ:dwdemos.dfw.ibm.com/wstk/common/wstkdoc/ogsa/docs/gt3\\_core.pdf+Globus+toolkit](http://66.102.1.104/scholar?hl=en&lr=&q=cache:vw2CSgO4DmUJ:dwdemos.dfw.ibm.com/wstk/common/wstkdoc/ogsa/docs/gt3_core.pdf+Globus+toolkit).
- [26] **Seshadri, Govind.** Understanding JavaServer Pages Model 2 Architecture. *JavaWorld*. [Online] December 29, 1999. [Cited: April 2, 2008.] <http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html>.
- [27] The Institute. *MTA SZTAKI*. [Online] [Cited: April 1, 2008.] <http://www.sztaki.hu/institute/>.
- [28] The Java Language Environemnt. *Sun Developer Network*. [Online] [Cited: April 2, 2008.] <http://java.sun.com/docs/white/langenv/Intro.doc2.html>.

[29] Welcome to the NetBeans Community. *NetBeans*. [Online] [Cited: April 4, 2008.] <http://www.netbeans.org/about/index.html>.

[30] **Wilton-Jones, Mark**. JavaScript history. *How to Create*. [Online] [Cited: April 21, 2008.] <http://www.howtocreate.co.uk/jshistory.html>.



## A. Email Notification System Performance Graphs

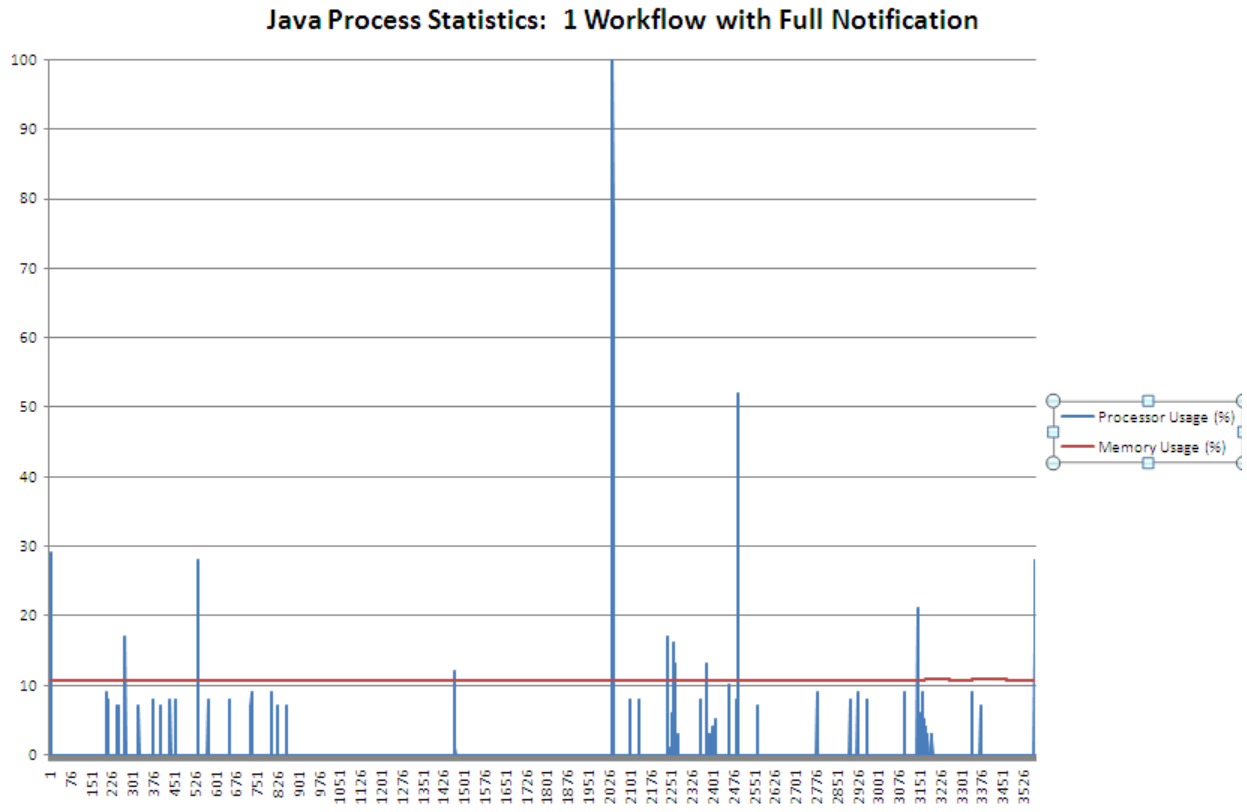


Figure A.1 - 1 Workflow with Full Notification

Java Process Statistics: 1 Workflow with No Notification

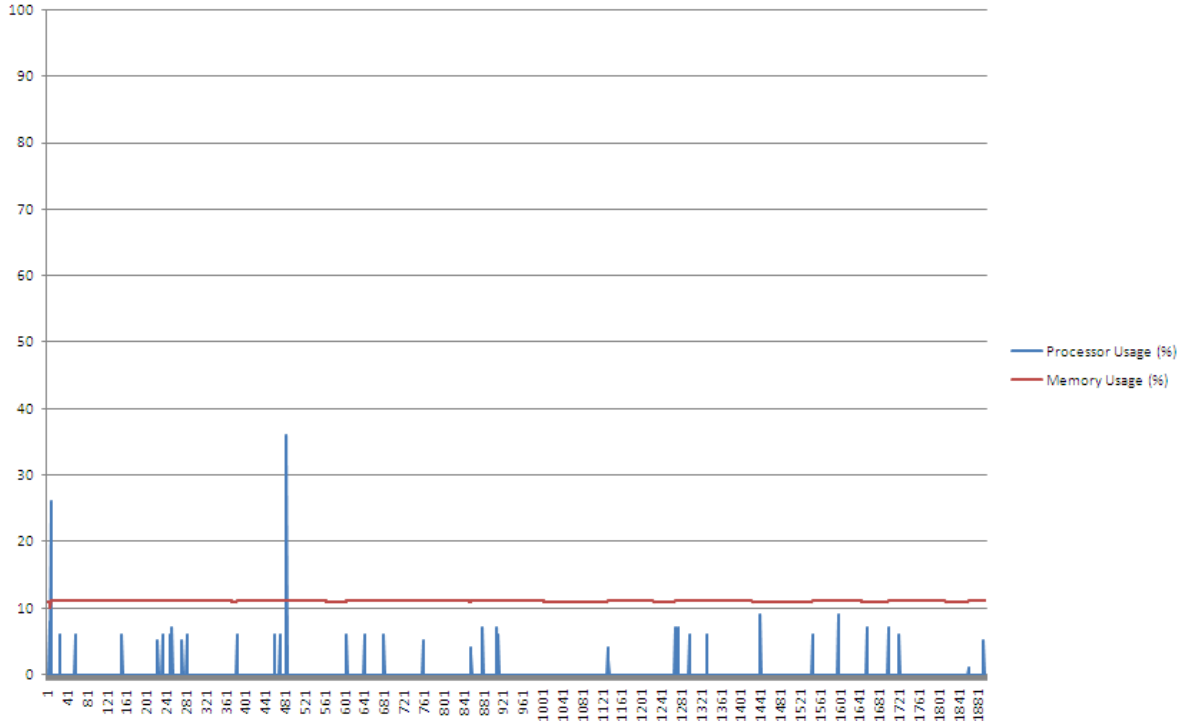


Figure A.2 - 1 Workflow with No Notification

Java Process Statistics: 20 Workflows with Full Notification

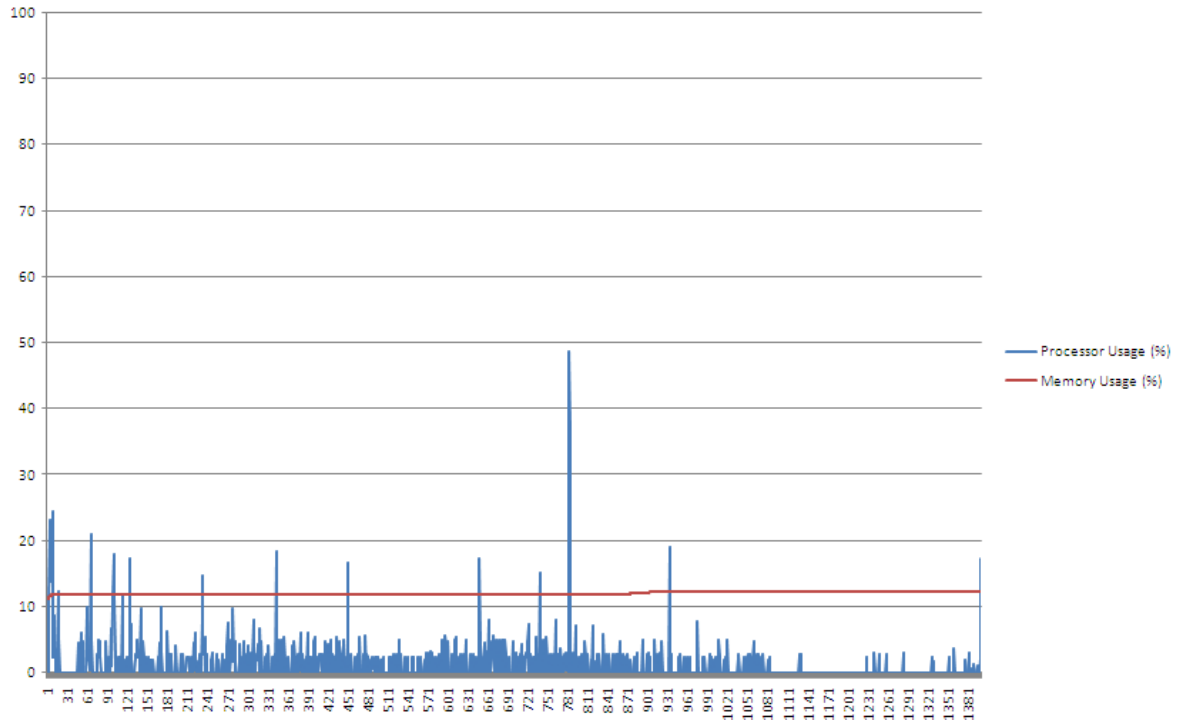


Figure A.3 - 20 Workflows with Full Notification

Java Process Statistics: 20 Workflows with No Notification

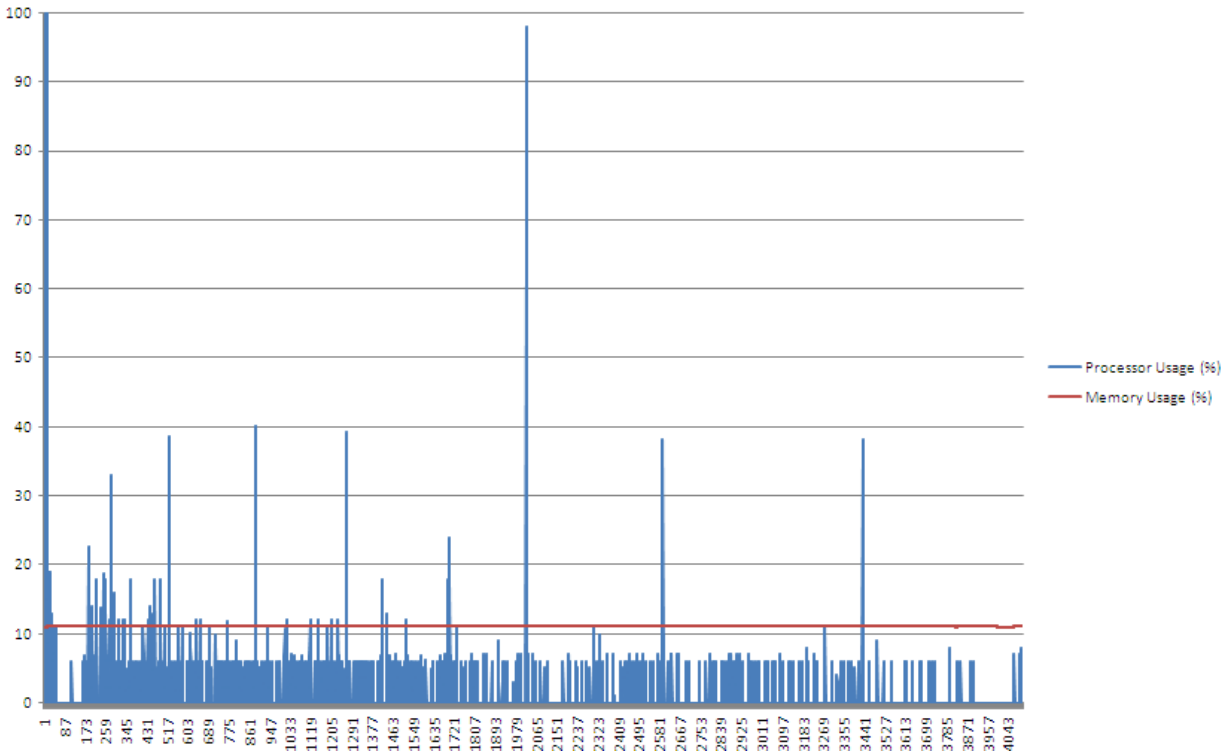


Figure A.4 - 20 Workflows with No Notification

Java Process Statistics: 50 Workflows with Full Notification

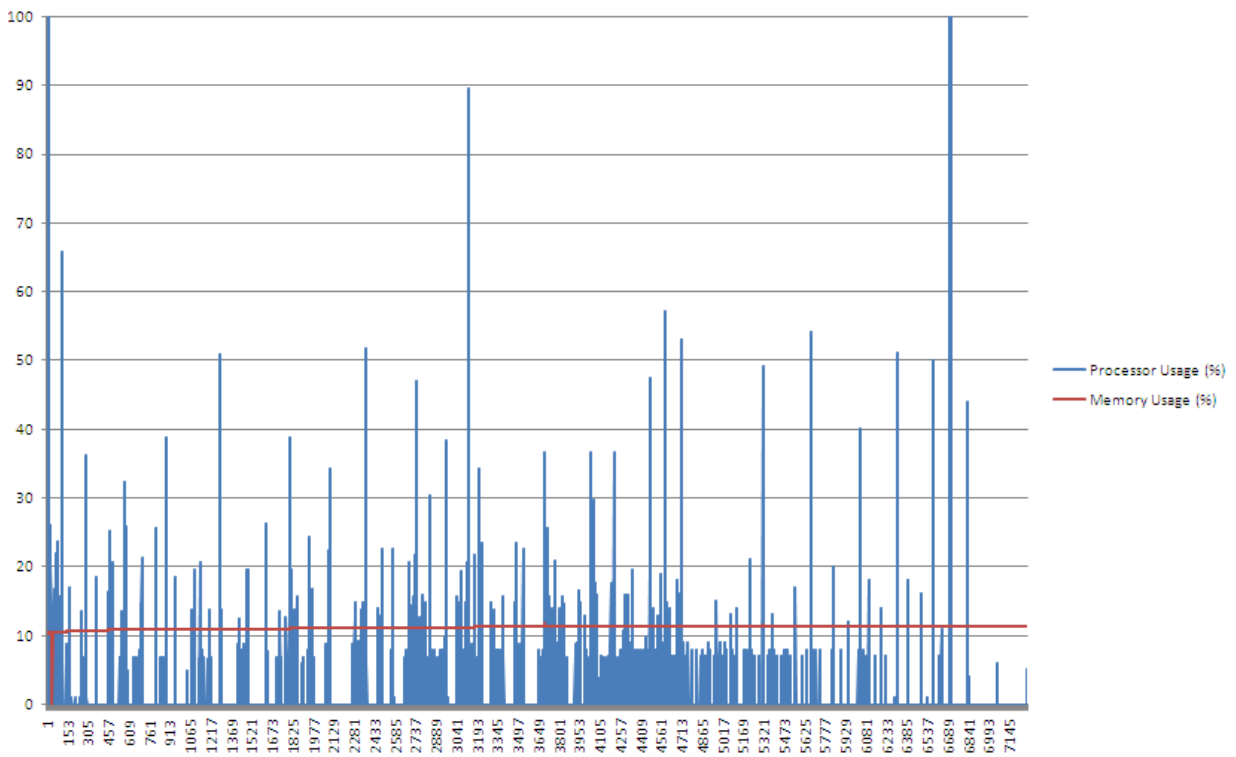


Figure A.5 - 50 Workflows with Full Notification

### Java Process Statistics: 50 Workflows with No Notification

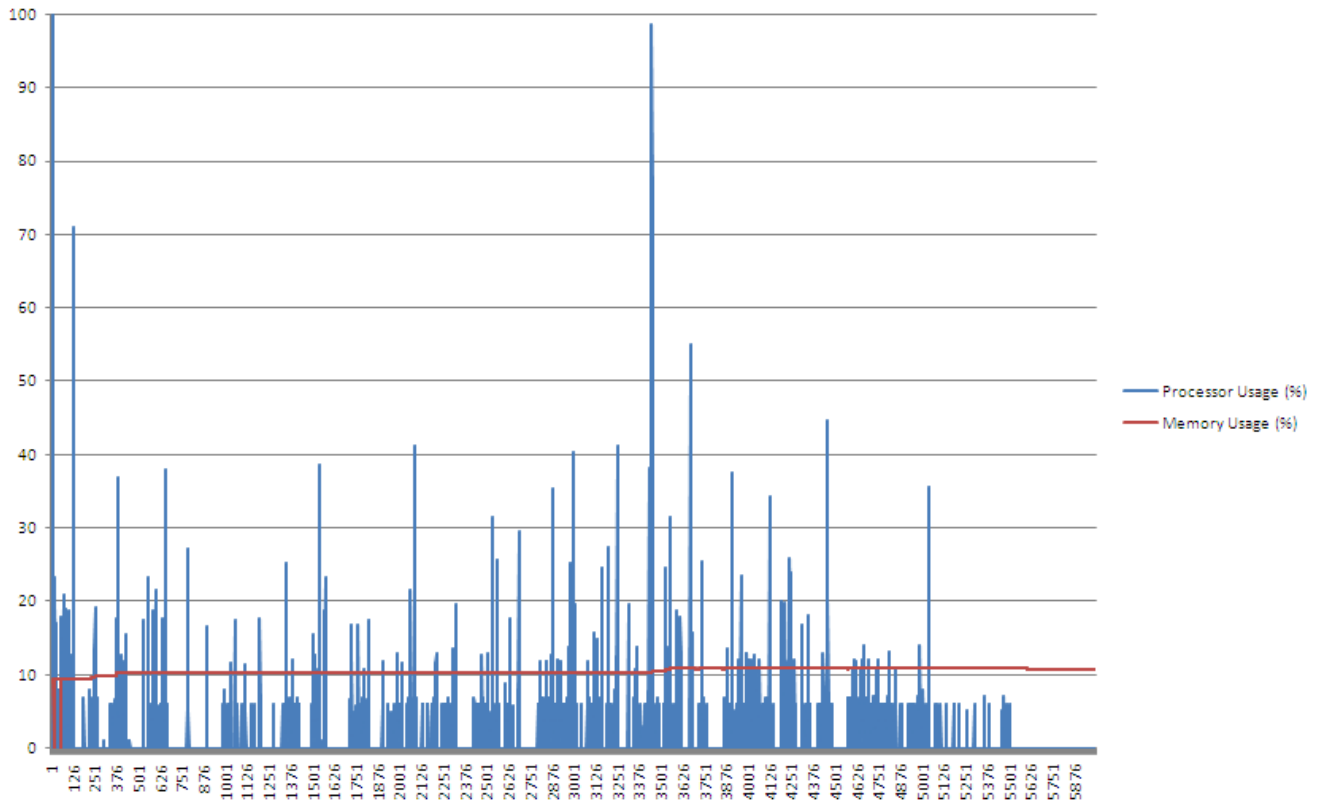


Figure A.6 - 50 Workflows with No Notification

## **B. Account Creator System Administrator Manual**

# **P-GRADE Multi-Portal Account Creation System Administrator Manual**

### **B.1. Introduction**

The purpose of this manual is to instruct administrators on how to install, configure, and maintain the P-GRADE Multi-Portal Account Creation System.

The account creation system is intended to provide a solution to the problem of a single organization administrating multiple grid portals. Instead of forcing an administrator to manually create user accounts on each portal requested by a user, all of the work of creating an account is done automatically. The administrator merely has to approve or deny requests and perform occasional maintenance to keep the system running.

### **B.2. Account Creation Workflow**

Located on the web server are an account request form, an automatic account creator, and a setting manager. On each portal server is an account creator service. Attached to each portal server and the web server are databases for persisting account data and settings. A diagram of the components and the flow of data between them is shown in Figure 1. The sequence of events is as follows:

1. A user seeking an account on one or more grid portals fills out the account request form on the web server, as shown in Figure 2.
2. The web server retrieves the request information from the form and verifies it with the account creator for each portal listed in the request.
3. The web server stores the request information in the common database for later retrieval.
4. The web server sends an email to the administrator with a link to accept or deny the account request. Another email, shown in Figure 3, is sent to the user notifying him or her that the

account request was successful. A message is then displayed to the user indicating whether the emails were sent out successfully, as shown in Figure 4.

5. The administrator clicks on one of the links in the email.
  - o The 'accept' link causes the automatic account creator to accept the request as it is.
  - o The 'accept with default roles and groups' link causes the automatic account creator to accept the request, but with the requested roles and groups reset to their default values.
  - o The 'deny' link causes the automatic account creator to delete the account request and send the user an email indicating that the account request has been denied.
6. If the administrator accepts the account request, the automatic account creator deletes the request information from the common database and logs the user name and email address so that they will be available for future reference.
7. The request information is dispatched to the account creator of each portal listed in the request.
8. The account creator on each portal saves the account information in the portal database.
9. The automatic account creator sends an email to the user indicating that the account request has been accepted.

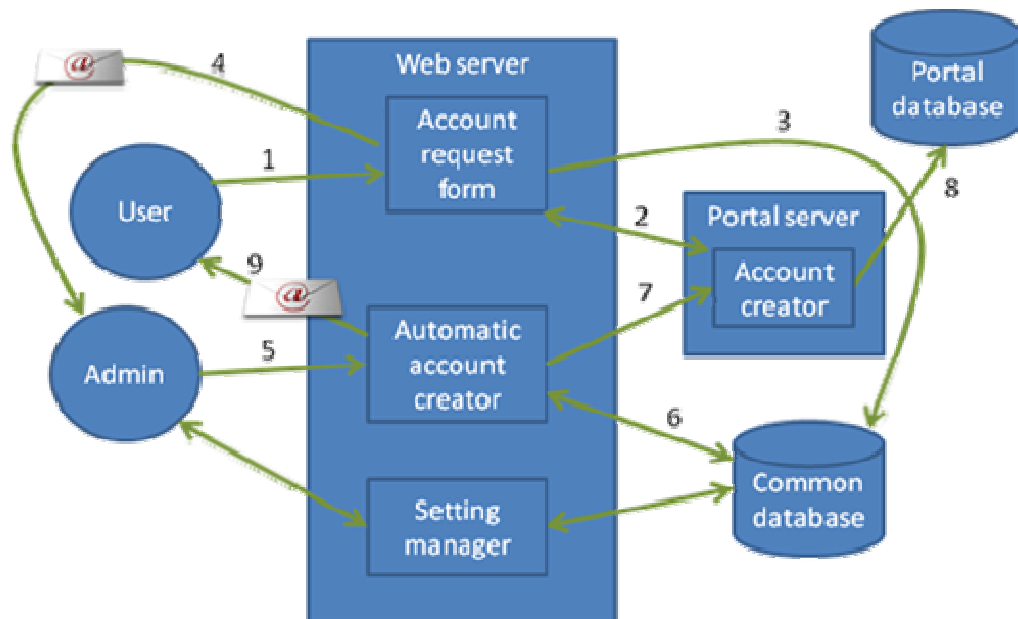



Figure 1: Workflow diagram



# P-GRADE

---

portal

---

**P-Grade Grid Portal access request form**

Home

» How to get access

Go to admin console

**Full name:**

**User name:**

**Password:**

**Confirm password:**

**Address:**

**Phone number:**

**Institute:**

**E-mail address:**

**Grid to use:**

- Biomed
- Compchem
- Gilda
- HunGrid
- SEE-GRID
- VOCE

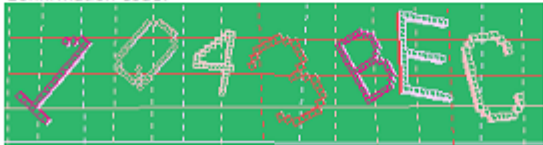
**Scientific case and planned usage :**

**Roles (optional):**  Optional Role    Used for account creation testing

**Groups (optional):**  Optional Group    Used for account creation testing

To prevent automated access requests, you are required to enter a confirmation code. The code is displayed in the image you should see below. If you are visually impaired or cannot otherwise read this code please contact [portalreq@lpds.sztaki.hu](mailto:portalreq@lpds.sztaki.hu).

Confirmation code:



Enter the code exactly as it appears. All letters are case insensitive, and there is no zero.

Figure 2: Account request form

## P-GRADE Portal Access Request Notification

P-GRADE Grid Portal Team [portalreq@lpds.sztaki.hu]

Sent: Tue 4/15/2008 2:24 PM

To: mreiter@127.0.0.1



**P-GRADE Portal Access Request Notification**

Dear A Physicist,

Your request (for access to SEE-GRID) has been successfully processed.

Our portal administrator will send the required information to your e-mail address (**mreiter@127.0.0.1**) very soon.

Yours sincerely,  
MTA-SZTAKI P-GRADE Grid Portal Team  
<http://www.lpds.sztaki.hu/pgportal>

© Copyright 2004-2008, MTA-SZTAKI LPDS, Hungary. All rights reserved.  
-> webmaster@lpds.sztaki.hu

Figure 3: Account request notification





Figure 4: Account request successful

## B.3. Installation

The following sections describe the steps required to install the account creation system.

### B.3.1. Server Topology

The account creation system will work with a wide variety of server topologies. Described in Figure 5 is one possible topology; you may modify it as you see fit. In the topology shown, each server is in a separate physical location. There is a single web server with hosting a common account creation website connected to one or more servers running P-GRADE portal. The web server and portal servers each have their own database, which can be on either the same computer or a separate computer. For additional security, each web server is located behind a firewall and only the specific ports needed are left open.

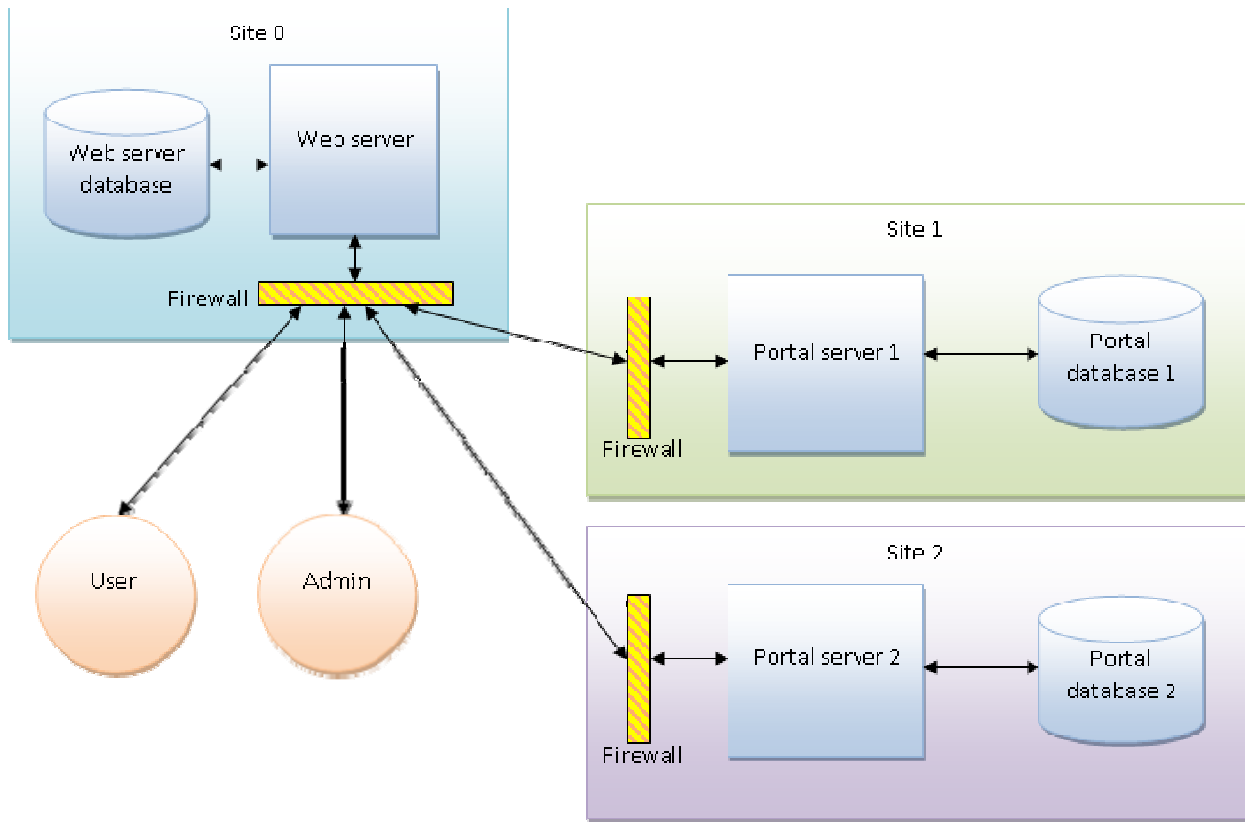


Figure 5: Server topology

### B.3.2. Prerequisites

In order for the account creation system to work, the following software must be installed and accessible by the web server:

- Web server software, such as Apache Web Server or Internet Information Services (IIS)
- PHP version 5.0 or higher
- MySQL Server version 4.1.2 or higher

Additionally, PHP must be configured to use the following extensions:

- `pecl_http` (used to communicate with the portals)
- `Mcrypt` (used for encryption of data)
- `MySQL` (used to connect to a MySQL database)

- SMTP (used to send email)
- GD2 (optional; required only if the [GD version](#) of the confirmation system is used)
- If you are building PHP on Linux, the hash module needs to be enabled by adding the 'hash' USE flag to the compilation command.

### B.3.3. Installing web server components

The following steps must be performed on the web server to install the account creation system:

- Download the archive containing the account creator files and extract them.
- Copy the "website" folder to a location accessible by the web server, such as the document root (htdocs on Apache and wwwroot on IIS).
- To configure the database connection, modify `includes/SAMPLE_config.inc.php` and rename it to `config.inc.php`. If the file already exists, this step is not necessary.
- If there is already a shared index file, the following code should be copied into the footer area directly below the copyright notice to enable custom footer text to be inserted by the account request form. While not strictly necessary, it reduces the chance of legal problems.

```
<?php    if    (isset($subst)    &&    isset($subst['customFooter']))    echo
$subst['customFooter']; ?>
```

- If you would like to use the index file that has been provided, rename `SAMPLE_index.php` to `index.php`.
- Similarly, you may either use your own home page or rename the provided `SAMPLE_home.inc.php` to `home.inc.php`.
- Log into MySQL Server and load `pgportal.dump` into the same database you entered in `config.inc.php`.
  - For example, you might type use `pgportal; source pgportal.dump;`

- You may also wish to modify `includes/access_request_config.php` to customize various aspects of the account creation system.

### ***B.3.3.1. Configuring Apache***

If Apache is being used, SSL must be configured if you want the server to be able to accept HTTPS requests. Using HTTPS for account creation and the administration console will increase the difficulty for an attacker to intercept sensitive data. To enable the built-in SSL module, the following lines in `conf/httpd.conf` should be uncommented:

- `LoadModule ssl_module modules/mod_ssl.so`
- `Include conf/extra/httpd-ssl.conf`

Additionally, you will need to acquire an X.509 certificate and private key for the server. The default configuration requires that these be placed in the `conf` directory. Alternatively, the certificate and private key may be located elsewhere if `conf/extra/httpd-ssl.conf` is modified to point to their location.

*Important: In order for the account creation system to take advantage of SSL, the `ENABLE_SSL` option in `includes/access_request_config.php` must be set to true.*

### ***B.3.3.2. Configuring the Firewall***

If the server is located behind a firewall, the firewall must be configured to allow incoming connections on the following ports. Otherwise, users outside the firewall may have difficulty making account requests or accessing the administration console.

- 80: used by HTTP
- 443: used by HTTPS

### B.3.4. Installing Portal Server Components

If you do not already have a portal, installation instructions may be found at the [P-GRADE Portal website](#) by selecting "Install the portal" on the menu.

The following steps must be performed on each portal server that will be used with the account creation system:

- Create two new roles: `SERVLET_CLIENT` and `ACCOUNT_CREATOR`.
  - To create a role, log in as a super user (such as 'root'), navigate to the Administration tab, click on "Roles" to get to the Role Manager portlet, and then click on "Create New Role". At this point you should be asked to edit role information. Enter `SERVLET_CLIENT` or `ACCOUNT_CREATOR` for the role name and whatever you want for the description. The description is not necessary, but may be useful in case you forget the role's purpose.

The screenshot shows the P-Grade Portal interface. At the top left, it says "RELEASE 2.6". The main header features the P-Grade logo, the text "P-GRADE", and a "portal" logo with an arrow pointing to the right. On the top right, there is a "Logout" link and a welcome message: "Welcome, Matthew Reiter". Below the header is a navigation menu with tabs: "Welcome", "Administration", "Workflow", "Certificates", "Settings", "Information System", "File Management", and "Help". The "Administration" tab is selected. Below the navigation menu is a "Portlets" section with links for "Users", "Groups", "Roles", "Layouts", and "Messaging". The "Roles" link is selected, leading to the "Role Manager" portlet. The portlet title is "Role Manager" and it contains a "Display All Roles" section. This section displays a table of roles:

Role Name	Role Description	Delete Role
USER	The standard user role	
ADMIN	Offers ability to add/delete users from a group	
SUPER	The portal administrator used for creating/deleting users, group, roles and layouts	
<a href="#">SERVLET_CLIENT</a>	Grants the ability to use portal servlets.	<input type="button" value="Delete"/>
<a href="#">ACCOUNT_CREATOR</a>	Grants the ability create new user accounts.	<input type="button" value="Delete"/>

At the bottom of the portlet, there is a "Create New Role" link.

**Figure 6: Role manager**

- Create two new user accounts, one which is given only the `SERVLET_CLIENT` role and one which is given only the `ACCOUNT_CREATOR` role. The web server will use these users to authenticate itself with the portal. Although the user names and passwords for these roles may be the same across all of the portals, for security reasons it is recommended that different user names and passwords be used for each portal.
  - To create a user account, navigate to the User Account Manager portlet from the Administration tab by clicking on "Users" near the top. From there, click on "Create a New User" and fill in the information. You can choose whatever user name and password you like. Select either `SERVLET_CLIENT` or `ACCOUNT_CREATOR` (but not both) as the user's role. Giving each role to a different user minimizes the damage that can be done if an attacker discovers an account's user name and password. Additionally, you may check the "disable account" checkbox to prevent the account from logging into the portal.

Portlets Users Groups Roles Layouts Messaging

? User Account Manager

## Edit User Information

*LEAVE PASSWORD FIELD BLANK TO KEEP EXISTING PASSWORD IF EDITING AN EXISTING USER*

User Name:

Full Name:

Email Address:

Organization:

Disable account?

Select Roles	Role name
<input type="checkbox"/>	USER
<input type="checkbox"/>	ADMIN
<input type="checkbox"/>	SUPER
<input checked="" type="checkbox"/>	SERVLET_CLIENT
<input type="checkbox"/>	ACCOUNT_CREATOR

Password:

Confirm password:

Figure 7: User manager

## B.4. Administration Console

The administration console serves as the portal for various administrative activities, including creating new user accounts and editing grid information.

The default administrator user name is "admin" and the default password is blank. It is recommended that you change the user name and password using the setting editor the first time you log in.

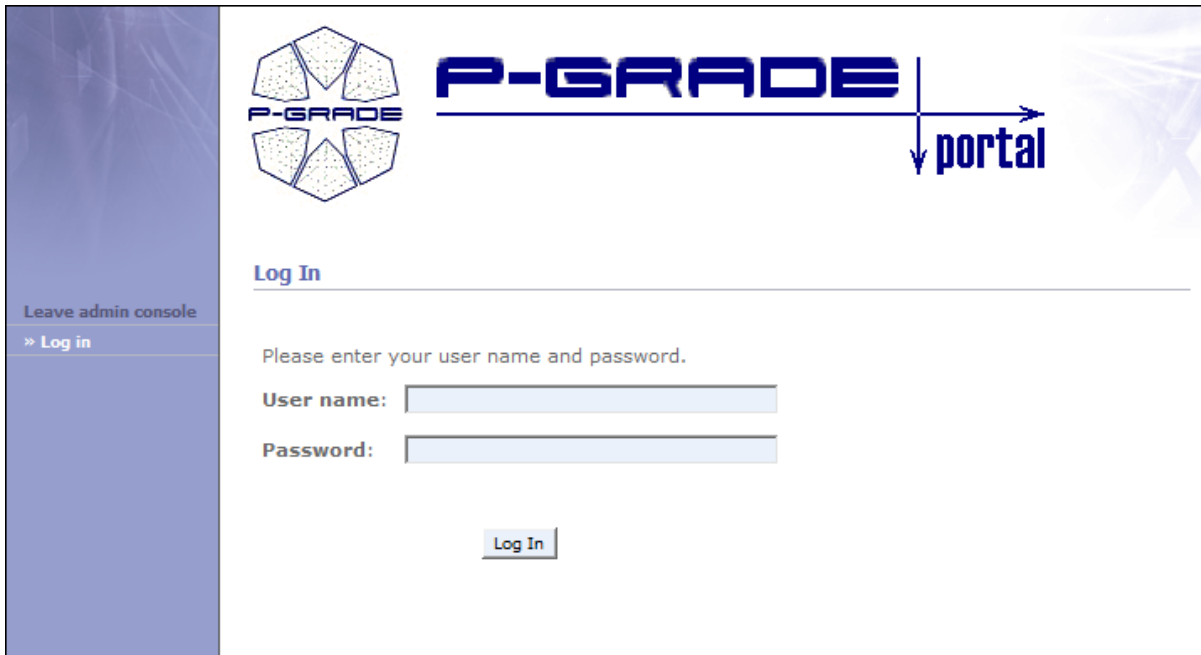


Figure 8: Administration console login

#### B.4.1. Exporting User Account Information

The user information exporter provides a means for the administrator to download user information for the purpose of sending mass emails, etc. Clicking the "export" button in the form will collate and download the user information for all grids managed by the account creation system. The information is provided as a comma-separated-value (CSV) file that is compatible with most spreadsheet software.



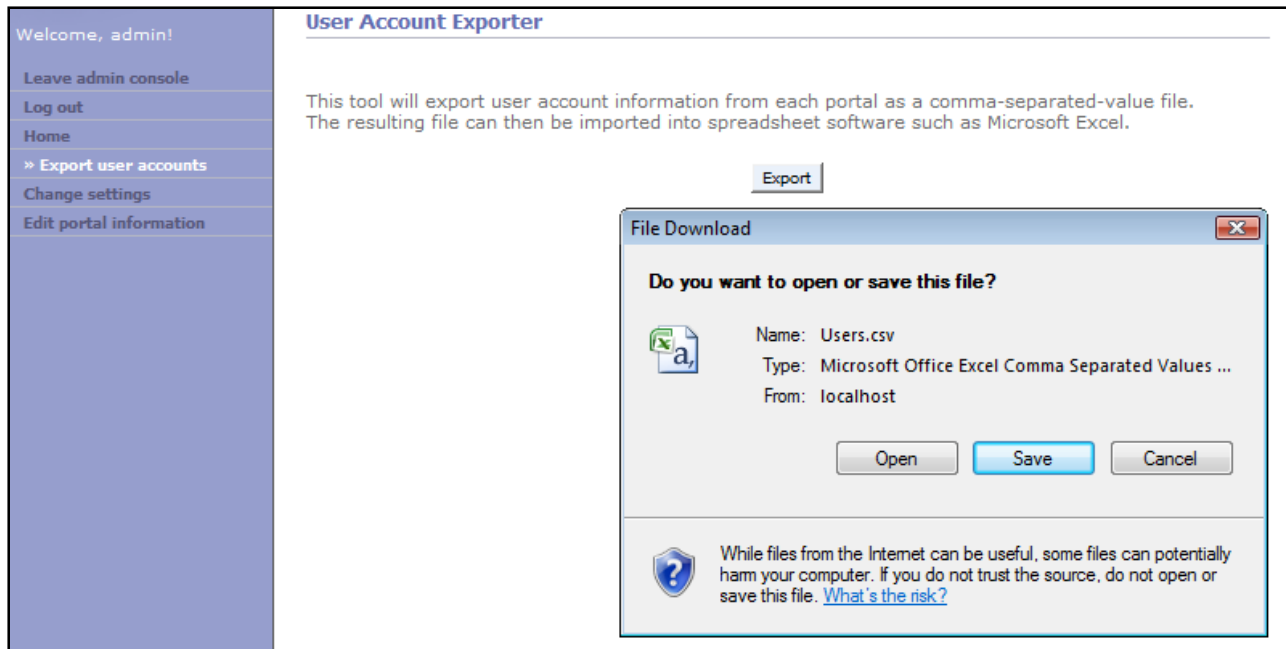


Figure 9: User Information Exporter

## B.4.2. Settings

The settings editor may be accessed by selecting "change settings" from the menu in the administration console.

Settings affect the operation of the account creation website. The following settings are available:

Name	Type	Description
Require confirmation	boolean	If set to true, the user must enter a confirmation code when requesting an account.
Use GD confirmation	boolean	If set to true, the GD version of the confirmation image is used. This version is better than the non-GD version, but requires that

		the GD2 extension be enabled.
GD confirmation foreground noise	boolean	If set to true, foreground noise is used to make the GD-based confirmation harder.
GD confirmation X grid	integer	The average number of pixels between horizontal grid lines in the GD-based confirmation.
GD confirmation Y grid	integer	The average number of pixels between vertical grid lines in the GD-based confirmation.
Session lifetime	integer	The lifetime of a session, in hours. This only applies to the account request page.
Servlet authentication key	string	The key used to encrypt the servlet user name and password; should be a unique (preferably random) string that is difficult to guess.
Admin user name	string	The administrator's user name.
Admin password	string	The administrator's password. This value cannot be retrieved, as it is stored in a hashed form in the database.
Admin email address	string	The administrator's email address.

Welcome, admin!

Leave admin console

Log out

Home

Export user accounts

» Change settings

Edit portal information

## Settings

---

**Require confirmation**

**Use GD confirmation**

**GD confirmation foreground noise**

**GD confirmation X grid:**

**GD confirmation Y grid:**

**Session lifetime:**

**Servlet authentication key:**

**Admin user name:**

The current administrator password must be set in order to change the user name or password. Leaving the 'admin password' and 'confirm admin password' fields blank will preserve the current password.

**Current admin password:**

**Admin password:**

**Confirm admin password:**

**Admin email address:**

Figure 10: Settings

### B.4.3. Editing Portal Information

The portal information editor may be accessed by selecting "edit portal information" from the menu in the administration console.

Portals can be added or removed using the buttons near the bottom of the form. To add a portal, click on "Add New Portal", which will add a group of fields where the portal information may be entered. To remove one or more portals, select all of the portals you wish to delete by clicking on the check boxes next to their names. Click on "Delete Selected Portals", review the portals listed in the dialog box, and then press "Okay" to confirm or "Cancel" if there was a mistake. Once a portal is deleted, its corresponding fields will be removed from the form. Keep in mind that no changes are made on the server until the form is submitted.

Clicking on the "Reset" button will reset all fields to their original values; however, it will not undo addition or removal of portals. If you wish to undo such changes, you may do so by refreshing the page.

The following fields are available for each portal:

**Portal ID**            The name that is used internally when referring to a given portal. Frequently changing the portal ID is not recommended.

**Portal name**        The name that is displayed to users.

**Portal URL**            The URL that points to the portal's root directory (for example, `https://portal.organization.com/szupergrid`).

**Servlet user name**    The user name of a portal user that has been given the ACCOUNT\_CREATOR role.

**Servlet password**     The password corresponding to the servlet user name.

**Account creator user name**    The user name of a portal user that has been given the SERVLET\_CLIENT role.

**Account creator password**    The password corresponding to the account creator user name.

Figure 11: Portal information editor

#### B.4.4. Account Request Verification

When a user submits an account request, an email, shown in Figure 12, is sent to the administrator with the details of the request and a link to verify the account request. Clicking on the link sends the administrator to a web page, shown in Figure 13, with options to approve or deny the account request in addition to a text box where the administrator may enter additional comments to send to the user who request the account. The administrator has four options for how to handle an account request: accept the request, deny the request, accept the request but with the roles and groups reset to their default values, or ignore the request. The third option is provided in case a user requests roles that he or she should not have, but the administrator feels that the request is otherwise valid. A request should be ignored if the administrator does not wish to send any notification to the user or if the email address provided by the user is invalid. Once the administrator submits the form, the server forwards the account request to each portal listed in the request and displays any error messages that were returned. If the account request succeeds on all portals, a success message, shown in Figure 14, is

displayed. However, if the account request fails on one or more portals, the administrator is presented with an option to resend the account requests to each portal that returned an error message, as shown in Figure 15. The emails a user receives when an account request is accepted or denied are shown in Figure 16 and Figure 17, respectively.

## P-GRADE Portal Access Request Verification

P-GRADE Grid Portal Team [portalreq@lpds.sztaki.hu]

Sent: Mon 4/21/2008 10:47 AM

To: mreiter@127.0.0.1



**P-GRADE Portal Access Request Verification**

Full name: *Joe Shmoe*  
User name: *Joe*  
Password: *password*  
Address: *Nowhere*  
Phone number: *12345*  
Institute: *College of Testing*  
E-mail address: *mreiter@127.0.0.1*  
Roles: *Optional Role*  
Groups: *Optional Group*  
Portals: *SEE-GRID*  
Scientific case and planned usage:  
*Grid will not be used.*

Date: *2008-04-21*  
Time: *10:47:01*

**Accept/Deny**

Figure 12: Account request verification email

Welcome, admin! Leave admin console Log out Home Export user accounts Change settings Edit portal information	<h3>Account Request Verification</h3> <hr/> <p><b>Account request information:</b> Full name: <i>Joe Shmoe</i> User name: <i>Joe</i> Password: <i>password</i> Address: <i>Nowhere</i> Phone number: <i>12345</i> Institute: <i>College of Testing</i> E-mail address: <i>mreiter@127.0.0.1</i> Roles: <i>Optional Role</i> Groups: <i>Optional Group</i> Portals: <i>SEE-GRID</i> Scientific case and planned usage: <i>Grid will not be used.</i></p> <p><b>Accept the request?</b></p> <p><input type="radio"/> Accept <input type="radio"/> Deny <input checked="" type="radio"/> Accept with default roles and groups <input type="radio"/> Ignore (no email will be sent to the user)</p> <p><b>Message to user (optional):</b></p> <div style="border: 1px solid #ccc; padding: 5px; min-height: 40px;">Membership in "Optional Group" is restricted to members of SZTAKI.</div> <p style="text-align: center;"><input type="button" value="Continue"/></p>
---	--

Figure 13: Account request verification form

<h3>Account Request Verification</h3> <hr/> <p>Account created on SEE-GRID.</p> <p>The account request has been successfully processed.</p>
---

Figure 14: Account request verification success

<h3>Account Request Verification</h3> <hr/> <p>Failed to create an account on SEE-GRID!</p> <p>The account request was not successfully processed.</p> <p><b>Account request information:</b> Full name: <i>Joe Shmoe</i></p>
---

Figure 15: Account request verification failure

## P-GRADE Portal Access Request Accepted

P-GRADE Grid Portal Team [portalreq@lpds.sztaki.hu]

Sent: Mon 4/21/2008 11:10 AM

To: mreiter@127.0.0.1



© Copyright 2004-2008, MTA-SZTAKI LPDS, Hungary. All rights reserved.  
->> webmaster@lpds.sztaki.hu

Figure 16: Account request accepted

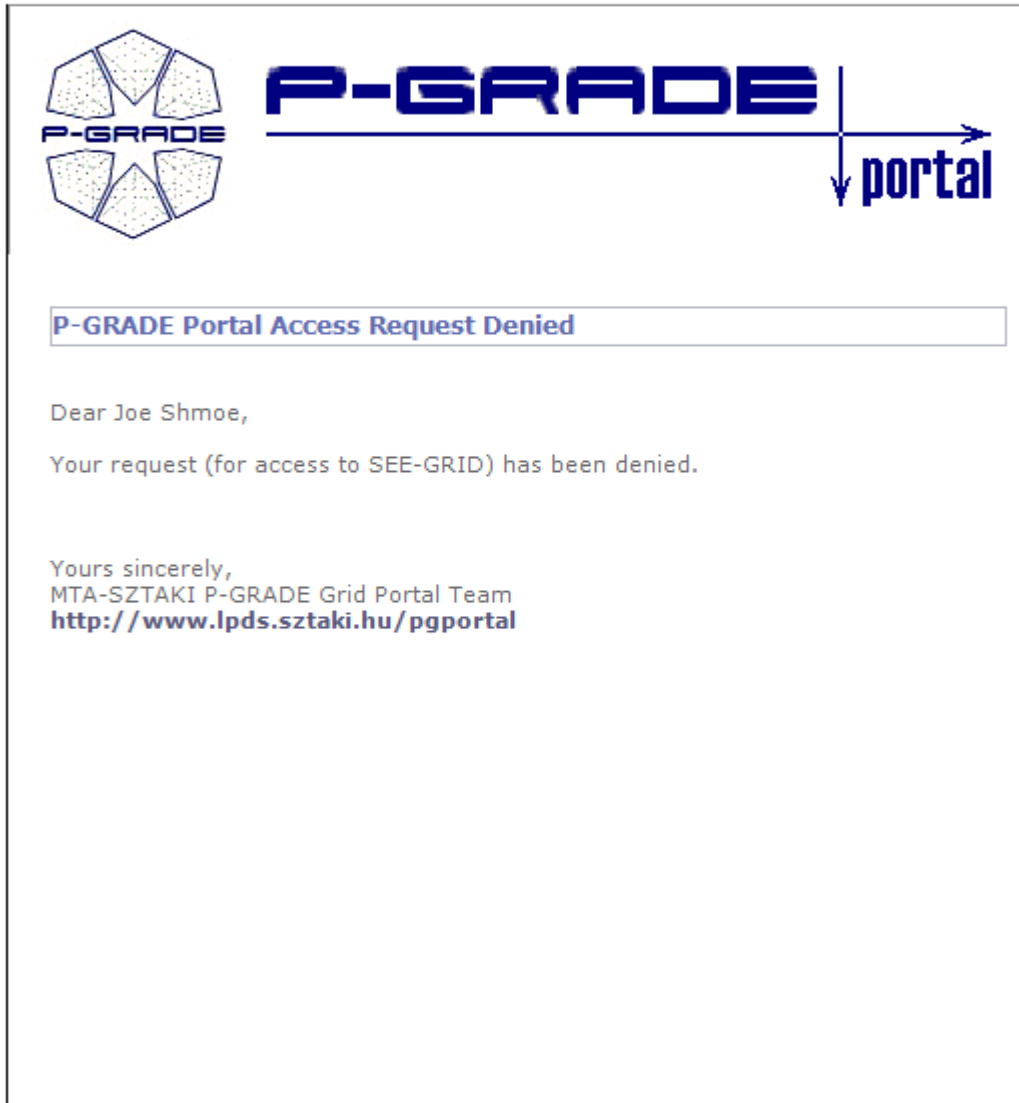


## P-GRADE Portal Access Request Denied

P-GRADE Grid Portal Team [portalreq@lpds.sztaki.hu]

Sent: Tue 4/15/2008 3:08 PM

To: mreiter@127.0.0.1



© Copyright 2004-2008, MTA-SZTAKI LPDS, Hungary. All rights reserved.  
->> webmaster@lpds.sztaki.hu

Figure 17: Account request denied

## B.5. Other Features

### B.5.1. Logging

A message is sent to the system log every time one of the following events occurs:

- A user submits an account request
- A user account is created
- An administration login is attempted
- There is an error connecting to a servlet

On Windows, the Application log is used instead of the system log. The application log may be viewed by opening the Computer Management console and navigating to System Tools -> Event Viewer -> Windows Logs -> Application. Log messages from the account creation system can be identified by looking in the "Source" column for "PHP-" followed by the version of PHP installed. For versions of Windows other than Windows Vista, a slightly different procedure may be required. An account request event can be seen in Figure 18.

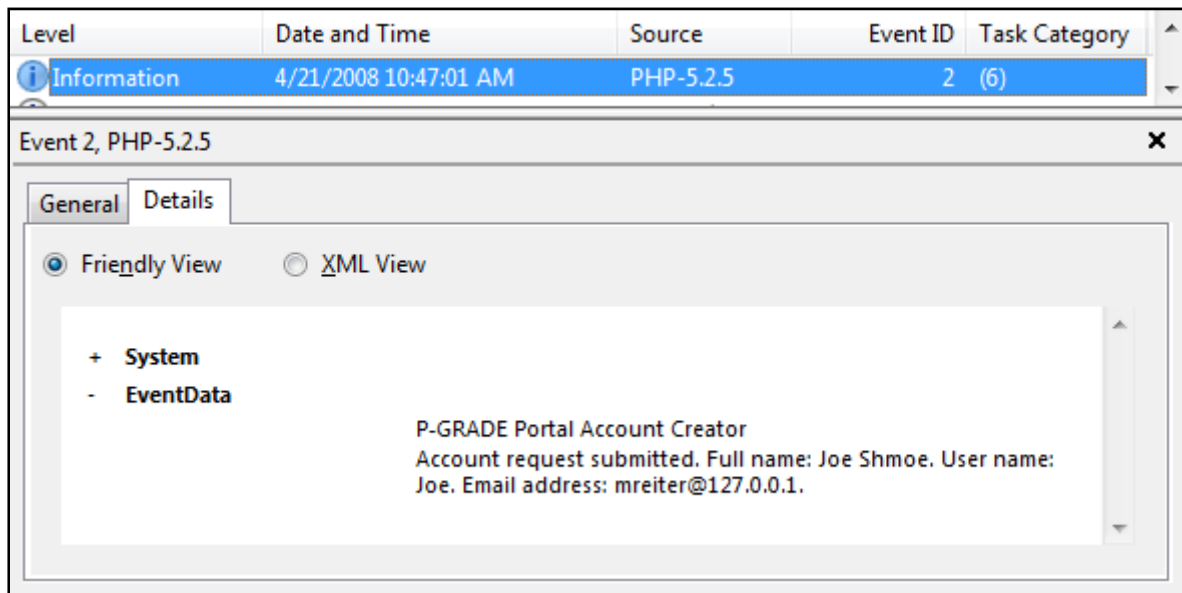


Figure 18: Account request event

## C. Email Notification System Help Documentation

### Introduction:

The user may instruct the system to send notification(s) about given state changes caused by the submission of a **workflow**. Two kinds of changes are distinguished:

- The **workflow** reaches an end state (“**finished**” or “**error**”) from where it may not be moved without manual user action.
- The state of the **workflow** has changed (for example from "**submitted**" to "**running**", or from "**running**" to "**finished**", etc.)

At present, the only way of the notification is an **e-mail message**.

The **tab Workflow/Notify** for configuring these messages is structured the following way:

#### I.

**E-mail Settings** groups the base information needed to send an e-mail involving the **recipient's address**, the **subject** of the message and the overall permission to send any letter.

#### II.

**Workflow Change Settings** is the editable skeleton of the letter sent in the case of a change of state (see above).

The **user** has a further filter possibility to **disable /enable** these letters by the selecting the proper value of the checklist **Enabled:**

As a summary, the sending of a letter has five conditions controlled by the user:

1. The proper e-mail address is set
2. **Sending an email** is enabled

3. **Sending for workflows** is enabled
4. Upon submitting the workflow, the user does not choose “**Never**” when asked when they would like to be notified
5. The event listened for has occurred

The content of the message is freely editable in the text area **Message**.

All **keys** will be evaluated at the time of the email being sent, and will be appropriately substituted into the letter.

These keys and their meanings are:

- #now# Time stamp of event
- #user# Owner of the **workflow**
- #portal# URL of the **portal**
- #workflow# Name of **workflow**
- #oldsatus# State prior the event
- #newsatus# State caused by the event
- #details# Detailed listing of job statuses

### **III.**

The settings must be saved by clicking the **Save button**