CS-GXS-0502

GRID Portal Application Visualization

submitted to the Faculty

of the

Worcester Polytechnic Institute

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

_____

Domenic Giancola
Date:

_____

Amanda J Jamin
Date:

_____

Professor Gábor Sárközy, Major Advisor

Abstract

Parameter studies are useful applications for researchers; however, these programs, although helpful, tend to be computationally expensive and due to their long execution time become tedious to execute. In this project we explored a method of implementing a parameter study module for the P-GRADE Portal at MTA-SZTAKI; Budapest, Hungary, an existing parallel application that allows users to create and execute a parallel program in an efficient manner without knowledge of MPI or PVM programming.

Acknowledgments

We would like to thank MTA SZTAKI, and specifically the Laboratory of Parallel and Distributed Systems, for giving us this opportunity to develop an extension of their P-GRADE Portal. Specifically, we would like to thank József Patvarczki and Gábor Hermann for their continued help and support during the development process. We would also like to thank WPI and the WPI Interdisciplinary and Global Studies Division for helping to provide this opportunity. Most of all we would like to thank Professor Gábor Sárközy for making this entire project and experience possible.

TABLE OF CONTENTS

ABSTRACT **2**

ACKNOWLEDGMENTS **3**

LISTING OF TABLES **6**

1 PROJECT STATEMENT **7**

1.1 THE EXISTING SYSTEM 7
1.2 THE NEED FOR A PARAMETRIC STUDY MODULE 8
1.3 OUR END RESULTS 8

2 BACKGROUND **10**

2.1 HISTORY OF GRIDS AND GRID COMPUTING 10
2.2 HISTORY OF PARALLEL PROGRAMMING 13
2.3 PGRADE 16
2.3.1 What is PGRADE *17*
2.3.2 How PGRADE Works *17*
2.3.3 Uses of the PGRADE System *19*
2.4 THE PGRADE PORTAL 20
2.4.1 How the PGRADE Portal Works *21*
2.5 PARAMETER STUDY 22
2.5.1 Uses of Parameter Study Programs *22*
2.5.2 Implementation of Parameter Study Applications *23*
2.5.3 Complications with Implementation *24*
2.5.4 Other Parameter Study Applications *25*
2.5.4.1 Nimrod 25
2.5.4.2 ILAB 26

3 METHODOLOGY **27**

3.1 TECHNOLOGIES INVOLVED IN IMPLEMENTATION 27
3.1.1 Java Server Pages *27*
3.1.1.1 How JSP Works 28
3.1.1.2 Alternatives to JSP 29
3.1.2 Portlet Technology *29*
3.1.3 GridSphere *30*
3.1.4 Java Swing *31*
3.1.4.1 History of Swing 31
3.1.4.2 The Use of Swing Within the Portal 32
3.2 EDITING SECTION 33
3.3 THE PARAMETER STUDY MANAGER 36
3.3.1 Development of the Parameter Study Manager *39*

4 IMPLEMENTATION **41**

4.1 IMPLEMENTATION OF THE WORKFLOW EDITOR SECTION 41
4.1.1 Parameter Set Generation Dialog *41*
4.1.2 Remaining Dialog Extensions *43*
4.1.3 Integration of the Editing Section *45*
4.2 IMPLEMENTATION OF THE PARAMETER STUDY MANAGER 47
4.2.1 Creating a Communication Method Between the JSPs and the Portlet *47*
4.2.2 Determining the Range of Indexes for Submission *48*
4.2.2.1 Evaluation of the Range of Indexes Algorithm 50
4.2.3 Creation of Reusable Code *51*
4.2.4 Displaying the Current State of the Parameter Study *52*
4.2.5 Removal of Designed Features *53*

4.2.6 Integration *54*

5 TESTING **55**

5.1 EXTERNAL TESTING 55
5.2 CREATION OF A LOCAL PSTUDYJOBLIST 56
5.3 CREATION OF A LOCAL PARAMETRICWORKFLOW OBJECT 57
5.4 MONITORING OF TRACE FILES 57
5.5 INTEGRATION TESTING WITH THE MANDELBROT SET 58

6 CONCLUSIONS **60**

6.1 THE FINAL PARAMETER STUDY MANAGER 60
6.2 THE FINAL PARAMETER STUDY EDITING SECTION 61
6.3 FUTURE IMPLEMENTATION 61
6.3.1 The Detailed View of a Single Index Set *61*
6.3.2 Full Integration of the Editing Section *62*
6.4 RECOMMENDATIONS FOR SZATKI 62
6.4.1 Development of a Sztaki Defined JSP Tag Library *62*
6.4.2 External Documentation *63*
6.4.3 Code Modularity *63*

APPENDIX A ALGORITHM TO DETERMINE THE RANGE OF INDEXES FOR SUBMISSION ALGORITHM **68**

APPENDIX B EXTERNAL TESTING CODE FOR THE PARAMETER STUDY MANAGER **69**

APPENDIX C SCREENSHOTS OF THE FINAL PARAMETER STUDY MANAGER **70**

Listing of Tables

TABLE 1 JSP TAGS 28

LISTING OF FIGURES

FIGURE 2.1 SAMPLE WORKFLOW IN THE P-GRADE PORTAL WORKFLOW EDITOR 16
FIGURE 2.2 DIAGRAM DESCRIBING THE INTERCOMMUNICATIONS OF THE PGRADE PORTAL 21
FIGURE 0.1 VIEW OF A CERTIFICATE DOWNLOAD……..23
FIGURE 3.1 SUGGESTED PARAMETRIC INPUT PORT REPRESENTATION IN THE WORKFLOW EDITOR WINDOW 33
FIGURE 3.2 SUGGESTED PARAMETRIC EXTENSION OF THE PORT PROPERTY DIALOG 34
FIGURE 3.3 SUGGESTED PARAMETRIC EXTENSION OF THE JOB PROPERTY DIALOG 35
FIGURE 3.5 PROPOSED PARAMETER STUDY MANAGER DESIGN 37
FIGURE 3.6 PROPOSED PARAMETER STUDY MANAGER DETAILED VIEW 38
FIGURE 4.1 IMPLEMENTED PARAMETER SET GENERATION DIALOG 42
FIGURE 4.2 IMPLEMENTED PARAMETRIC PORT PROPERTY DIALOG 44
FIGURE 4.3 IMPLEMENTED PARAMETRIC JOB PROPERTY DIALOG 45
FIGURE 4.5 KEYS FOR INDICATING THE STATUS OF A PARMATER STUDY 52

1 Project Statement

Our project aims to develop an interface available via the World Wide Web that will allow users of the system to execute parallel parametric study applications on grid computing resources. This interface will exist as a part of a collaborative project between two research teams who together will produce a Parameter Study Module for an existing parallel process development tool. This tool is the PGRADE Portal system currently being developed by the SZTAKI labs in Budapest, Hungary.

1.1 The Existing System

The fully integrated PGRADE project exists in two distinct parts: the PGRADE System and the PGRADE Portal. Currently these two projects support the creation, execution and monitoring of parallel applications. The main distinction between the two portions is where they can be executed. The PGRADE system is not available via the web whereas the PGRADE Portal, through the use of Portlets, can be accessed by anyone with an Internet connection and permission to execute a process via the application.

This project will work mainly with the PGRADE Portal. This Portal currently allows users to create and edit workflows though a WebStart application called the Workflow Manager. Once designed, these workflows can be scheduled for submission, and their progress monitored. In addition this system allows the user to choose the grid or grids on which to run their application. Multiple users can access the software, each with their own user name and password, regulated by a single administrator. As all of this is done via a graphical web interface, a user of the PGRADE Portal does not have to be a master at parallel programming to benefit from its use.

1.2 The Need for a Parametric Study Module

Although the existing PGRADE Portal system allows a user to graphically define workflows and execute programs in parallel, it has yet to reach its full potential. Currently, the Portal is limited to those applications that are not parametric applications. A parametric study is a program that executes numerous times, each time with a different set of input values. Although there is a vast area of applications that fall into this category, such as some weather monitoring systems, traffic simulations, and certain chemistry applications, the Portal can be extended to include parametric studies. Parametric applications are useful to mathematical and scientific researchers. However, they are often time consuming to calculate making them tedious to work with. If these programs were executed in a parallel manner on a grid system, their time consuming nature would diminish.

The PGRADE Portlet offers the perfect environment to develop a tool to use for the execution of these parameter studies. As the Portal already allows for programs to be executed in parallel on a variety of resources, applying the project to parameter study problems simply requires an extension of the current application. This module would not only further satisfy the needs of the current users of the Portal, but also create a whole new user group who could use the application.

1.3 Our End Results

At the conclusion of this project, we aimed to have created a complete user interface for the Parameter Study Module for the PGRADE Portal system. In order to have successfully implemented this project, we needed to coordinate our project with the

Development of Algorithms on the GRID MQP team. Our project, although a separate entity, needed to integrate with this project in order to have a full module to present to the PGRADE Portal development team.

Prior to our integration, however, we had to first develop two distinct user interfaces: a client side Workflow Editor for parameter study applications and a server side Parameter Study Monitor. The Workflow Editor interface required extending the current PGRADE Portal Workflow Editor so that allowed the user to define the ports through which the application can be run and describe the parametric keys related to the study. A more in depth discussion of the implementation of these interfaces will occur later in this paper.

The Parameter Study Monitor was created as its own set of Java classes and Java Server Pages (JSP). The tabbed pane that was created as a result of this interface looks similar to the existing PGRADE interface. This interface will also be discussed in more detail in the sections that follow.

2 Background

In order to understand both the PGRADE and PGRADE Portal system and how they work, one must first understand the technologies that these systems implement. These systems allow a user to take advantage of grid computing, a relatively new research area in the computer science field. In addition, they allow a user to break away from the traditional sequential programming style and develop code that runs in parallel enabling one to use resources on several systems simultaneously. Furthermore, for one to fully understand why a Parametric Study would benefit the development of the PGRADE Portal, how the PGRADE system and the PGRADE Portal work must be described and the need for parameter studies explained.

2.1 History of Grids and Grid Computing

The Internet, otherwise known as the World Wide Web, has grown by leaps and bounds since its invention. Every day people surf around for both business and pleasure, interconnecting millions of computers for a variety of purposes. Much like how the Internet has expanded, the average household has also expanded into this new age. Many more households than ever before have a personal computer (PC). Computer technology has grown so quickly that a mid to low range PC loses more than 90% of its value every year. Thus as organizations and even individuals upgrade their machines, their old machine, being worth very little, is either given to another person, thrown away, or put into storage. [8]

One new technology that is currently in development by researchers and software engineers around the globe is a service by which these machines can be put to use, along

with the surplus resources of other PCs and supercomputers. This new service would pull these resources together into a grid. The grid would be much like the World Wide Web, but it would be for sharing computer processing power and data storage instead of information. This would greatly help alleviate many of the resource problems researchers and scientists have had solving complicated problems that a simple cluster of computers or even a supercomputer just cannot handle. The eventual dream many people have for the grid is that it will connect every computer in the world, dynamically allocate whatever resources the user may need, handle security and permissions issues for them, and basically allow them to work with a "limitless" supply of computing power.

This resource could help biologists simulate thousands of molecular drug candidates to see how they react with specific proteins. Physicists could use it to create and share the 10 Petabytes (10^15 bytes) of data they will be gathering from the collisions of extremely energetic fundamental particles. Earth Scientists could use it to help keep track of the levels of atmospheric ozone. Geneticists could use it to help analyze the massive quantities of data necessary to map the human genome. The grid is what every researcher has been dreaming of for years. [8]

While that is the dream, reality and the dream have not yet met, though it is definitely coming. Most of the grid is still in the developmental stages, being worked on by researchers all over the globe. As such there are many different grid implementations. A few example projects are:

- Development of Virtual Supercomputing Service Using Academic Network [26]
- Automatic Performance Analysis: Real Tools (APART2) [23]
- GridLab - A Grid Application Toolkit and Testbed [35]

- DataGrid Research and Technological Development for an International Data Grid [16]
- Graphical Supervising System for Geographically Distributed, Heterogeneous Metacomputing Environment [28]
- Simbex: A metalaboratory for the priori simulation of crossed molecular beams experiments [17]
- DemoGrid [5]
- Hungarian Supercomputing GRID [29]
- Chemistry GRID and its Application for Air Pollution Forecast [27]
- JGrid: An Integrated Graphical Application Development and Grid Execution Environment Based on Jini [25]
- EGEE: Enabling Grids for E-sciencE [13]

As is quite evident even with only this sampling of grid projects, grid computing is a hot topic in the technological world. Consortiums of developers have created standards to help speed up and organize the process. These standards allow researchers to build off of each other much easier and quicken the pace towards the ultimate goal of a global Grid.

One such project was the Condor project. It was started in 1988 at the University of Wisconsin to develop a system for pooling the computing resources of all of the computers in a university. Condor paved the way for later implementations in that it performed cycle scavenging (using free time on computers for processing), found appropriate resources, and recovered from faults. Newer versions of Condor are still in use today in grid computing. One such implementation is Condor-G, which incorporates some tools from the Globus Toolkit to allow it to submit jobs to the current grid implementation. [48]

The Globus Toolkit is the major backbone of protocols and services used by most current Grid implementations. It is being developed by the Globus Alliance, a cooperative effort of many programmers from around the world. It is being released as an open-source software suite to allow others to use, update, and modify the software to their needs and then share their improvements with the rest of the world. Another benefit to the Globus Toolkit is that it is being developed through an object-oriented approach. This allows developers to choose only the portions of the toolkit that they wish to use in their application. [46]

With the development of these grid projects, developers were no longer restricted to using only homogenous clusters as this new infrastructure allowed for heterogeneous resources comprised of supercomputers and PC clusters to be tied together into true grids. This heterogeneous grid infrastructure introduced quite a few new problems into the grid scene, with the main problem being that at the time they did not have the programming models or libraries necessary for this type of setup. [8]

2.2 History of Parallel Programming

To use a new technology like the Grid, or any other cluster of computers for that matter, new programming methods and standards must be developed and followed. Programming was originally only sequential, in that a program started on a machine, did what it was programmed to do, and then exited. The program may have communicated with other programs during its execution, but usually only as different pieces to complete a task. Parallel programming allows the user to execute one task in pieces on multiple machines to reach a result sooner than would be possible on a single machine. [44]

Two main types of parallel programming models exist, data parallel and function parallel. In a data parallel model processors operate on different pieces of a data set, sharing results with one another. In a function parallel model a parent process manages a set of child processes that each perform a given task and then give their results back to the parent for processing. [44]

There are two main implementations of these programming models currently in use- MPI, Message Passing Interface, and PVM, Parallel Virtual Machine. Both implementations are message-passing libraries, but they differ in their approaches and functionality. MPI was created by the MPI Forum, a group of vendors, scientists, and users. It permits programs with separate address spaces to synchronize with one another and move data from the address space of one process to that of another by sending and receiving messages. MPI is not technically a language, but it is rather a collection of subroutines and their arguments. MPI works very efficiently on supercomputers and inside clusters. As such MPI is typically used for Single Program Multiple Data (SPMD) style programming. SPMD is a programming model where different processors execute the same program on different sets of data. However, if an MPI application is distributed on several Grid sites the performance turns out to be much worse than expected. It is in these environments that the second implementation, PVM, really shines. [47]

PVM, or Parallel Virtual Machine, is another message passing library like MPI and is distributed with a set of tools used to create and execute concurrent and parallel applications. PVM, while performing the same general function as MPI, does so in more of an abstract and parallel manner. It is this main difference that allows PVM to be used more for Multiple Program – Multiple Data (MPMD) style programming. MPMD is a

parallel programming model where different, but related, programs are run on different sets of data. [41]

By comparing the two implementations, PVM and MPI, it becomes obvious that in a heterogeneous environment like a grid, an integration of both MPI and PVM is necessary for any complex problem. MPI can be used locally within a single cluster to cut down on overhead with PVM pulling together the general processing nodes together. This is only one possible model though out of numerous possibilities. This new blended model is what many researchers and developers are currently working to grasp. [31]

It is at this new blended model level that the workflow concept has been created. A workflow is an acyclic dependency graph with the nodes representing jobs and arc connections between the jobs defining dependency relations. This new model introduces parallelism at two levels. Top-level parallelism comes from the workflow concept, where independent branches of a workflow can be executed simultaneously on several Grid sites. Bottom-level parallelism is applied when some of the nodes are themselves parallel (MPI or PVM) programs. This new blended model allows for multi-site parallel application execution without the performance degradation inherent in using solely one implementation. [31]

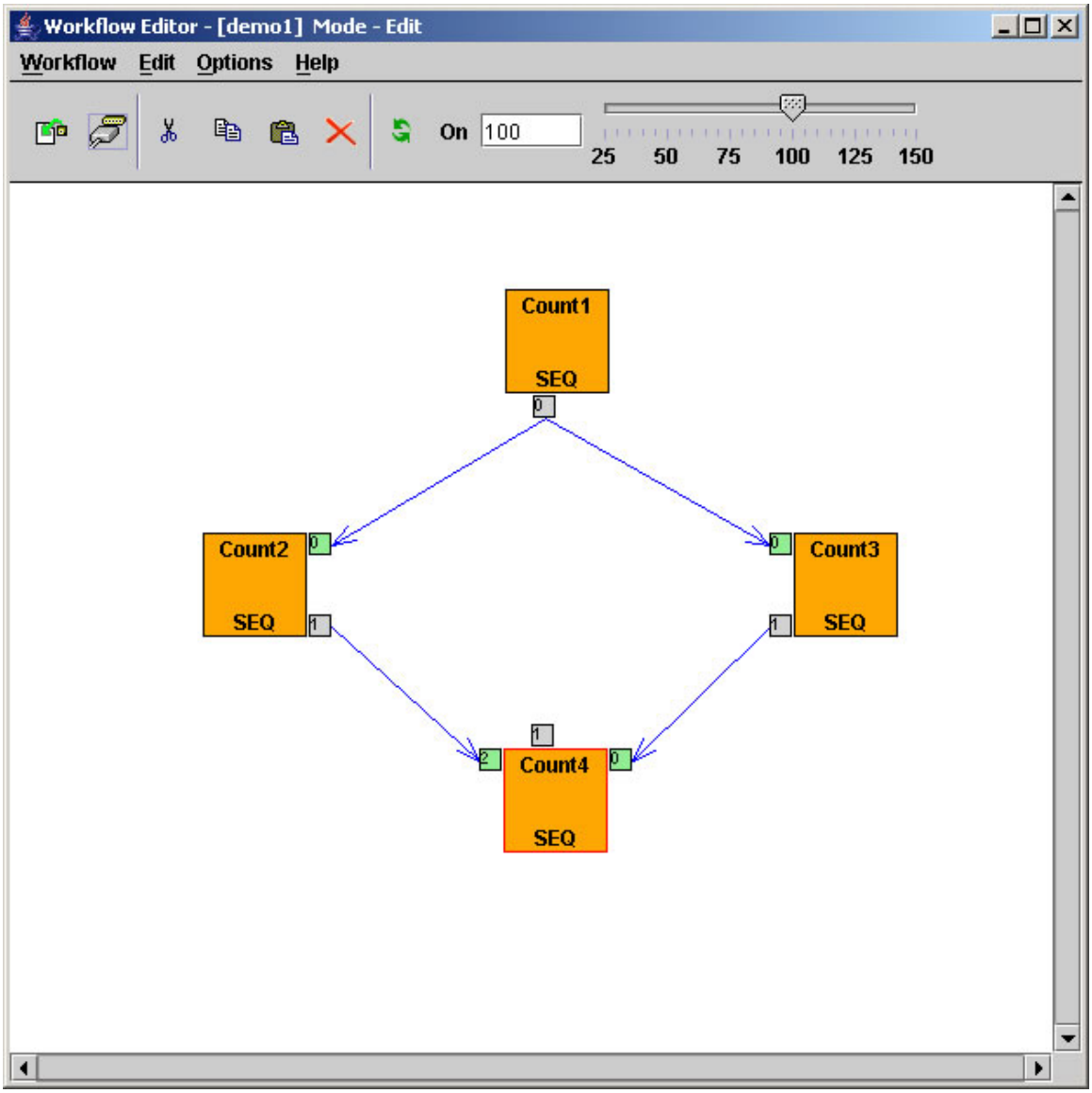


Figure 2.1 Sample workflow in the P-GRADE Portal Workflow Editor

2.3 PGRADE

Located in the Hungarian capital, Budapest, there exists an institute dedicated to research in Mathematics and Computer Science. Named the Computer and Automation Research Institute of the Hungarian Academy of Sciences (SZTAKI), this organization is determined to bring university students into contact with the most recent developments in

scientific research [2]. As part of the completion of these goals, SZTAKI has been broken down into eighteen departments each of which is classified as either a research or a development department.

One of the research divisions that exists within the SZTAKI infrastructure is the Parallel and Distributed Systems Laboratory. This lab does research in cluster and grid computing and has participated in almost every Hungarian Grid Project as well as several outside of Hungary. Currently, this lab's main focus is on the development of grid computing applications and the department has created three programs related to this task: PGRADE, the PGRADE Grid Portal, and the Mercury Grid Monitor. Our parallel application project deals directly with this department as its main focus is the extension of the PGRADE Grid Portal to allow for parameter study applications.

2.3.1 What is PGRADE

The PGRADE project is a high level graphical interface that allows users to develop parallel programs that can run on supercomputers, grids, and clusters without requiring the user to have any extensive programming knowledge. PGRADE is designed to be used by the non-professional programmer and can be used on almost every software platform. The system aims at making parallel program development more simplistic, enabling it to become a tool accessible to all [42].

2.3.2 How PGRADE Works

PGRADE is implemented in a manner that uses a graphical language, called GRAphical Process Net Language (GRAPNEL). GRAPNEL allows all inter-process communications to be defined graphically by the user allowing the underlying message-

passing system to be hidden. From the user's graphic representation, GRAPNEL generates all necessary PVM or MPI code. In addition to the GRAPNEL language, PGRADE uses the GRED editor tool to aid the user in creating the graphical portions of GRAPNEL. These GRAPNEL programs, edited with GRED, are then saved in an internal GRP file, which contains all the textual and graphical information that a GRAPNEL program needs [30].

Debugging with the PGRADE system requires the use of the DIWIDE debugger. This tool supports both graphical and C/C++ level debugging. It allows for the creation of breakpoints, step by step execution, and variable inspection [30].

PGRADE's implementation also allows the user to monitor the current running application. The system supports the use of two tools for monitoring, GRM and Mercury. GRM sets up a local monitor on each of the hosts on which the user's application runs. In addition, GRM sets up a monitor on the system that the user is working. This main monitor gathers trace information from each of the local monitors and stores them together.

For situations when GRM does not function properly, like when there is a firewall setup between the Grid sites where the local monitors are running and where the main monitor is running, Mercury can be used instead. Mercury uses sensors to gather measurements and performance data. These sensors are controlled by producers who can transfer the information to consumers when necessary. Communication between the producers and the consumers is handled through channels that can be created by either process. Channels started by the consumer are used mainly for interactive monitoring while those created by producers are used primarily for data archiving and event

reporting. Like GRM, Mercury sets up a local monitor on each node on which a part of the user application is running. These monitors use the sensors as a shared library. Also like the GRM implementation, the Mercury tool generates a main monitor on the main system the user is working on, which collects the information gathered by the local monitors [29].

The trace information created by either of these two systems is then used by PROVE, allowing the user to visualize and analyze the results. PROVE can be accessed both on and off-line. The user can control which hosts, jobs, processes and messages should be focused on. In addition, the user can access source code using a "click-back and click-forward facility" (W).

2.3.3 Uses of the PGRADE System

The PGRADE system can be used by mathematicians, researchers, and scientists with a limited amount of coding experience to perform resource intensive calculations. One project that has used the PGRADE system is the Hungarian Meteorological Service's Nowcasting algorithm which analyzes and predicts ultra-short range weather changes. These weather situations have the potential to be dangerous as they can destroy property and have a high risk of death. This project used PGRADE for its most costly computations, as the system could be easily parallelized within PGRADE's graphical environment. By using parallel code, the system could create, calculate, and re-calculate the current weather conditions allowing warnings to be given in a reasonable amount of time when necessary [35].

The University of Westminster located in the United Kingdom also used the PGRADE system. The cluster located at the school consisted of thirty-two computers

with a single master node. In order to create the simulation, researchers with the University of Westminster implemented a two part simulation using the MadCity Simulator [Using Clusters for Traffic Simulations]. The MadCity technology includes two tools; the Graphical Visualiser (GRV) which helps to design a particular road network and the SIMulator (SIM) which in this case study was implemented on the PGRADE system. The SIM portion of this project was implemented as a parallel program. In further executions of this study, this application was further extended to simulate numerous cities' traffic systems as well as interconnected road networks [20].

PGRADE can also be used for chemistry applications such as Reaction-diffusion equations. Reaction diffusion equations appear from the variety of spatiotemporal patterns that arise during chemical reactions and chemical diffusions. The evolution of these patterns is described through a second order partial differential equation [36].

The Department of Physical Chemistry at the Eötvös University worked with SZTAKI to implement such a system. The parallelized version of their code was then deployed on a self-made Linux cluster consisting of twenty-nine interconnected dual-processing nodes and used to forecast air pollution. As in the case of the weather forecasting application, the parallelized code enabled the calculations to return the required data in a timely fashion [35].

2.4 The PGRADE Portal

The PGRADE Portal uses Portal technology to provide access to the PGRADE system in an easy to access method. Using high-level Web interfaces, the Portal allows

users to create, execute and monitor jobs workflows. This system allows users to access the PGRADE project without having to use grid protocols and commands [45].

2.4.1 How the PGRADE Portal Works

The PGRADE Portal exists in two parts, the client side interface and the server side interface. On the client side, HTML is generated by the server side dynamic html web pages. This HTML code includes the entire interface and Java GUI required for interaction with PGRADE, including, the visualization applet and the workflow editor. These client side interfaces have corresponding server components in the form of servlets [45].

On the server end of the PGRADE Portal, Portlets are written with Java Server Pages (JSP), providing code that can generate the HTML code needed for the user. There is also underlying C code that gets accessed through JNI. These libraries include files for Mercury, trace2png which includes the png images needed for the visualization applet, and grp2c scripts [45]. These connections are visually shown in Figure 2.2.

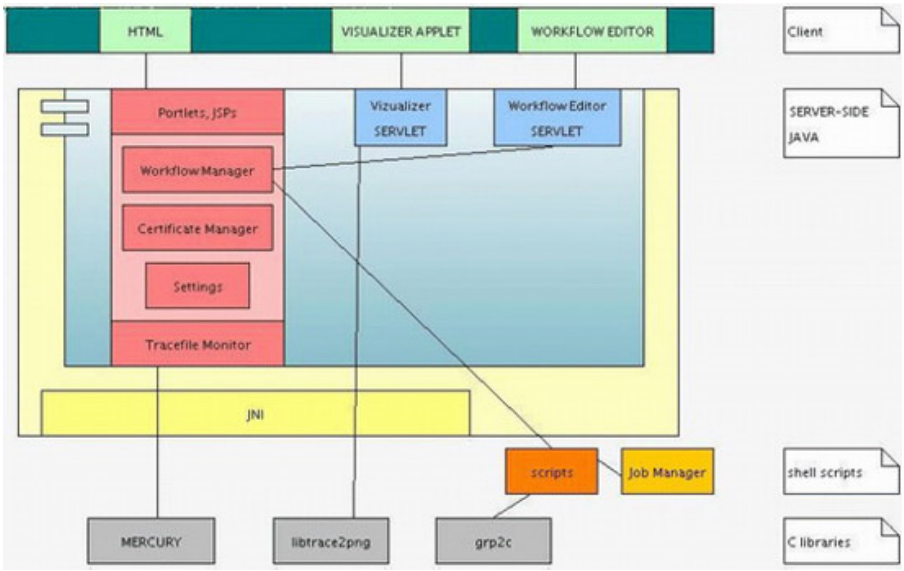


Figure 2.2 Diagram describing the intercommunications of the PGRADE Portal

When running the PGRADE Portal, in order to access the PGRADE system, a user must first obtain a certificate. In order to do this, the information in the window displayed in **Error! Reference source not found.** must first be filled out. For security purposes, these certificates have a limited lifetime and are downloaded from a MyProxy server. Once a certificate

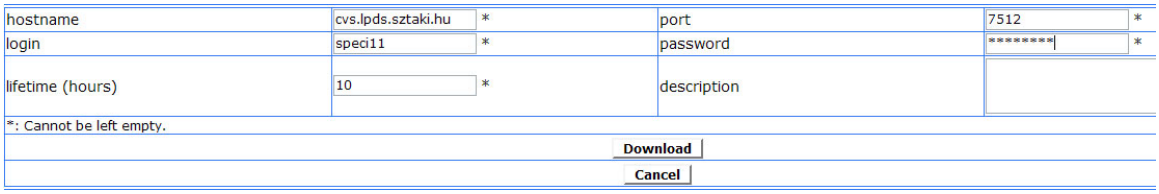


Figure 2.3 View of a Certificate Download

has been obtained, it can be set to be used with a particular grid. Once this has been done, a user can use the workflow tab to create a new job for the PGRADE Portal and execute it on the grid for which they have a certificate. This job can then be submitted and its process monitored through the Workflow list also accessible from the Workflow tab [SZTAKI PowerPoint Presentation].

2.5 Parameter Study

A parameter study application is a computer program that is capable of executing the same algorithm numerous times each time with a different set of input values. The values, or parameters, that are passed to the program vary along a range – also known as the parameter space. Once the program has completed, the application generates a set of results known as the parameter study. [51]

2.5.1 Uses of Parameter Study Programs

One function of a parameter study application is the execution of the same simulation numerous times. Each time the simulation is run a different input value, or set of input values, is passed to the program. Each execution, and its respective input parameters, represents a different set of tangible conditions that can occur. This allows a scientist to examine how something is going to react in the physical world in different circumstances without altering its current state [51].

Parameter study programs are valuable to the mathematical community. Statisticians use Monte Carlo applications to find statistical averages. According to the simplest definition, a Monte Carlo application is any application that involves random numbers. These problems have applications in economics, chemistry, and even nuclear physics [50]. The Monte Carlo algorithm needs to be run numerous times, each time with a random number seed in order to be accurate for statistical purposes. A random number seed is required in order to provide the algorithm with a sequence of pseudo-random bits. Computers, being deterministic, cannot generate truly random numbers, creating a need for a changing seed, such as the time on the system clock, to mimic randomness [4]. A Monte Carlo algorithm can become a parameter study problem if the random number seed is treated as the changing input value [51].

2.5.2 Implementation of Parameter Study Applications

The idea behind parameter study applications is not a new concept; however, the concept was not feasible until the emergence of Grid Computing. Prior to the existence of Grid Computing, parameter study applications had to be run locally on one's own machine. As it is in the nature of parameter study applications to be executed numerous times, running them locally created a strain on the computer. This forced the parameter

applications to be less complex then the needs of most scientific and mathematical problems.

The emergence of Grid Computing and computing advances in programs, such as Globus and Legion, has allowed parameter study application to be run remotely on grids [51]. Globus is a software tool used for designing Grids created by the Globus Alliance. This toolkit handles security, resource and data management, interprocess communication, error detection and portability [1]. Legion, like Globus, provides a user with an architecture that will allow them to develop Grid applications. Created as a project at the University of Virginia, Legion provides a system of hosts and objects that creates an illusion of a single computer while providing access to resources made available through their network [33]. Tools like these two enable users to write programs that are designed as parallel applications, allowing applications that could not be executed before due to their large computational needs to be executed and in a time efficient manner. It has now become practical for researchers to develop parametric applications.

2.5.3 Complications with Implementation

Although parametric applications are now a realistic consideration for conducting experiments, with the increase in speed and performance of computing, the complexity of development also increases. The more complex a parametric program is, the more likely it is that the application will require several layers of parameterization and the repetition of processing data and archiving it. It is also to be expected that these program characteristics will lead to the need for branching within the coding which only further

complicates the process as synchronization of the branches and processes will then also be necessary [10]. This process can be overall difficult and time consuming.

Furthermore, in order for parametric study programs to be a feasible solution to the scientific and mathematical communities, portions of the process must be hidden from the end user, while still allowing them to have some flexibility in the execution of their application. Each user should be able to define a logical method for how their process is run, be able to identify which parameters change and the behavior in which they change, and determine how their process is parallelized. The rest of the process, as it is beyond what the user should need to understand, should be hidden from them [10].

Also, as parameter applications become more complicated, the output data of these processes becomes larger. In order for this data to be useful to the person who is running the program, the data must be displayed and stored in a comprehensive manner. Without this key implementation step, the development of a parameter study system is futile as researchers using the program will be incapable of interpreting, their results and replicating their findings [10].

2.5.4 Other Parameter Study Applications

Other than the PSTUDY application developed as part of the SZTAKI PGRADE system discussed in this paper, there are several other similar applications that have been developed. Some of these programs (Nimrod and ILAB) will be discussed in the following section.

2.5.4.1 Nimrod

Nimrod is an older parameter study program and can thus only run simple parameter study applications. The program was developed so that it uses an internal "meta-language" to implement the creation of the parameter study. One of its strengths is that it can parameterize command line arguments with ease [51]. However, Nimrod can only be applied to single-level parametric models due to the simplicity of the declaration language that it uses. With the development of Nimrod/G and Nimrod/O some improvements were made as these two programs used the grid services provided by Globus [10].

2.5.4.2 ILAB

ILAB was developed by NASA to run on the NASA national infrastructure, the Information Power Grid (IPG). The goal of the NASA IPG project is to "provide ubiquitous and uniform access to a wide range of computational communication, data analysis, and storage resources" [51]. The ILAB project implements a parameter study program for NASA's IPG that includes a workflow manager [10]. In addition, ILAB supports four job modes – execution solely on the local machine without a scheduler, execution on a cluster without a scheduler, execution on a cluster with a scheduler, and execution on a cluster with the assumption that Globus middleware and a PBS scheduler are being used [10]. However, the ILAB software is limited to the complexity of Computer Aided Design (CAD) which it uses to display output and that does not support nested levels [51].

3 Methodology

Prior to implementing our Parameter Study Module, we planned our development process in order to efficiently spend this phase of our project. During this time, we determined what technological tools and in what languages our Module should be written. Furthermore, we determined how our interfaces should look and behave. Our decisions are explained in detail throughout our Methodology.

3.1 Technologies Involved in Implementation

In order to successfully design and integrate a Parameter Study interface for the SZTAKI PGRADE system, the interface developed had to have a similar look and feel to the existing system. In addition, there had to exist a seamless edge between both the old and the new interface. In order to fulfill these design tasks, technological choices had to be made regarding how to implement the client and server side interfaces. In the sections that follow, the tools that were chosen are described and reasons for choosing them are explained.

3.1.1 Java Server Pages

Java Server Pages (JSP) is a technology used for the creation of dynamic WebPages. An extension of the Java Servlet technology, the JSP package provides a method of imbedding Java code into HTML WebPages. In addition to supporting traditional HTML tags, JSP allows the programmer to define their own tags to be used within a JSP page. Furthermore, applications that are written in JSP are executable on nearly every computer platform [18].

3.1.1.1 How JSP Works

As mentioned above, JSP technology is an extension of the pre-existing Java Servlet technology. In fact, once accessed by a user, JSP files are complied into a Java Servlet class. This process involves the use of external software known as Tomcat. Tomcat is an open source program that implements Java Servlet and JSP technologies enabling them to be viewed on the internet [6]. The Tomcat server compiles the JSP code and generates the Java class by wrapping the HTML portions of the JSP file into Java output statements [6]. Then, when the JSP file is accessed from an Internet browser, the Tomcat server compares the modification timestamp on the JSP file and the timestamp on the compiled file and recompiles the JSP file if the modification timestamp is newer than that of the compiled file [6].

Tag	Meaning
<%-- … --%>	Comment
<%! … %>	Declaration of variables or Methods
<%@ .... %>	Include files, define attributes and tag libraries
<%= .... %>	Convert value of expression to string and write to output
<% ..... %>	Code

Table 1 JSP Tags

As mentioned above, JSP supports using HTML tags, user defined tags and Java code within the same file. In order for this process to work successfully, JSP defines special tags in order to distinguish between different forms of code. In addition to HTML supported tags, there are five other usable tags notwithstanding user defined tags and the inclusion of a tag library. These tags can be found in Table 1 shown above [6].

3.1.1.2 Alternatives to JSP

Overall, JSP is classified as a server side scripting technique. Included in this classification with it are technologies such as Active Server Pages (ASP) and PHP Hypertext Pre-Processor (PHP). ASP was designed for Microsoft Web servers. Due to this, ASP supports more languages than JSP, which only supports Java. These languages include JScript, VBScript, and C# in addition to other .NET languages. However, although the intended design of ASP allows ASP to support more languages that JSP, it also limits the platform compatibility to only Microsoft Web servers, making it a poor technology option for this project [9].

PHP is a third server side scripting technology. Scripting in PHP, unlike ASP and JSP, is done using a language written solely for use by PHP scripts. Although normally not an issue, the lack of support for Java by PHP makes PHP a poor option for this project as integration between the new Parameter Study Manager Portlet and the pre-existing PGRADE Portlet would be difficult [9]. For this reason, although PHP is supported on multiple platforms, it was not chosen as the method of implementation for this project.

3.1.2 Portlet Technology

As defined by Wikipedia, Portlets are reusable web-based components that provide information to Portal users [43]. Further defined by IBM during the development of the WebSphere Portal, a Portlet is defined as "visible active components users see within their Portal pages... In the simplest terms, a Portlet is a Java servlet that operates inside a Portal" [15]. Portlets are overall a new method of developing applications for the web. Due to their young age, there is no standardization as of yet, although steps have

been made towards standardization with the release of Java Specification Request (JSR) 168.

Although Portlets in their simplest form are Servlets that use JSP to create their pages, there are improvements in the Portlet technology that do not exist in Sevelet technology. Servlets, when serving requests, use two functions a “doGet” and a “doPost” which work in the same manner as the GET and POST requests in HTML. Portlets however, implement these methods through the Portal server rather than through the Web browser. Portlets also have a better logging system then Servlets and features that are not available in Servlet development, such as built in support to change the JSP page by device and the ability to treat user interactions as action events similar to the model used in Java programming [15]. Portlets were chosen as the implementation technique of choice for the web based server side interface due to ease of using Portlets with JSP and because the existing PGRADE Portal uses this technology.

3.1.3 GridSphere

To further extend the capabilities of the project, the Portlet technology will be implemented in conjunction with the GridSphere Portal [19]. GridSphere is a project whose objective it is to develop a standards based Portlet framework that will provide Portlet developers with a complete Grid Portal development application. The GridSphere Portal utilizes the Grid Application Toolkit (GAT) developed as part of the GridLab Project. The GridLab Project, of which GridSphere is a member, aims to create a method through which Portlet application developers to design more powerful Portlets [19].

The GridSphere Portlet is the encapsulating Portlet for all of the existing PGRADE Portlets. Through the use of GridSphere, Portlets that exist as part of the

GridSphere project do not need to be re-implemented by PGRADE developers. One use of the GridSphere technology within the existing PGRADE Portal is the execution of the "Administration" tab which handles the management of user accounts. As the existing PGRADE Portal system utilizes the GridSphere project, our Parameter Study Project does as well.

3.1.4 Java Swing

3.1.4.1 History of Swing

Swing was a project started in 1996 by Sun Microsystems to build up their existing GUI toolkit, the Abstract Window Toolkit (AWT). AWT was created around the concept of widgets, or small components, within a GUI that the user interacts with. The main problem with the AWT implementation was that it relied upon heavyweight components. Heavyweight components are widgets that rely on native peers within the local OS for their look-and-feel. This means that a GUI created using AWT would look different on almost every different operating system it was run on. This caused problems for developers since they had little control over the look of their components. Often this would lead to extensive cross-platform code fixes, which seriously hampering development time. The final problem with AWT was that these cross-platform incompatibilities made a common application GUI almost impossible. This directly contradicts what the Java programming language was developed for, a cross-platform language.

The new Swing GUI design implementation fixed these problems by changing to more lightweight components. These components were developed using two new design

models -- the Delegation Event Model and the Pluggable Look-and-Feel Model. These new design models were a variant of the Model-View-Controller Model. The Model-View-Controller design model operates around three design pieces. The Model is the element that manages the state of the component, the View is the visual representation of that component, and the Controller decides how the user can interact with the component. The Pluggable Look-and-Feel model is "Pluggable" because it allows the application to change its entire look-and-feel during runtime. It does so by allowing the entire component tree to define its look-and-feel dynamically through widgets mimicking design schemes. The final design model necessary, the Delegation Event Model, was actually developed prior to the development of Swing. It focused on moving away from the current chain based model of events where only components could handle events and an event was passed down the chain of components until it was either consumed or reached the root. The new model created three new elements, events, sources, and listeners. Events are the actual user interactions being performed, sources are the GUI components firing the events, and listeners are attached to components to handle the events specified to them. Using these new design models Swing was able to move away from many of the problems inherent in the AWT components. The final compatibility issues were resolved by using only pure Java code to create the components, thus pulling them completely away from reliance on native components and code. [7]

3.1.4.2 The Use of Swing Within the Portal

The current GUI system within the P-GRADE Portal uses Java Swing components for all workflow, job, port, and miscellaneous editing windows. This allows the Portal to hold cross-platform compatibility within the look-and-feel of all of its GUIs. For this

project many of these GUIs were extended to implement the additional GUI components required by a parametric study. To maintain this level of cross-platform compatibility, only Java Swing components were used in the creation of the new extended GUIs.

3.2 Editing Section

The editing section's specific tasks were to extend the current workflow editing structure and GUI to implement the new addition of parametric study addition. This was to be completed by extending the workflow editor, port definition, and job property dialogs and also to create a new parameter set generation dialog for defining parametric key values.

The dialog extensions called for:

1. A new parametric input port to be added to the job icon within the workflow editor window

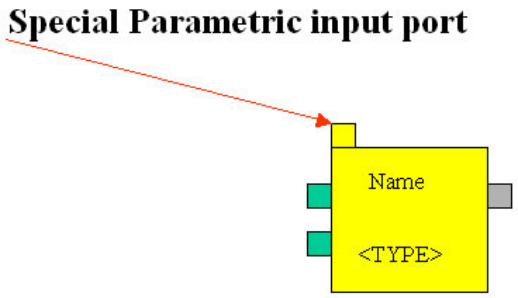


Figure 3.1 Suggested Parametric Input Port Representation in the Workflow Editor Window

2. A GUI extension within the port property dialog to allow for parametric input keys to be defined

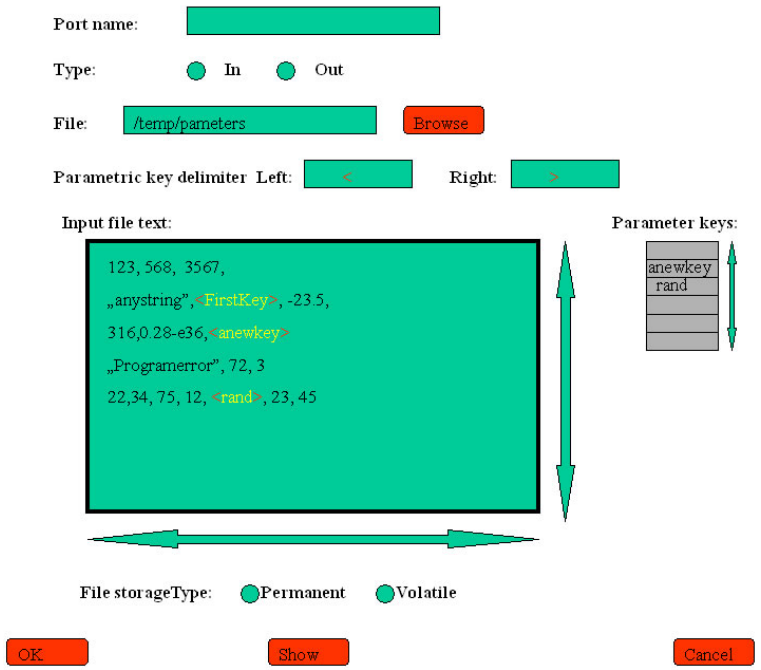


Figure 3.2 Suggested Parametric Extension of the Port Property Dialog

3. A GUI extension within the job property dialog to allow for the number of key values used within job execution and their order within the Cartesian product to be defined

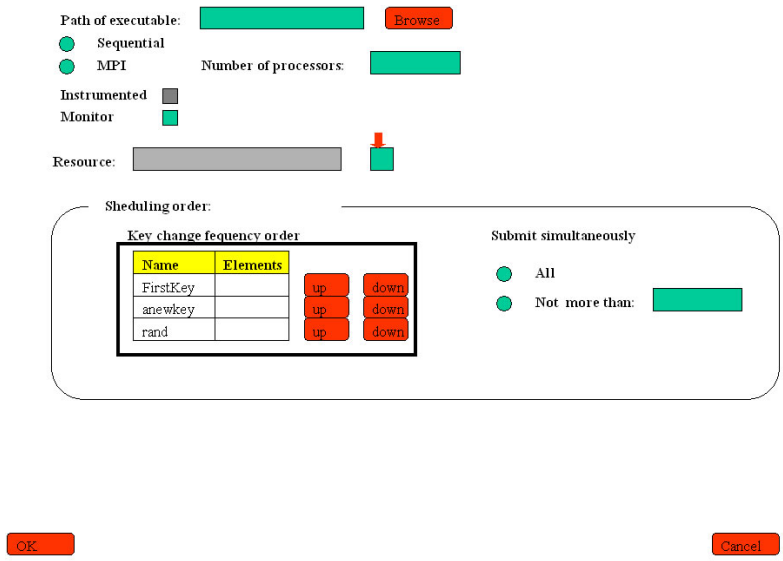


Figure 3.3 Suggested Parametric Extension of the Job Property Dialog

Figure 3.4 Suggested Parametric Set Generation Dialog

The parameter set generation dialog is a new dialog opened from within the port property dialog to define the values for a parametric key. This new dialog allows for the user to

define the key type (INT, REAL, or CHAR), format (free or bound), and the values for the key. The values for the key can be generated in four different ways. In the first method they can be generated through the user entering a set expression. A second way allows the values can be generated by the user specifying a local file to use that contains the set expression. Both of these methods call for a special delimiter character to be specified by the user. A delimiter is a special character used to separate two pieces of data. The third method uses a simple for-loop structure to define the range of values by allowing the user to enter the from, to, and iteration values. The fourth and final method for key value generation uses an internal random number generator to which the user supplies a seed number to. This method also requires the input of the number of cases to compute, and from, to, and iteration values which are used for mapping the range in a linear way.

3.3 The Parameter Study Manager

To create a seamless edge between the currently existing PGRADE system and the additional features created through this parameter study project, we decided that the Parameter Study Manager should exist as a new tabbed pane in the old PGRADE Portal design. This tabbed pane was designed to look similar to the existing Workflow pane allowing old users to feel comfortable with the new features almost immediately while lowering the learning curve for people who are new to the system.

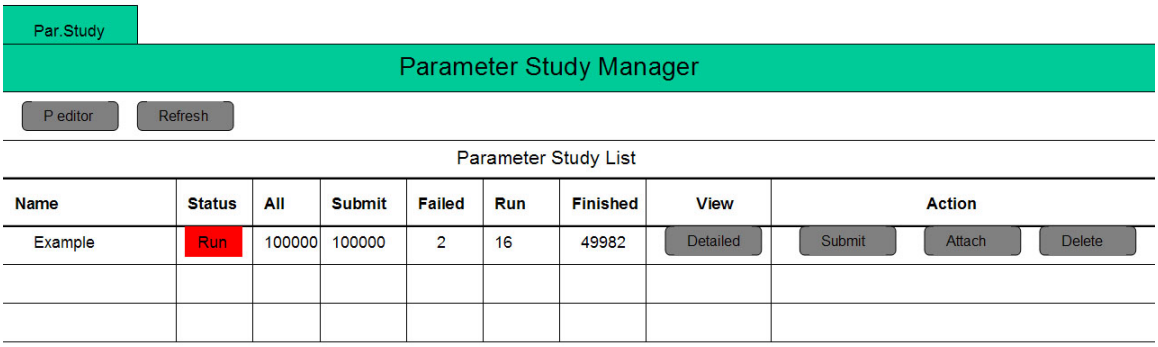


Figure 3.5 Proposed Parameter Study Manager Design

Like the existing Workflow tab, the planned Parameter Study tab was designed with a table that will contain a list of the current user's existing Parameter Study jobs (see the above Figure 3.5). As a user submits, attaches, and deletes jobs, this table refreshes in order to insure accuracy. Furthermore, in addition to listing the names of each of the user's jobs, the table includes other basic information about the job such as its status, the full set of the Cartesian product for that job, the number of instances of that job that have been submitted, the number of instances that have failed, and the number of instances that have finished. In addition to this basic information about a job, more information about the Parameter Study is accessible through the "Detailed" button which brings up a second view of the Parameter Study Manager.

The Detailed View of the Parameter Study Manager pane includes information regarding each of the parameters whose values can be changed from one instance of a job to another. In addition to the name of the parameter, information regarding the overall

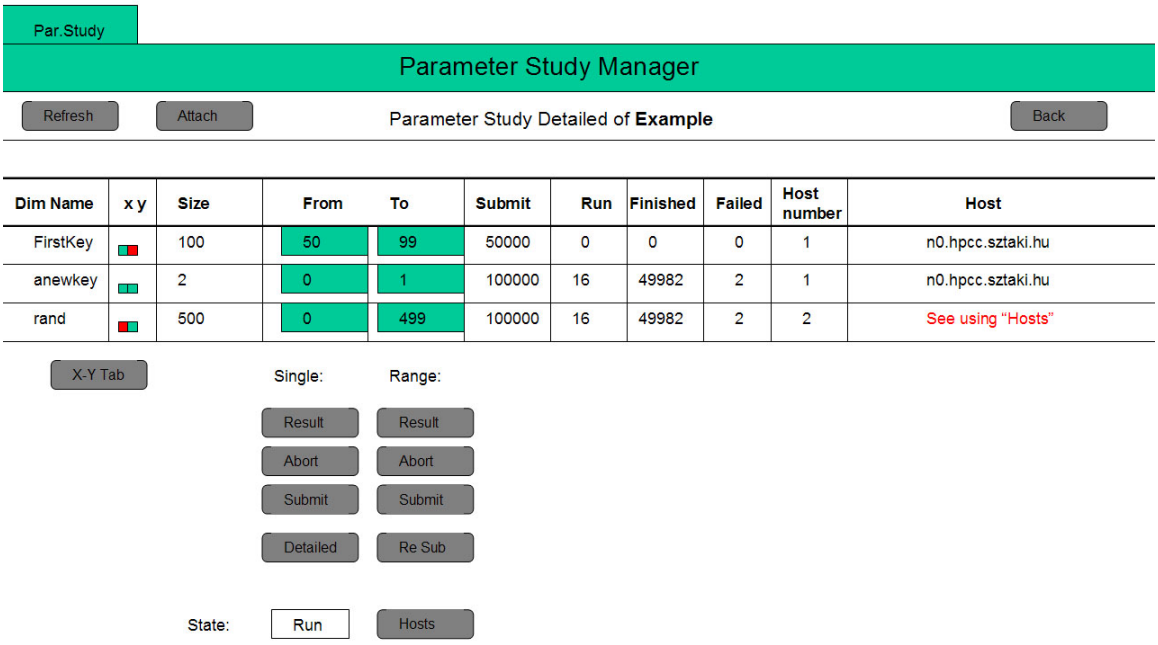


Figure 3.6 Proposed Parameter Study Manager Detailed View

range of values; the range of values currently being chosen from for each instance, statistics regarding the number of successful, failed, and currently running tasks, and the hosts on which the computations that use the parameters are running are displayed. Through this window, the user can also submit (or attach) their job, view the graphical results of their job, and abort the job. In addition, if the job is a ranged job, the user can re-submit the entire job and fill in information regarding the hosts. A visual representation of this design is shown above in Figure 3.6.

In order to implement the aforementioned GUIs in a manner that reflects the current design while allowing for the proposed design to be successfully completed, Portlets were chosen as the implementation technique. This technique, already used in

the PGRADE system, allowed the new GUIs to look as much like the old ones as possible especially when it is taken into account that a Portlet implementation supports the usage of the existing PGRADE Portal cascading style sheet (CSS). In order to create these Portlets, JSPs were written and called through the Parameter Study Manager's Portlet class. In addition, a bean class was created for the Parameter Study Portlet. This class stores useful functions necessary for the implementation of the Parameter Study Portlet.

3.3.1 Development of the Parameter Study Manager

In order to create the server side interface that is the Parameter Study Manager, the first task that had to be completed was the creation of the tab itself. This required editing three preexisting Extensible Markup Language (XML) files so that these documents referenced a new Portlet which, in following the naming standards established by the SZTAKI development team, are called the PStudyPortlet. This Portlet was designed to mimic the current PGradePortlet. This was because the role the Parameter Study Monitoring tab fulfills relates to the Parameter Study Module in the same manner that the role of the Workflow tab corresponds to the PGRADE Portal.

Once the Portlet itself was created, a JSP file was written that displays the initial window. This window, designed to look like Figure 3.5, was temporarily written with the entries for the Parameter Study table hardcoded. This changed during the integration phase of our project when our interfaces were joined with the applications developed by the Development of Algorithms on the Grid MQP team with whom we have worked in conjunction. Once integration was completed this table became a dynamic listing of the current parametric jobs that a given user has submitted for execution.

In addition to the main JSP file, other secondary JSP files were created to handle events that result from user interactions with the aforementioned main interface. One of these JSP files handles the detailed view of the Parameter Study Manager. This interface, whose importance ranks near that of the main interface, will be designed to look like Figure 3.6. JSP files were also created to create a user friendly system. These files include, but are not limited to, a help page, and confirmation of submission, abortion, and deletion.

A bean class was also created to be used by the PStudyPortlet class. This class stores standard functions that are necessary for the Parameter Study class. These functions are be implemented as a bean so that future Portlets that wish to interact with the Parameter Study jobs can do so without rewriting these basic functions. Furthermore, this bean class extends the existing PGradeBean so that functionality that the two beans share, such as storage of the username, did not have to be duplicated. This class, continuing with the naming style of the PGRADE Portal, was called the PStudyBean.

4 Implementation

Once a design for the project was completed, implementation of these tasks was begun. The implementation, as planned in the planning portion of our project, was completed in two phases – the development of the extension of the Workflow Editor of the PGRADE system to allow for parameter study and the creation of a graphical Parameter Study Manager. The process of completing these two phases of the Parameter Study Module is discussed in detail below.

4.1 Implementation of the Workflow Editor Section

4.1.1 Parameter Set Generation Dialog

Development of these extensions began first with the parameter set generation dialog in an offline form as this allowed for a quicker implementation with integration following at a later date. The parameter set generation dialog seemed a logical choice for the first piece to implement as it was the only completely new dialog to be created and thus would take the greatest amount of development time.

It was initially thought that this dialog was going to be created manually, but due to design time constraints the Visual Editor plugin extension for the Eclipse development tool was chosen to speed up development. The Visual Editor plugin allowed for visual development of the GUI, which greatly sped up design and testing phases. As with any visual editor though, this added a good amount of additional cleanup time as the code needed to be reworked and optimized for better performance and readability. [12]

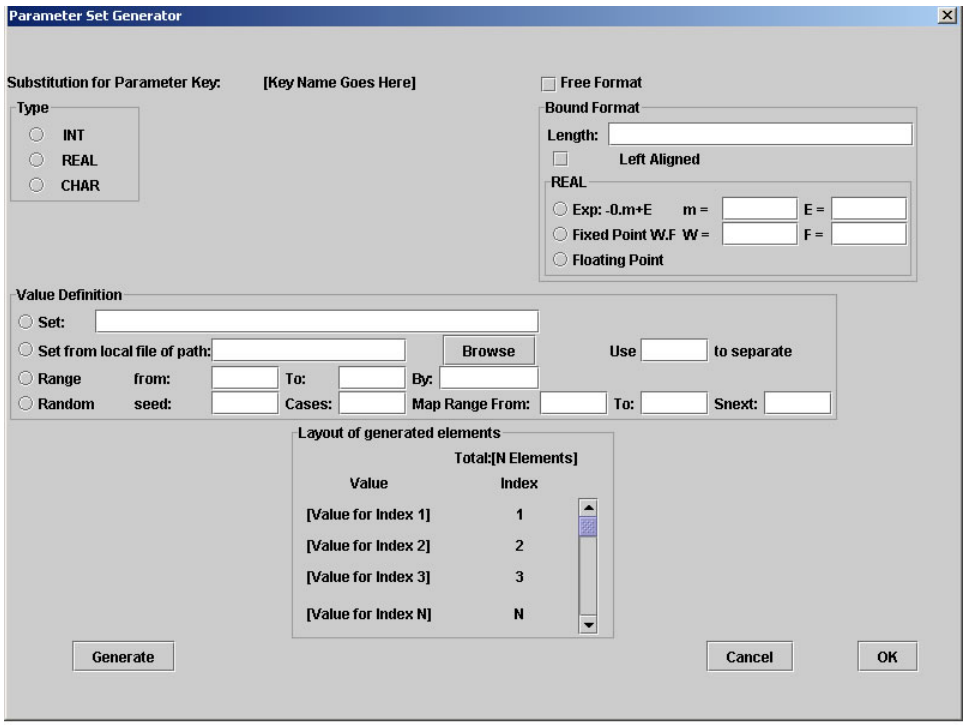


Figure 4.1 Implemented Parameter Set Generation Dialog

One major design issue encountered within this piece of development was the data structure or object to be used for the keys and their generated values. The initial suggestion was to create a linked list containing a designation as either a "P – Plain Text Node" or "K – Parametric Key Node". The plain text nodes would then link to a string object while the parametric key nodes would link to an array containing the keys values. The final implementation differed from this suggestion. The parametric key list and parametric keys were created as classes with the parametric key list object extending Java's LinkedList class. This allowed for close functionality with the initial suggested implementation while also allowing greater modularity. After generation of this new object to contain the specified structures the integration of this object into the middleware piece of the parametric study addition had to be addressed.

Since the parameter set generation dialog lives entirely within the client side of the application, the corresponding objects and data files must be uploaded to the server before they can be accessed and used by the middleware layer. As such parsing methods to pull the necessary data from the new key object for uploading were necessary. Three such methods were implemented – one to pull out all of the keys, a second to pull out their generated values, and a third to parse the two together into a delimited text string for uploading. The third one was the main parsing method necessary for integration but the other two were helpful to local access methods within the other dialogs. The delimited text string representation was necessary as all transfers were done through an HTTP transfer layer over TCP/IP.

The second major design problem encountered was exactly where to place the parametric key class objects within the object hierarchy. The first implementation had the objects entirely within the port property and parameter set generation dialogs but this did not allow visibility to the workflow editor itself. To fix this problem the parametric key class objects were moved up into the new parametric input port class to allow for visibility within all of the necessary dialogs.

4.1.2 Remaining Dialog Extensions

After the set generation dialog was created the ability to access the dialog from the workflow editor and the other dialogs was the next logical implementation step. The first piece to implement was to add the new parametric input port icon to the job icon within the workflow editor. This was accomplished by first creating a new parametric input port class extending the current input port class. The new class contains the

parametric input file, the parametricKey class, and a linked list representation of the input file text and the parametric keys and their values.

This new parametric input port object is defined within the new extended parametric port definition dialog. The new dialog allows for defining the parametric input file, left and right delimiters for input file text parsing, direct editing of the input file text, and allows for accessing and defining the values for each parametric key by opening the parametric set generation dialog upon the user double clicking a parametric key within the GUI.

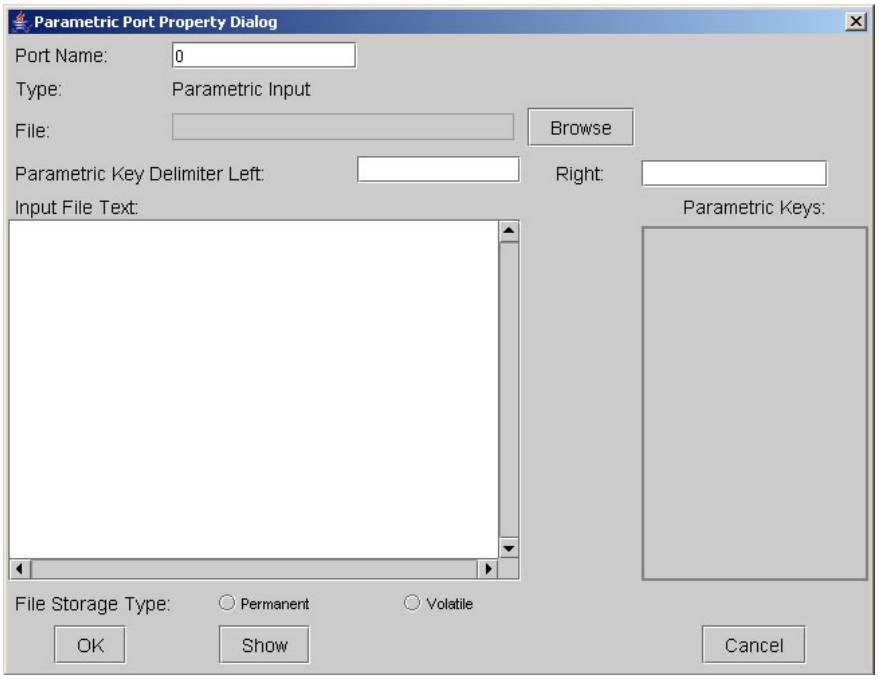


Figure 4.2 Implemented Parametric Port Property Dialog

A new parametric job class was now necessary to implement the connection between the workflow jobs and their new parametric input port.

The new parametric job class extends the current job class allowing for greater modularity within the code. The main extensions within this new class are the ability to

set the order of the parametric keys within the Cartesian product for execution scheduling, the ability to submit all or a specified number of parametric key/value combinations, and the connection to the new parametric input port. The parametric job object and the new extended functionalities mentioned above are accessed through a new parametric job property dialog.

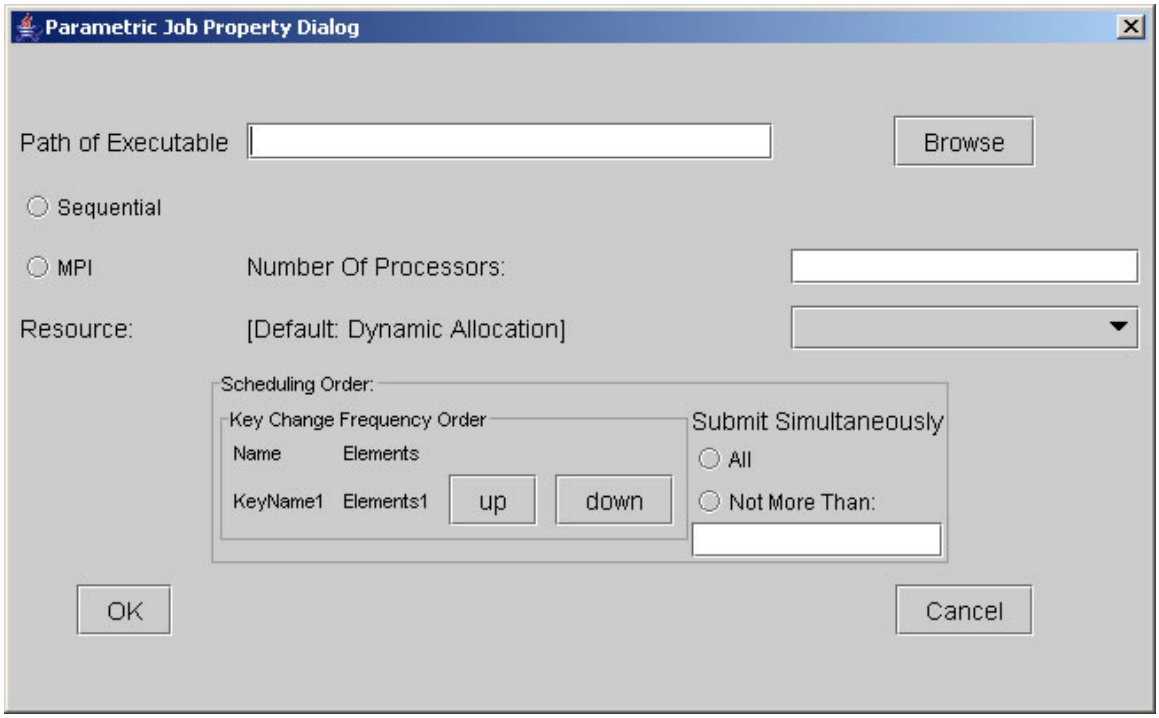


Figure 4.3 Implemented Parametric Job Property Dialog

4.1.3 Integration of the Editing Section

With the connection hierarchy and new parametric classes and dialogs now created, integration into the P-GRADE Portal and the other project sections began. This started with the development of communication standards between the new editing section windows and the middleware layer created by the Development of Algorithms on the GRID MQP team.

The main communication link is established during saving/uploading of the current workflow in the parametric workflow editor. A new workflow file format was established to define the new parameters necessary for later execution of the parametric workflow. This new workflow file was structured as shown on the following page-

workflow "WorkflowName"
{
}
{

"JobName" JOB_TYPE (is_instrumented=true/false;monitor=true/false) "[Location of the executable]"
{
""
}
"LINUX"
{
"&&##!!HOSTNAME!!##&&"
}
PortNumber "[Port File Location]" PERMAMENT/VOLATILE INPUT/OUTPUT
}

{
"JobName" X Y
}

The next communication standard necessary was for the content of the parametric input file, and the parametric key list file. The final representation was that the parametric key list file would be structured as- "left_delimiter;right_delimiter;key_name1,value1,value2,…;key_name2,value1,value2,…". The

included left and right delimiters were used to then parse out the key names from within the parametric input file. This allowed for all transmissions to be done cleanly and purely via text strings.

4.2 Implementation of the Parameter Study Manager

As mentioned earlier, the Parameter Study Manager was implemented as a collection of Java files and JSP files. Integrated together, these files formed a visual method of interaction with the PStudy Module for the user. Once these files were created, they were integrated with the code developed by the Development of Algorithms for the GRID MQP team to create a fully functionally parameter study application.

4.2.1 Creating a Communication Method Between the JSPs and the Portlet

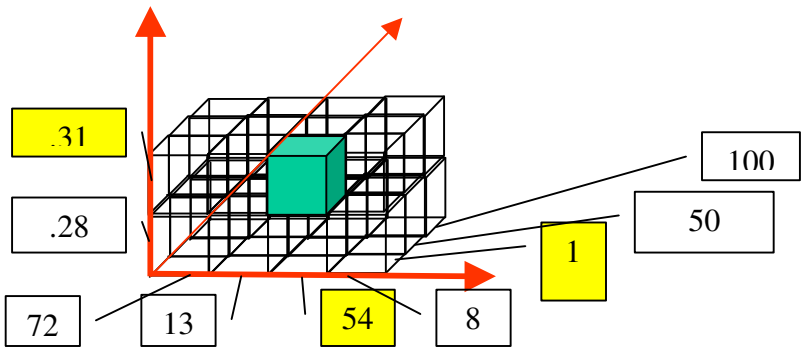
When developing with JSPs, there is no method to pass variables directly from one JSP to another. In order to maintain a concurrent version of the Parameter Study data, the active JSP must send information back to the Portlet which invoked it and, once the data has reached the Portlet, the Portlet must save this information in a manner that allows it to pass the data along to future JSPs. For example, if a user submits a Parameter Study Job through the index JSP, in order for the other JSPs to register the job's state change, the index JSP must register the submit action with the Portlet. The Portlet, in turn, must save this state change and, when necessary, pass the updated data to other JSPs, such as the JSP that loads the detailed view.

To maintain an accurate data set, the implemented PStudyBean was used. By declaring a single bean variable globally in the PStudyPortlet, each Portlet function was

able to access the bean's variables such as the current job list id. This single bean entity was then updated as user actions were preformed. In addition to being accessible by each function in the PStudyPortlet class, this same been entity could then be made accessible by each of the JSPs by having the function that instantiated the next JSP call PortletRequest.setAttribute(). Although the JSP cannot update the bean once it has been invoked, it can fake this action by passing parameters back to the Portlet through hidden fields. The Portlet can then receive this data through the PortletRequest.getParameter() function and update the bean's data accordingly. Both of these two functions (getParameter and setParameter) implement HTML functions in their low level implementation.

4.2.2 Determining the Range of Indexes for Submission

In order to submit a parameter study job to the job manager, a range of indexes must be passed to the PStudyJobList class's submit function. This range of indexes represents the array of values of each key (or parameter) which should be executed. This range of indexes, however, does not contain the actual values, but a listing of the ids of the values (based upon their location in the internal array in which the values are stored) in every possible combination of the ids. For example, if a parameter study contained three keys;



length, width, and height; and these keys had values that could be .28 or .31; 72, 13, 54, or 8; and 1, 50 or 100 respectively, this range of indexes would be [{0,0,0}, {0,0,1}, {0,0,2}, {0,1,0}, {0,1,1}, {0,1,2}, {0,2,0}, {0,2,1}, {0,2,2}, {0,3,0}, {0,3,1}, {0,3,2}, {1,0,0}, {1,0,1}, {1,0,2}, {1,1,0}, {1,1,1}, {1,1,2}, {1,2,0}, {1,2,1}, {1,2,2}, {1,3,0}, {1,3,1}, {1,3,2}]. A graphically representation of this is shown in Figure 4.4. Although an easy list to produce for a specific circumstance, this range becomes difficult to innumerate when one realizes that for each parameter study not only can the number of keys vary, but the range of acceptable values for each of these keys can also change from study to study.

In order to develop an algorithm that would produce this list of numbers for all situations, we first analyzed the situation mathematically. According to combinatorics, if there are n objects there are n! ways in which these objects can be arranged. Applied to this problem, this statement becomes more complicated. Here we have n distinct sets of objects and one object from each set must be included in the final result. As the order of the sets within the final result is fixed in order to provide accuracy when submitting, the number of final permutations in based solely on the changing parameters. Applying the combinatoric theorem stated above, this indicates that if there are n parameters, and each parameter have a difference between there maximum and minimum values of m_i where i goes from 0 to n-1, the number of ways to arrange the indexes of these values is (m_i * m_{i+1} * m_{i+2} * … * m_{n-1}). Although this calculation is easy enough in the present circumstances, enumerating and storing each of these permutations in a manner that can be used to submit the parameter study is a little more computationally costly.

One necessary characteristic of the algorithm designed to solve this problem is that it must choose an index value for each key between some minimum index value and some maximum index value. In designing our algorithm, we choose to use a for-loop to execute this process. By using a for-loop, we ensured that every possible value within a given key's range of indexes is seen by the algorithm. However, it is not enough for our algorithm to know ever index within the range of each key. Our algorithm needs to pair each of these indexes with every other key's range of index values. In order to complete this task, each key's for-loop was imbedded within the for-loops of the other keys. For the example given earlier, the puesdo code would be as follows:

```
for i = 1 → i = 2
        for j = 2 → j = 4
                for k = 4 → k = 5
                        permutation = i, j, k
```

However, as the number of keys can vary from job to job, it cannot be known prior to execution how many for-loops to embed. To solve this problem, we chose to design our algorithm to be recursive. By writing our algorithm in this method, a for-loop can be executed for each key without limiting the number of keys a user can have. The code for our final algorithm can be found in Appendix A.

4.2.2.1 Evaluation of the Range of Indexes Algorithm

The first computation the algorithm requires is the calculation of the maximum number of permutations that can be formed by the range of indexes for each key. In order to compute this number for an arbitrary number of keys, this product is formed through the implementation of a for-loop. This for-loop performs one iteration for every key therefore it runs in n time where n is the number of keys.

The second computation preformed by the algorithm embeds a for-loop for each key in the parameter study. Each for-loop runs from the minimum index to the maximum index for its respective key. If each of these for-loops has a range m_i where i corresponds to the key index, beginning at 0 and incrementing by one up to n, the innermost for-loop will execute for (m_i * m_{i+1} * m_{i+2} *… m_{n-1} * m_n) iterations. This product, called the Cartesian product, will grow depending on the number of keys and the range for each key. In test runs of this algorithm, for small numbers the execution took a unnoticeable number of milliseconds. However, for larger numbers, such as 20,000 the lag in execution was noticeable. For the purposes of this Parameter Study Module, however, numbers this large are not factors in execution time as users are limited to a Cartesian product of no more than 10,000 which still executes in a reasonable time.

4.2.3 Creation of Reusable Code

As the Parameter Study Module of the PGRADE Portal will continue to be developed after the completion of this project due to new developments in this research area, it was determined that the internal functions of the PStudyPortlet and the JSP files should be as reusable as possible. This implementation choice will allow future developers to make changes to the code without having to change the same lines numerous times. However, in order to implement the submission, abortion, and deletion of both a ranged and single Parameter Study job from the detailed view, a global data type had to be declared that would maintain the user inputted range value for each key. This variable allowed the user confirmation to be handled by one function for ranged operations and a second for the single set operations but the actually task, either abortion, deletion, or submission, to be handled by a single function.

Three variables were declared in the Java bean to handle this design issue. Two of these were int[]. One of them held the minimum values selected and the other the maximum. The third variable held the String[] that contained the entire listing of combinations required by the back end for submission, deletion, and abortion. Each time the detailed page is refreshed, or a button is pressed on the detailed page, these variables are updated.

4.2.4 Displaying the Current State of the Parameter Study

One important aspect of the Parameter Study Manager is its ability to display the current execution state of each parameter study to the user. In order to complete this task, a key of states was developed. Each state – running, submitted, initialized, error and aborted – were assigned their own font color and background color. The job state re-submitted was considered the same as submitted for these purposes. These colors were each stored in an array in the index corresponding to the number version of their state. This allowed the user to easy recognize the status of their study with limited confusion. The keys chosen are shown below in Figure 4.5.

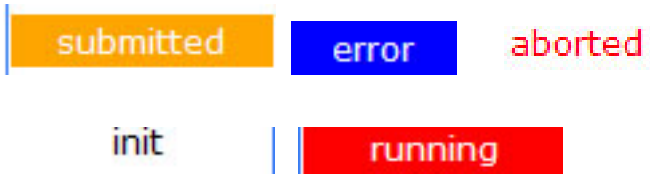


Figure 4.5 Keys for Indicating the Status of a Parmater Study

4.2.5 Removal of Designed Features

During the course of integration, it was determined that two features originally included in the design of the Parameter Study Manager should not be included in the final implementation. One of these features was the "Hosts" button. This button was originally designed to allow a user manually decide on which host each instance of the parameter study should be executed. However, as a parameter can have numerous instances, this task could become tedious. In addition, if the user were to make a poor decision, their actions would make the execution less than optimal. Instead, the back end was implemented in a manner that dynamically distributed each instance to a host. The load balancing algorithm used decides on each host not at the beginning of executing, but as jobs finish and resources free up. This insures the host being assigned never has finished with all previous instances before another one is added to its queue. However, although this button has been removed from the tab, information regarding the hosts can be added to the detailed view's table allowing the user to know which host is running a particular key.

A second button which has been removed is the "X-Y tab" button. Rather than developing this feature as another portion of the Parameter Study Portal, it was programmed as a separate Portal accessible through its own tabbed pane. This allowed this feature to be implemented not only for parameter studies, but regular workflows as well. The development of this portion of the module was completed by the Development of Algorithms for the GRID team and incorporated with our project during the integration phase.

4.2.6 Integration

Once the Parameter Study Manager was completed, it had to be made functional through implementation with the Portal Back End. In order for this process to be successful, prior to the beginning of implementation, it was determined what functionality the back end had to support. These functions included submission of both a range of index sets and a single index set Parameter Study, abortion of both a range of index sets and a single index set Parameter Study, deletion of an entire Parameter Study Job, and resubmission of a Parameter Study given a range of indexes. Stubs for these functions were created allowing for the Parameter Study Portal to include these functions before they were fully implemented. Once these functions for these processes were written these functions were able to be executed without changes being made to the Parameter Study Manager.

In addition to integrating with the back end, the Parameter Study Manager had to integrate with the Workflow editor. The "Attach" buttons and the "Parameter Study Editor" button both call the Workflow editor allowing a user to create a workflow for a parameter study job. However, unlike integration with the back end, standards did not have to be determined beforehand as the Workflow Editor operates as an applet, which a JSP can easily initiated.

5 Testing

Throughout the course of our development phase, we locally tested our project ensuring that each section worked before implementing the next state. Once this had been done, integration with the Development of Algorithms for the GRID team was done. Following our integration we tested the entire project as a working module. Our methods of completing these stages of testing are further explained in the sections that follow.

5.1 External Testing

One method of testing used in development of this project was external testing. When executing this form of testing, code was removed from the Java project and copied into a separate Java application. In the development of the Parameter Study Manager this allowed Java code to be tested without involving communication between the Portlet and the JSP. Through this manner we were able to determine whether a segment of code was failing due to communication issues between the JSP and the Portlet or due to logical errors in the Java code. This is necessary as JSP programs upon compiling and executing does not often meaningful error messages; most of the time either the webpage does not display or an "Attempt to invoke invalid Portlet action" error message is displayed. One area in which this method of testing was used was during the development of the Evaluation of the Range of Indexes Algorithm. The test files for this code segment can be found in Appendix B.

External testing was also used heavily for the editing section of the parametric study Portal extension. This allowed testing of the look-and-feel of the GUIs as well as inter-functionality prior to integrating them into the full P-GRADE Portal system. Many

problems located too late after trying to compile it into the system using ANT could be quickly spotted this way and corrected prior to taking the time necessary for transfer to the Portal testing environment.

5.2 Creation of a Local PStudyJobList

A second method of testing involved the creation of a local version of a PStudyJobList instance. This involved creating dummy PStudyJobs, adding them to the fake PStudyJobList, and setting the respective variable in the PStudyBean to this PStudyJobList. By implementing these temporary local test variables, basic functionality of the Parameter Study Manager could be tested without waiting for other portions of the project to finish.

In order to be useful testing objects, the dummy jobs were created with a various number of keys and with each key having a different number of values. Furthermore, grids were set for each of the jobs so that certificate validation could be checked. These features allowed the dynamic formation of the tables on both the Parameter Study Manager window and the Detailed View to be checked. In addition, it allowed the verification JSP and the window redirections to be validated prior to the calling of the back end's functions. This limited the number of locations errors could exist when integration took place.

These testing variables also made it possible to work without having an active Workflow Editor. Had these variables not existed, the Parameter Study Manager's functionality would not have been able to be tested until the very end of the project at which point it would have been too late to fix any bugs found.

This phase of testing was the most useful as it allowed errors in the JSP code to be found. In addition, it enabled progress to be made without having a functional parameter study application. Furthermore, it allowed us to test every combination of buttons that a user could and would press in order to determine if the right event happened at the right moment.

5.3 Creation of a Local ParametricWorkflow Object

Due to difficulties integrating the new extended parametric study GUIs into the P-GRADE Portal a local testing system was developed using dummy ParametricWorkflow, ParametricJob, and ParametricPort objects. This allowed for dry run tests through the extended GUIs using sample data to test their functionality.

Sample workflows were created to mimic possible scenarios encountered by the GUIs in actual external use. These workflows consisted of many different possible combinations of jobs, ports, parametric keys, and value combinations allowing for extensive testing of the system. One example configuration was one created to mimic the Mandelbrot program created by the Development of Algorithms for the GRID MQP team. Using this system we were able to ensure that the existing foundation for the parametric study GUI worked correctly and could easily be expanded and integrated further in the future.

5.4 Monitoring of Trace Files

Testing was also done using trace files. Prior to our implementation, The PGRADE Portal allowed output statements to be written to a file name Catalina.out.

Through this file, details about how the system was operating could be seen. During the development and integration phases of our project, we took advantage of this file. By adding output lines to our program, we could monitor the execution steps of the system. The allowed us to determine how data was being received from functions enabling us to limit where errors were occurring. This was especially useful when ensuring the Parameter Study bean was correctly passing information to and from the JSPs.

5.5 Integration Testing with the Mandelbrot Set

Once our project was integrated with the Development of Algorithms for the GRID team, the entire system had to be tested to ensure each portion of the module was operating correctly when combined with the other sections. During the course of the development of their project, the Development of Algorithms for the GRID team developed a parameter study that solved a Mandelbrot equation and produced graphically output for the program. The Mandelbrot Set is created through the following function:

$$Z_{n+1} = Z_n^2 + c \text{ where } Z_0 = 0$$

The c represents any complex number. This function is recursively evaluated for the desired number of iterations. Based on whether for a particular value the function converges or goes to infinity, a graph can be colored where each color represents a different rate of convergence [49]. An example output for a particular Mandelbrot set is shown in Figure 5.1

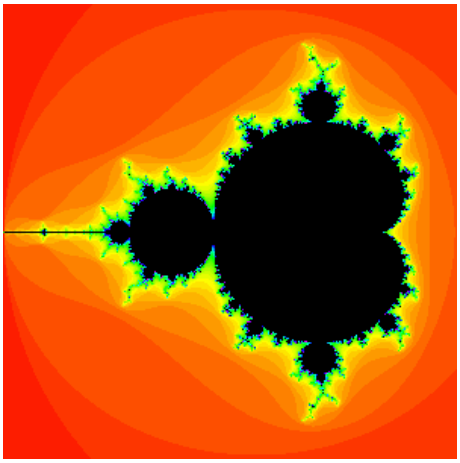


Figure 5.1 Output from a Mandelbrot Set

Through the Mandelbrot program, all aspects of the module could be tested. Prior to this form of testing, only local hardcoded PStudyJobLists and PStudyJobs were run through the program. This did not some features to be tested, such as the refresh operation and whether data changed in the index view was updated on the detailed view (and vice versa). In addition, prior to this, it was impossible to test whether the aforementioned state keys changed correctly for all possible states. Furthermore, full integration testing could only be completed through the execution of an actual application. Overall, this stage of testing was a complete success.

6 Conclusions

After designing, implementing, integrating and testing our project, we were able to implement the majority of a Parameter Study Module for the PGRADE Portal. Paramter Study applications are just beginning to be developed. Due to this, there are additional features that can be added to our project, but that due to time constraints, are not currently implemented. Our recommendations for these features are described below. Furthermore, in addition to recommending future additions to our module, we have made recommendations to the Sztaki development team for future improvements to the overall system.

6.1 The Final Parameter Study Manager

After the completion of the implementation and testing stages of our Parameter Study Manager development phase, our project resulted in a complete Parameter Study Manager interface. This design, although slightly different than the original plan, is a working model of each of the features. Submission, abortion, deletion, and re-submission of jobs all seem to successfully operate. In addition, the interface is capable of displaying messages to the user and displaying confirmation windows for most user actions making it a very user friendly design. Furthermore, information regarding each state is displayed to the user accurately and updated upon refresh allowing the user to always have an up to date status report on their job. Overall, the implementation of the Manager was a success. Final screen shots of this GUI can be found in Appendix C.

6.2 The Final Parameter Study Editing Section

Following the final testing and integration phase the editing section GUIs and supporting classes created a solid foundation for future development in this area. A complete Parameter Set Generation Dialog was developed with all main functionality intact. Also the necessary underlying ParametricKey, ParametricKeyList, and supporting classes were developed and supported all necessary functionality. Foundation classes and GUIs were created for the remaining dialogs and GUI extensions and implemented most of the functionality required.

6.3 Future Implementation

As parallel parameter study applications are a relatively new area of development, there are several functionalities that can be added to our current Parameter Study Module. On the Parameter Study Manager's detailed view, there is an unimplemented buttons that exist in the original design -- the detailed view for a single set of indexes. Also, future work within the Portal GUI is necessary to integrate the Parametric Study Editor.

6.3.1 The Detailed View of a Single Index Set

The "Detailed" button that is linked to from the Parameter Study Detailed View would work for a single set of indexes. A user would select values from the "From" column and specific information regarding that particular instance of the Parameter Study job would be displayed. Information that could be viewed from this page would be the particular host that instance is running on, the state of that instance, and the output file for that instance. It is recommended for consistency reasons that this section be displayed in the form of a table.

6.3.2 Full Integration of the Editing Section

The editing section, while providing a solid foundation for later development, was never fully integrated into the P-GRADE Portal due to time constraints. Future work in this area should focus on greater modularity between the base Portal GUI and the Parametric Study GUI. After this modularity is established then integration of the two systems should be much faster to accomplish and should result in better system interoperability.

6.4 Recommendations for Szatki

As new members of the Sztaki development process, we were able to see the current development process from an outsider's perspective. This allowed us realize possible improvements in methods of implementation that a veteran of the team may not consider. In order to improve future development by new members of the Sztaki team, we have described these improvements below.

6.4.1 Development of a Sztaki Defined JSP Tag Library

One of the advantages of writing JSPs is the ability to define a tag library. This allows one to make commonly used designs automatic. Currently, the PGRADE Portal is implemented using only the GridSphere Portalui library. However, close inspection of this library revels that although it implements common HTML features, such as tables and listboxes, it removes functionality from these tags. For example, in current versions of HTML, the width of a button can be set to a defined length. This allows all buttons on a page to be the same size improving the look of the GUI. However, the GridSphere tag library does not allow for this element. For this reason, each button on the detailed view of the Parameter Study Manager is a different size. If, instead of using the GridSphere

library, a PGRADE Portal library was defined, these HTML supported features could be created. Furthermore, common necessities, such as validation of text fields, could be defined in a tag. This would allow a developer to create numerous textfields that accepted only integers without having to write a validation function.

6.4.2 External Documentation

The second improvement to the Sztaki development process would be external documentation regarding how to complete common development operations. Such operations would include setting up a grid on the PGRADE Portal, how to create and debug a Portal, and what information is stored in each trace file. These tasks, while simple, are not known to new developers. In our situation, we discovered how to complete these tasks late in our development process. If external documents had been presented to us regarding these matters at the beginning of development, the overall process would have been much smoother.

6.4.3 Code Modularity

One final improvement to the Sztaki development process would be to highly encourage greater code modularity among their developers. Development of the parametric study extension of the workflow editor would have gone much smoother and quicker if the functionality and physical representation of the components had been modularized according to common coding practices. It is always a good practice to separate an object's functionality and it's GUI representation. All functionality though within the workflow editor was kept directly within the GUI so extension of these classes was greatly hampered. Any future GUI development for the P-GRADE Portal should

strive to implement this type of code modularity by separating their classes away from their GUI representations. This will greatly improve any future extensions of the Portal.

Bibliography

[1] About the Globus Toolkit. Retrieved from http://www-unix.globus.org/toolkit/about.html

[2] About the Institute. (2000) http://www.sztaki.hu/sztaki/about.jhtml

[3] Bailey, A. (2005) Introduction to Java, Part 2 – JSP. Retrieved from http://www.macromedia.com/v1/handlers/index.cfm?ID=16558

[4] Basten, C., (March, 2002) Random Number Seed. Retrieved from http://statgen.ncsu.edu/qtlcart/manual/node31.html

[5] Benczúr, A., (2002) DemoGRID - Connecting Heterogeneous Systems to Solve Data and CPU Intensive Problems. Retrieved from http://caesar.elte.hu/eltenet/projects/demogrid/index.html

[6] Burden, P. Introduction to JSP. Retrieved from http://www.scit.wlv.ac.uk/~jphb/sst/jsp/intro.html

[7] Cannings, R., (2000) Java Intermediate Course. Retrieved from http://cannings.org/oldCourses/JavaIntermediate/swing.html

[8] CERN, (2005) The Grid Café. Retrieved from http://gridcafe.web.cern.ch/gridcafe/index.html

[9] Comparison of ServerSideScripting Techniques. Retrieved from http://www.b2bsim.de/documents/wewior/main.html

[10] Currle-Linde, N., Boes, F., Lindner, P., Pleiss, J., & Resch, M. (2004). A Management System For Complex Parameter Studies and Experiments in Grid Computing [Electronic Version].

[11] Developing and Deploying Portlets. (2003) Retrieved from http://docs.sun.com/source/816-6758-10/ch6.html

[12] Eclipse contributors and others, (2004) The Eclipse Project. Retrieved from http://www.eclipse.org

[13] EGEE: Enabling Grids for E-sciencE. (2005) Retrieved from http://www.egee.hu/

[14] EU Closing In On Transparent Grid. *GRID Today* 3(3). (January, 2004). Retrieved from http://www.gridtoday.com/04/0119/index.html

[15] Fred, A., & Lindesmith, S., (February, 2003) The case for Portlets, How to decide if Portlets are your best option. Retrieved from http://www-106.ibm.com/developerworks/ibm/library/i-Portletintro/

[16] Gagliardi, F., (2004) DataGrid Research and Technological Development for an International Data Grid. Retrieved from http://www.datagrid.cnr.it/

[17] Gervasi, O., (2005) Simbex: A metalaboratory for the a priori simulation of crossed molecular beams experiments. 2005 Retrieved from http://www.lpds.sztaki.hu/index.php?load=projects/current/simbex.php

[18] GreenSuite Technical Glossary (2005). Retrieved from http://www.greensuite.com/whitepapers/whitepapers_03.html

[19] Gridsphere Portal Framework, About. Retrieved from http://docs.sun.com/source/816-6758-10/ch6.html

[20] Gourgoulis, A., Kacsuk, P., Terstyanszky, G., & Winter, S., Using Clusters for Traffic Simulation

[21] Hall, John F. (1995) *Parameter study of the response of moment-resisting steel frame buildings to near-source ground motions.* Technical Report: CaltechEERL:1995.EERL-95-08. California Institute of Technology.

[22] Hepper, S., & Hesmer, S. (August, 2003). Introducing the Portlet Specification. *Java World*. Retrieved from http://www.Javaworld.com/Javaworld/jw-08-2003/jw-0801-Portlet.html

[23] IST Working Group, (2004) Automatic Performance Analysis: Real Tools (APART2). Retrieved from http://www.fz-juelich.de/apart/

[24] JavaServer Pages – Apache Tomcat. (2004) Retrieved from http://Java.sun.com/products/jsp/tomcat/

[25] JGrid: An Integrated Graphical Application Development and Grid Execution Environment Based on Jini. (2004) Retrieved from http://pds.irt.vein.hu/jgrid/

[26] Kacsuk, P., (2002) Development of Virtual Supercomputing Service Using Academic Network. Retrieved from http://www.lpds.sztaki.hu/index.php?load=projects/completed/ni2000.php

[27] Kacsuk, P., (2004) Chemistry GRID and its application for air pollution forecast. Retrieved from http://www.lpds.sztaki.hu/index.php?load=projects/current/ikta5-137.php

[28] Kacsuk, P., (2002) Graphical Supervising System for Geographically Distributed, Heterogeneous Metacomputing Environment. Retrieved from http://www.lpds.sztaki.hu/index.php?load=projects/completed/otka.php

[29] Kacsuk, P., (2003) Hungarian Supercomputing GRID. Retrieved from http://www.iif.hu/mszgrid/

[30] Kacsuk, P., Dozsa, G., Kovacs, J., Lovas, R., Podhorszki, N., Balaton, Z., & Gombas G., P-GRADE: a Grid Programming Environment. Retrieved from http://lpds.sztaki.hu

[31] Kacsuk, P., Sipos, G., & Farkas, F., (2004) World-wide Parallel Processing by the P-GRADE Grid Portal Retrieved from http://lpds.sztaki.hu

[32] Klaene, M., Understanding the Java Portlet Specification. Retrieved from http://www.developer.com/Java/web/article.php/3366111

[33] Lagzi, I., Lovas, R., & Turanyi, T., (2004) Development Of A Grid Enabled Chemistry Application. In Distributed and Parallel Systems: Cluster and Grid Computing, Kluwer International Series in Engineering and Computer Science, Vol. 777, pp. 137-144

[34]Legion A Worldwide Virtual Computer. Retrieved from http://legion.virginia.edu/overview.html

[35] Lovas, R., Kacsuk, P., Horváth, A., & Horányi, A., (2002) Application of P-GRADE Development Environment in Meteorology. In Proceedings of the 4th Austrian-Hungarian Workshop on Distributed and Parallel Systems, DAPSYS 2002, pp. 109-116.

[36] Lovas, R., Kacsuk, P., Lagzi, I., & Turanyi, T., (2004) Unified development solution for cluster and grid computing and its application in chemistry. In Computational Science and Its Applications – ICCSA 2004, LNCS 3044, 226-235

[37] [34] Lopez, I., Follen G., Gutierrez, R., Foster, I., Ginsburg, B., Larsson, O., Martin, S., Tuecke S., & Woodford D. NPSS on NASA's IPG: Using CORBA and Globus to Coordinate Multidisciplinary Aeroscience Applications [Electronic Version]

[38] Mazzoni, S., (April, 2005). …Run Parameter Study. Retrieved from http://peer.berkeley.edu/~silvia/OpenSees/gettingstarted/765.htm

[39] Nabrzyski, J., (2005) *GridLab* - A Grid Application Toolkit and Testbed. Retrieved from http://www.gridlab.org

[40] Novotny, J., Russell M., Wehrens, O., (2004) GridSphere and the GridLab Project. Retrieved from http://www.gridsphere.org/gridsphere/wp-4/Slides/gridsphere/pdf/GridLabOverview.pdf

[41] Oak Ridge National Laboratory, (2005) PVM – Parallel Virtual Machine. Retrieved from http://www.csm.ornl.gov/pvm/

[42] PGRADE. (2003) Retrieved from http://www.lpds.sztaki.hu/pgrade/main.php?m=1

[43] Portlet. (January, 2005) In *Wikipedia the Online Encyclopedia*. Retrieved from http://en.wikipedia.org/wiki/Portlet

[44] SARA Computing and Networking Services, (2005) Backgrounds of Parallel Computing. Retrieved from http://www.sara.nl/userinfo/reservoir parallel_general/index.html

[45] The PGRADE Portal. (2005) Retrieved from http://www.lpds.sztaki.hu/pgPortal20/

[46] The Globus Alliance, (2005) The Globus Project. Retrieved form http://www.globus.org/

[47] University of Tennessee, (1999) MPI – Message Passing Interface. Retrieved from http://www-unix.mcs.anl.gov/mpi/

[48] University of Wisconsin-Madison, (2005). Condor High Throughput Computing, Retrieved from http://www.cs.wisc.edu/condor/

[49] Ward, M., Mandelbrot and Julian Sets. Retrieved from http://davis.wpi.edu/~matt/courses/fractals/mendel.html

[50] Woller, J., (1996) The Basics of Monte Carlo Simulations. Retrieved from http://www.chem.unl.edu/zeng/joy/mclab/mcintro.html

[51] Yarrow, M., McCann, K., Biswas, R., & Van derWijngaart, R., (2000). An Advanced User Interface Approach for Complex Parameter Study Process Specification on the Information Power Grid [Electronic Version].

Appendix A Algorithm to Determine the Range of Indexes for Submission Algorithm

Algorithm to Find the Maximum Number of Combinations:

```
For  j = 0  → number of Keys
        numCombinations = numCombinations * (MaxOfKey[6]-MinOfKey[6])
```

Algorithm to Find all the Permutations of the Key's Indexes:

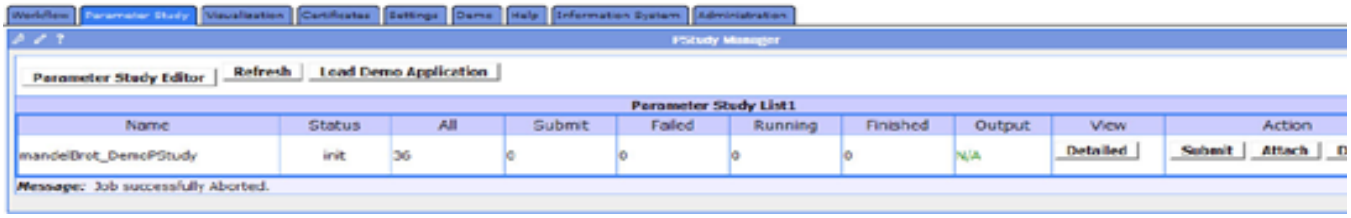
```
if NumberOfCombinationsLeft == 0
        break
else
        for i = minNumIndexes[CurrentKey] → maxNumIndexes[CurrentKey]

                ListOfIndexes[CurrentKey] = i
                NumberOfCombinationsLeft = findIndexes(ListOfIndexes)


findIndexes(ListOfIndexes)
    if NumberOfKeysLeft == 0
            NumberOfCombinationsLeft--

    else
            for i = minNumIndexes[CurrentKey] → maxNumIndexes[CurrentKey]
                    ListOfIndexes[CurrentKey] = i
                    NumberOfCombinationsLeft = findIndexes(ListOfIndexes)
```

Appendix B External Testing Code for the Parameter Study Manager

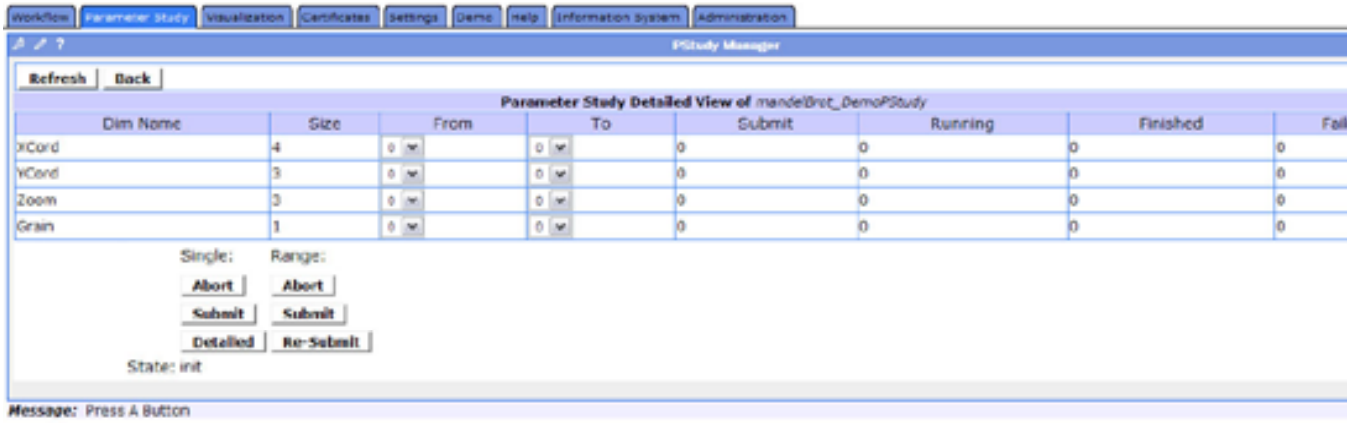
Testing Code for the Determining the Range of Indexes for Submission Algorithm

```
public class Tester
{
   private  static int numCombinationsLeft = 0
   public static void main(String[] args)
   {       tester();    }

   public static void tester()
   {
            int numKeys = 3;
            int[] minNumValues = new int [numKeys];
            minNumValues[0] = 0;
            minNumValues[1] = 0;
            minNumValues[2] = 0;
            int[] maxNumValues = new int [numKeys];
            maxNumValues[0] = 2;
            maxNumValues[1] = 4;
            maxNumValues[2] = 3;

            int numCombinations = findMaxCombinations(numKeys, maxNumValues,   minNumValues);

            String[] indexes = new String[numCombinations];
            int tmpNumKeys = numKeys;
            numCombinationsLeft = numCombinations;
            findIndexes(numKeys, tmpNumKeys, numCombinations, maxNumValues, minNumValues,
            indexes);

            String[] stringIndexes = new String[numCombinations];
            int[] intIndexes = new int[numKeys];

            for(int i = 0; i<numCombinations; i++)
                     System.out.println(indexes[3]);
}
```

Appendix C Screenshots of the Final Parameter Study Manager



Final view of the index.jsp



Final view of the detailed.jsp