Visual Parameteric Modeler for Rapid Composition of Parameter-Sweep Applications for Processing on Global Grids

Shoaib Burq[1], Steve Melnikoff[1], Kim Branson[2], and Rajkumar Buyya[1,*]

[1] Grid Computing and Distributed Systems Lab
Dept. of Computer Science and Software Engg.
The University of Melbourne, Australia
[2] Structural Biology
Walter and Eliza Hall Institute
Parkville, Melbourne, Australia

Abstract. Grids are emerging as a platform for the next-generation parallel and distributed computing. Large-scale parametric studies and parameter sweep applications find a natural place in the Grid's distribution model. There is little or no communication between jobs. The task of parallelising and distributing existing applications is conceptually trivial. These properties of parametric studies make it an ideal place to start developing integrated development environments (IDEs) for rapidly Grid-enabling applications. However, there is a lack of the availability of IDEs for scientists to Grid-enable their applications, without the need of developing them as parallel applications explicitly. This paper presents a Java based IDE called Visual Parameteric Modeler (VPM), developed as part of the Gridbus project, for rapid creation of parameter sweep applications. It supports automatic creation of parameter script and parameterisation of input data files, which is compatible with the Nimrod-G parameter specification language. The usefulness of VPM is demonstrated by a case study on a composition of molecular docking application as a parameter sweep application. Such applications can be deployed on clusters using the Nimrod/enFuzion system and on global Grids using the Nimrod-G grid resource broker.

1 Introduction

As high-speed networks become ubiquitous and research in middleware technologies matures, new windows of opportunity for application scientists to run their applications on parallel and distributed computing environments, such as clusters and Grids [3], are increasing. The underlying infrastructure, providing the low-level facilities to run applications in a heterogeneous and distributed environment; and high-level tools that facilitate the creation of Grid applications and their deployment on distributed resources, makes up the Grid.

There exist a number of models for the construction of parallel and distributed applications. Parameter sweep is one of the simplest and most practical of the models that can yield powerful results. Parameter sweep applications consist of programs that

* Correspondence to: Rajkumar Buyya, email: raj@cs.mu.oz.au

P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2659, pp. 739–749, 2003.

are run independently on different nodes with different input parameters or data sets. There are numerous application areas where parametric studies find a use. Some application scenarios include:

- molecular biologist (drug designer) looking for compounds, in a large chemical data sets , that best dock with a particular protein [8];
- geologist looking at the change in the density and depth of ore-body and the overlying rock's density to optimize cost and production;
- aerospace engineer understanding the role of geometry parameters in the aerodynamic design and optimization process [11];
- high energy physicist investigating on the origin of mass by analysing petabytes of data generated by high-energy accelerators such as the LHC (Large Hadron Collider) [14]; and
- neuroscientist performing brain activity analysis by conducting pair-wise cross co-relation analysis of MEG (Magneto-EncephaloGraphy) sensors data [13].

The practical implications of performing parametric studies make it difficult for an application scientist, who has little or no knowledge of distributed computing, to use it effectively. The vision of the Grid is precisely to bridge this gap by providing a seamless access to compute and other scientific resources without the need of users concerning about the lower-level details of the computing infrastructure or the resource management issues [1]. High-level tools for creation of distributed applications and their deployment on the Grid make up an essential part of this vision. Currently, there is still lack of the availability of integrated development environments (IDEs) with visual interface for scientists to rapidly Grid-enable their existing applications.

This paper presents a Java based IDE called Visual Parameteric Modeler (VPM), developed as part of the Gridbus project, for rapid creation of parameter sweep applications. VPM provides a simple visual interface for the manipulation of scripts or input files of existing applications. Users can assign parameters to certain values by highlighting them. They can select from a number of different data types and domains to describe their parameters. VPM also incorporates a task editor for creating the tasks carried out by different jobs during different stages of a distributed execution. The parameters and tasks together provide the basis of each run. VPM allows the rapid creation and manipulation of the parameters. While being flexible, it is also simple enough for a non-expert to create a parameter script, known as a plan file. The parameter sweep applications composed using VPM can be deployed on global Grids using the Nimrod-G resource broker that supports scheduling based on the user's quality of service (QoS) requirements – such as the deadline, budget, and optimization preference – and the access price of resources.

The rest of this paper is organised as follows. Section 2 presents related tools and their capabilities including differences. The VPM architecture is discussed in Sect. 3 and the design and implementation is discussed in Sect. 4. The use of VPM for composing molecular docking application as a parameter sweep application is presented in Sect. 5, followed by a conclusion in Sect. 6.

2 Related Work

VPM draws inspiration from or builds on the concepts developed in Nimod [2] and its commercial version (Enfuzion [1]); and its Grid-enabled version (Nimrod-G [9]) that support the creation and execution of parametric applications on clusters and Grids respectively. A declarative language, called parameter specification language, supported by Nimrod describes the parameters and the tasks that make up the plans.

For the creation of plans, Enfuzion takes a wizard approach. Enfuzion will take a user through the operation of creating a job specification file step-by-step, because it is too complex for novice users to create parameter script on their own. In the input file to the application, the user must change the value assigned to a parameter to a place marker. Although simple and less prone to error, this approach is too rigid, slow and cumbersome for someone working on several input files at the same time. As the parameter script and parameterized input data files generated by VPM confirm to the Nimrod parameter specification language, it serves as a complimentary tool. This ensures that VPM can be used by EnFuzion and Nimrod-G users.

Using VPM, the users can select all application input data/configuration files and parameterise easily. The users can drag and select the value in the input file that they wish to assign a parameter to, or they can create parameters independent of an input file. This gives the user a great deal of flexibility and control. By giving the user fields to input their parameter configuration and then generating the plan specification automatically we can prevent errors. Even if the users create parameter script in their favorite editor, VPM allows them to import and make use of its capabilities. Once the plan specification is created, the users proceed to execution phase during which they have an option of changing values assigned parameters. Like enFuzion, the VPM will automatically create application jobs each with different parameter values will be created. Such jobs can be analysed on clusters or Grids using enFuzion or Nimrod-G respectively.

Other related works include, APST (AppLeS Parameter Sweep Template) [10] and NASA IPG (Information Power Grid) parameter process specification tool [11]. APST expects application scientists to explicitly create jobs and assign parameter values to them. IPG provides graphical environment for parameterising the data files. Both the APST and IPG schedulers use traditional system centric policies for resource allocation. As VPM confirms to the Nimrod-G parameter specification language, it enables the users to harness Grid resources using the Nimrod-G resource broker depending on their QoS requirements and the access price of resources. Thus, it supports the Grid economy, which is essential for management and allocation of resources based on the supply and demand.

3 Architecture

The visual parameteric modeler architecture and parameter sweep application creation flow model is shown in Fig. 1. VPM supports the creation of a new parameter sweep applications from scratch or the utilisation of the existing parameterised application plans with further update. In the first case, the users can add all those files to be pa-

rameterised and use VPM to parameterise data items of interest. In the second case, the users can import the existing parameteric plans and pass through the VPM scanner and parser that identify parameters and make them available for further update. The users can use the VPM task editor to create a task to be associated with jobs. Based on parameter types and their values a number of jobs, each representing a different parameter scenario, are generated automatically.

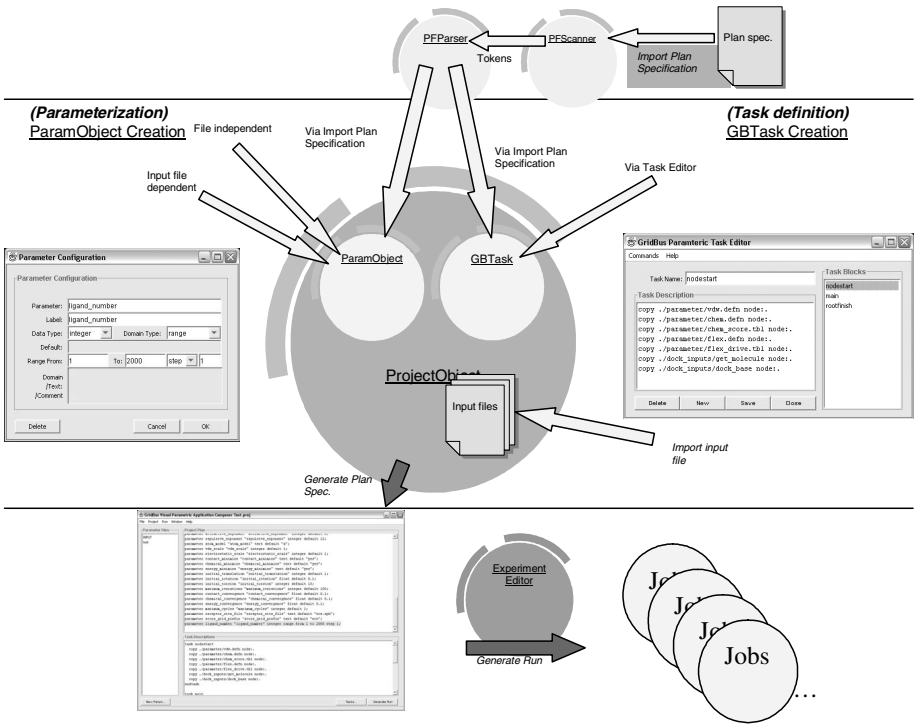

Fig. 1. The visual parameteric modeler architecture

VPM consists of three major visual components: Project, Input Files and Tasks. These components are represented as Project Window, Input File Window and Task Editor, respectively. The design of VPM, shown in Fig. 2, allows a single project to have several input data files and tasks.

These visual components provide the user access to the objects that encapsulate the plan's information-model, namely to ParamObject and GBTask. ParamObject is created and manipulated from the project window or input file window, while the GBTask is created and manipulated using the TaskEditor.

A plan consists of parameters and task. In VPM, parameters are internally represented as ParamObjects and tasks as GBTasks. ParamObjects are created by any of the following three methods.

- File dependent parameterization
- File independent parameterization
- Via imported plan specification

File Dependent Parameterization

Once an input file or a script file is imported into VPM, values that have to be assigned parameters are highlighted and the parameter defined and assigned by a simple click of mouse.

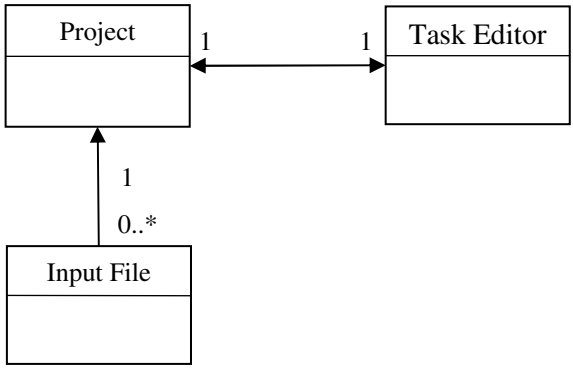

Fig. 2. Basic visual components of VPM

File Independent Parameterization

New parameters may also be created by simply defining its properties.

Via Imported Plan Specification

VPM contains a LALR (Look Ahead, Left to Right) parser for plan specification that confirms to the Nimrod parameter specification language. This allows the reuse of an existing plan file (parameter script). The parser translates each parameter definition into a ParamObject and each task description into a GBTask (see Figure 1).

Experiment Editor and Job Generation

Once a plan specification is completed, VPM can generate a run specification. This enumerates every value lying within the range of the parameters described by the plan, and a description of the jobs in terms of the values assigned to them. Hence, the run specification describes the distribution model of the application parameterized using VPM.

4 Design and Implementation

VPM is coded in Java and MVC (Model-View-Controller) architecture [12] design pattern that decouples the data model from the component that represents it on the screen. The graphical user interfaces are created using the Java Swing component set that uses MVC architecture consistently.

Besides the above-mentioned objects, VPM has various components that facilitate the creation of a plan specification (parameter script) and parameterisation on input data files. VPM consists of many packages and associations and reverse-associations between them are shown in Fig. 3. The arrow heads point at the dependent packages. Notice, a single class, Jobs, in GBJobs package, is responsible for the production of Grid enabled jobs. This can be extended to support creation of job specification for different scheduling systems.

ExperimentEditor

This contains the GUI classes for the ExperimentEdior. It also contains a controller class (following the classic MVC architecture) that processes the user input.

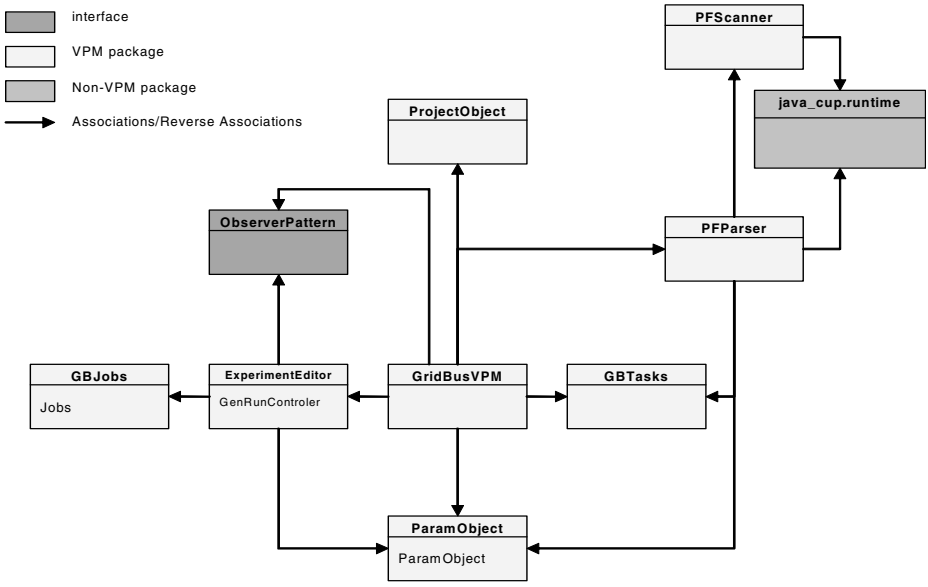

Fig. 3. VPM package associations and reverse-associations

GBJobs

This contains a single class, `Jobs`. "`Jobs`" takes as its input a count (N) of those parameters that have a range of values and an array of integers of size N containing the maximum value taken by each of these parameters.

GBTask

This package contains a single class, `GBTask`. It is a serializable object. It encapsulates the commands that execute during different phases of the distributed run.

GridBus

This is the largest package containing mostly the GUI classes for VPM. Following the MVC architecture, it contains all the "view" components. It also includes a utility class, called `GBFileManager`, for handling all file operations within VPM. In addition, this package contains the class that has VPM's main method, named `Project`.

ObserverPattern

This package contains two interfaces `Observer` and `Subject`. This facilitates the implementation of MVC architecture, by decoupling related objects [4]. A subject may have a number of observers. All observers are notified when the subject undergoes a state change. In response, the observer may query the subject to synchronize its state with the subject. The observer implements the `update()` method while the subject implements the `addObserver()` and `removeObserver()` method. On a state change, the subject calls each observer's update method.

ParamObject

This package contains a single class: `ParamObject`. The `ParamObject` is the heart of VPM. It is a serializable object encapsulating the state of a parameter, it contains two key methods: `makePlanStep()` and `makeRunStep()`. These methods are responsible for automating the process of plan and run specification creation. The `makePlanStep` method converts the fields of the ParamObject into a line of the simple declarative language following the grammar of Fig. 4. `makeRunStep` converts a parameter's declaration into a statement of a run specification. This declaration identifies the possible value(s) taken by the parameter. Currently `makeRunStep` generates a Nimrod-G readable statement.

$plan \rightarrow plan\ \ rest \mid error \mid \varepsilon$

$rest \rightarrow planStep \mid taskBlock \mid newline$

$planStep \rightarrow PARAMETER\ \ ID\ \ label\ \ type\ \ domain\ \ SEMI$

$label \rightarrow LABEL\ \ QUOTE \mid QUOTE \mid \varepsilon$

$type \rightarrow INTEGER \mid FLOAT \mid TEXT \mid FILE$

$domain \rightarrow DEFAULT\ \ value\_opt$

$\mid RANGE\ \ range\_values$ domain2

$\mid SELECTANY\ \ value\_list\ \ default\_opts$

$\mid SELECTONE\ \ value\_list\ \ default\_opt$

$\mid RANDOM\ \ range\_values$ points_opt

| COMPUTE expr

| JITP jitp_expr

points_opt → POINTS value_opt | ε

default_opts → value_opt value_list | value_opt

domain2 → POINTS value_opt | STEP value_opt | ε

value_opt → ID | QUOTE | NUM

expr → expr PLUS term | expr MINUS term | term

term → term TIMES factor | factor

factor → NUMBER | LPAREN expr RPAREN

Fig. 4. Context free grammar for plan specification

PFScanner
PFScanner, (plan file scanner) created using an open source tool called JLex [5], performs lexical analysis of the plan specification. It comes into play when the user wishes to import an existing plan specification into VPM. It interfaces with the PFParser (discussed below) providing it with a stream of identified tokens.

PFParser
PFParser, (plan file parser) written using an open source tool called CUP [6], interfaces with the PFScanner and attempts to match the stream of tokens to a complete parameter or task definition as described by the A context-free grammar shown in grammar in Fig. 4. All caps denote the terminals. In doing so, it generates new `ParamObjects` or `GBTasks`. It contains two public methods for the retrieval of ParamObject and GBTasks: `getParams()` and `getTasks()`.

ProjectObject
`ProjectObject` encapsulates all the attributes necessary to describe a VPM project. It contains the `ParamObjects`, `GBTasks`, paths to input files and other attributes that uniquely identify a project.

5 Use Case Study – Molecular Docking Application

Molecular modeling for drug design involves screening millions of ligand records or molecules of compounds in a chemical database (CDB) to identify those that are potential drugs. This process is called molecular *docking* [7]. It helps scientists explore how two molecules, such as a drug and an enzyme or protein receptor, fit together. Docking each molecule in the target chemical database is both a compute and data intensive task. In [8], a virtual laboratory environment has been developed and demonstrated distributed execution of molecular docking application on Global Grids. The application has been formulated as a parameter sweep application using a simple parameter specification language and deployed on global Grids using the Nimrod-G resource broker.

We now discuss how the application has been parameterized (i.e., the creation of parameter script and parameterisation of data files) using the VPM. In [8], the creation of parameter script and parameterisation of data/configuration files has been carried out manually using a text editor. Although this task is simple, it becomes cumbersome when an application contains multiple data files and has a large number of data entries to be parameterised. This approach is also prone to creating parameter script with syntax errors. The use of visual modeler helps overcome these limitations and aids in the rapid parameterisation of the molecular docking application such as the "Dock" [7] software package.

Fig. 5 shows the parameterisation of docking application configuration input file using VPM. First, the configuration input file is imported into the VPM. When the value of a data item to be parameterised is selected (see the highlighted text "S_1" in Fig. 5), it appears in the dialogue box where the parameter name can be defined along with the attributes (data type and values). In this example, the name of a data item, "ligand_atom_file", indicates the molecule to be screened. As the aim of parameteri-

sation is to screen multiple molecules, this parameter need to be defined as the "range" data type and then assign values for index start, end, step. For example, to screen the first 2000 molecules in the chemical data base, the initial values to be assigned are 1, 2000, and 1 respectively. VPM will automatically create a parameter statement and add to the script (see the highlighted statement in Figure 6). A task specification creation module provides dialogue facility selection of appropriate commands associated with the execution of a parametric job (see a small window in Fig. 6).

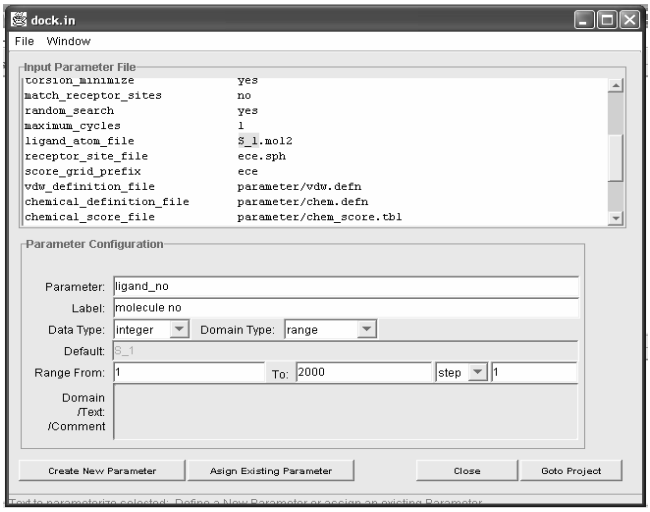

Fig. 5. The parameterisation of docking configuration input file

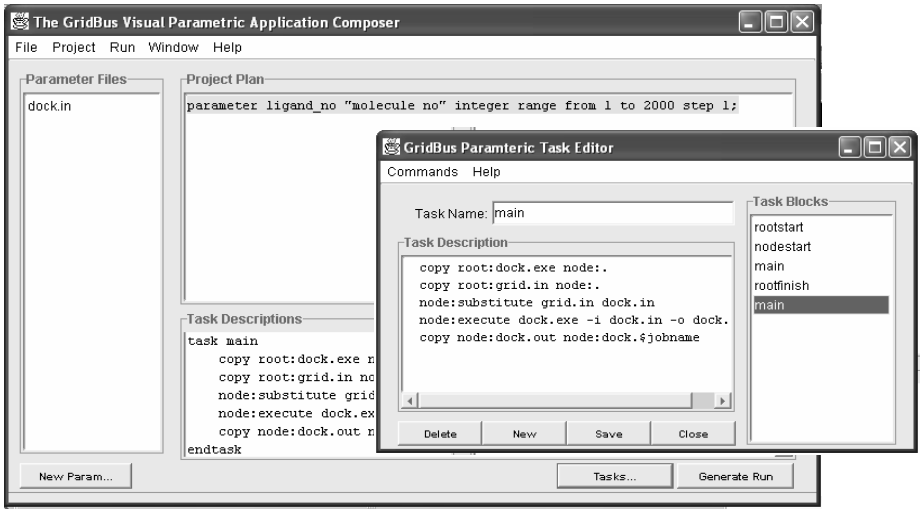

Fig. 6. The creation of docking parameter script

6 Conclusion

In this paper, we outlined the need for the development of IDEs and other applications and tools in order to provide the applications scientist with user-friendly environments to run their code on the Grid. We introduced VPM developed to provide one such environment for parameter sweep applications. We identified its key features, while giving some of its implementation details. Finally, we showed how application scientists can use VPM to parameterise their applications. Such parameterised applications can be deployed on Global Grids using the Nimrod-G resource broker.

Acknowledgements. We thank Srikumar Venugopal, Elan Kovan, Anthony Sulistio, and Sarana Nutanong for their comments. We thank anonymous reviewers for providing excellent comments.

References

[1] D. Abramson et. al., EnFuzion Tutorial, Chapter 4, EnFuzion Manual, 2002. Available at: `http://www.csse.monash.edu.au/cluster/enFuzion/tutorial.htm`

[2] D. Abramson, R. Sosic, J. Giddy, and B. Hall, *Nimrod: A Tool for Performing Parameterised Simulations using Distributed Workstations*, Proceedings of the 4th IEEE Symposium on High Performance Distributed Computing, Virginia, August 1995.

[3] I. Foster and C. Kesselman (editors), *The Grid: Blueprint for a Future Computing Infrastructure*, Morgan Kaufmann Publishers, USA, 1999.

[4] G. Krasner and S. Pope, *A cookbook for using the model-view controller user interface paradigm in Smalltalk-80,* Journal of Object-Oriented Programming, 1(3):26–49, August/September 1988.

[5] E. Berk, *JLex: A lexical analyzer generator for Java(TM)*, Department of Computer Science, Princeton University Version 1.2.5, September 6, 2000 `http://www.cs.princeton.edu/~appel/modern/java/JLex/`

[6] S. E. Hudson, *CUP: LALR Parser Generator for Java(TM)*, GVU Center, Georgia Tech. Version 0.10, July 1999 http://www.cs.princeton.edu/~appel/modern/java/CUP

[7] T. Ewing (editor), *DOCK Version 4.0 Reference Manual*, University of California at San Francisco (UCSF), USA, 1998. Online version: `http://www.cmpharm.ucsf.edu/kuntz/dock.html`

[8] R. Buyya, K. Branson, J. Giddy, and D. Abramson, *The Virtual Laboratory: Enabling Molecular Modelling for Drug Design on the World Wide Grid*, Journal of Concurrency and Computations: Practice and Experience, Wiley, USA, Jan 2003.

[9] R. Buyya, D. Abramson, and J. Giddy, *Nimrod-G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid*, The 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000), Beijing, China, May 2002.

[10] H. Casanova, G. Obertelli, F. Berman, and R. Wolski, *The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid*, Proceedings of the Super Computing (SC 2002) Conference, Dallas, USA.

[11] M. Yarrow, K. McCann, R. Biswas, and R. Van der Wijngaart, *An Advanced User Interface Approach for Complex Parameter Study Process Specification on the Information Power Grid*, Proceedings of the 1st Workshop on Grid Computing (GRID 2002), Bangalore, India, Dec. 2000.

[12] Java and MVC architecture, http://javanook.tripod.com/patterns/java-mvc.html
[13] R. Buyya, S. Date, Y. Mizuno-Matsumoto, S. Venugopal, and D. Abramson, *Economic and On Demand Brain Activity Analysis on Global Grids*, Technical Report, Grid Computing and Distributed Systems (GRIDS) Lab, The University of Melbourne, Australia, Jan. 2002.
[14] CERN, the LHC Grid Project, http://lcg.web.cern.ch/LCG/