Visual Modeler for Grid Modeling and Simulation (GridSim) Toolkit

Anthony Sulistio, Chee Shin Yeo, and Rajkumar Buyya

Grid Computing and **D**istributed **S**ystems (GRIDS) Laboratory,
Department of Computer Science and Software Engineering,
The University of Melbourne, Australia
ICT Building, 111 Barry Street, Carlton, VIC 3053
{anthony, csyeo, raj}@cs.mu.oz.au
http://www.gridbus.org

Abstract. The Grid Modeling and Simulation (GridSim) toolkit provides a comprehensive facility for simulation of application scheduling in different Grid computing environments. However, using the GridSim toolkit to create a Grid simulation model can be a challenging task, especially when the user has no prior experience in using the toolkit before. This paper presents a Java-based Graphical User Interface (GUI) tool called Visual Modeler (VM) which is developed as an additional component on top of the GridSim toolkit. It aims to reduce the learning curve of users and enable fast creation of simulation models. The usefulness of VM is illustrated by a case study on simulating a Grid computing environment similar to that of the World-Wide Grid (WWG) testbed [1].

1 Introduction

Grid computing has emerged as the next-generation parallel and distributed computing that aggregates dispersed heterogeneous resources for solving all kinds of large-scale parallel applications in science, engineering and commerce [2]. This introduces the need to have effective and reliable resource management and scheduling systems for Grid computing. There is also the need to administer resources and application execution depending on either resource users' or owner's requirements, and to continuously keep track of changes in resource availability.

Managing various resources and applications scheduling in highly distributed heterogeneous Grid environments is a complex and challenging process [3]. A generic view of the World Wide Grid (WWG) computing environment is shown in Figure 1. The Grid resource broker hides the complexities of the Grid computing environment from a user. It discovers resources that the user can access using information services, negotiates for access costs using trading services, maps application jobs to resources, starts execution and monitors the execution progress of tasks.

Different scenarios need to be evaluated to ensure the effectiveness of the Grid resource brokers and their scheduling algorithms. Given the inherent heterogeneity of a Grid environment, it is difficult to produce performance evaluation in a

P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2659, pp. 1123–1132, 2003.

repeatable and *controllable* manner. Therefore, the GridSim toolkit is developed to overcome this critical limitation. The GridSim toolkit is a Java-based discrete-event Grid simulation toolkit that provides features for application composition, information services for resource discovery, and interfaces for assigning application tasks to resources and managing their execution. The GridSim toolkit has been applied successfully to simulate a Nimrod-G [4] like Grid resource broker and to evaluate the performance of deadline and budget constrained cost- and time- optimization scheduling algorithms [3].

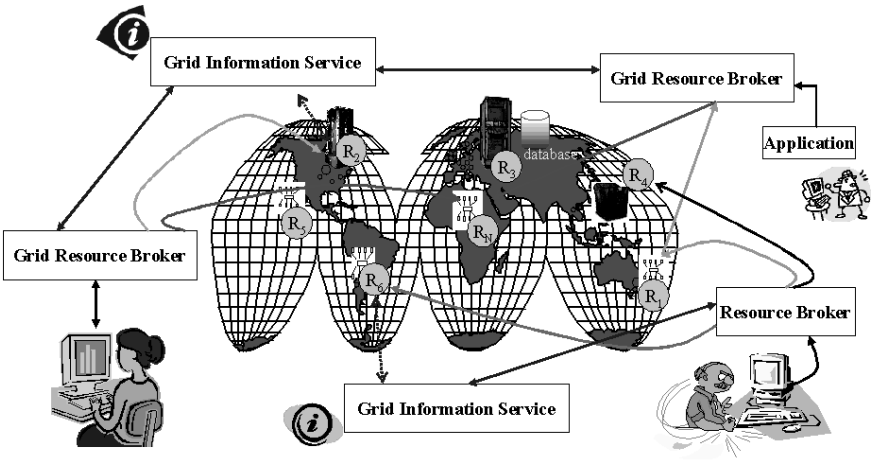

Fig. 1. A generic view of World-Wide Grid (WWG) computing environment

Since the GridSim toolkit is an advanced and powerful simulation toolkit, its users will experience a high learning curve in order to utilize the toolkit functionalities for effective simulations. In addition, the users need to write Java code that use the toolkit packages to create the desired experimental scenarios. This process is repetitive and tedious. It is specially disadvantageous to those who have little or no experience in Object-Oriented (OO) concepts in general, and Java programming in specific. Therefore, it is necessary to build a Graphical User Interface (GUI) for GridSim that enables its users to create and modify simulation models easily and quickly as many times as required.

This paper presents a Java-based GUI tool called Visual Modeler (VM) that is designed as a separate additional component residing in a layer above the GridSim toolkit. Figure 2 illustrates the role of VM in creating a simulation model. A GridSim user utilises VM as a tool to create and modify the Grid simulation model (see Figure 2 step 1A). VM will then generate Java code for the simulation (see Figure 2 step 1B). In the absence of VM, the user needs to write Java code manually using a text editor or development tools such as JBuilder (see Figure 2 step 2). However, this approach is prone to create programs with syntax errors.

VM provides a simple and user-friendly GUI to facilitate its user to be able to create and modify different simulation models easily. This relieves the users from spending a lot of time and effort trying to understand the GridSim toolkit code. VM is designed to enable the users to create simulation models without the need to know the actual Java code behind it or the GridSim toolkit code. In addition, it automatically generates Java code that uses the GridSim toolkit, so users can compile and run the simulation. Therefore, the aim of VM is to reduce the learning curve of GridSim users and to enable them to build simulation models easily and effectively. The initial prototype of VM has been implemented and bundled together with the new release version of the GridSim toolkit 2.0 in November 2002.

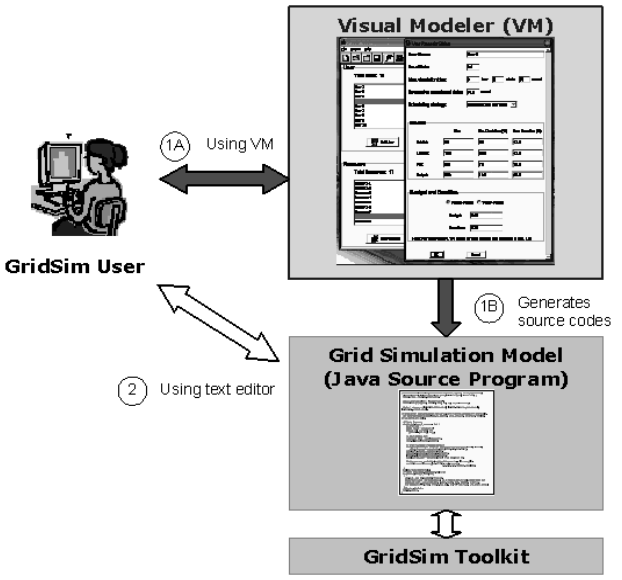

Fig. 2. Relationship between *Visual Modeler*, *Grid Simulation Model* & *GridSim toolkit*

This paper is organized as follows: Section 2 mentions related work. Section 3 presents the architecture and features of VM, while Section 4 discusses the design and implementation of VM. Section 5 illustrates the use of VM for simulating a Grid computing environment. Section 6 concludes the paper and suggests some further work to be done on VM.

2 Related Work

The GridSim 1.0 toolkit does not have any additional GUI tools that enable easier and faster modeling and simulation of Grid environments. It only comprises the Java-based packages that are to be called by users' self-written Java

simulation programs. VM has now been included in the recently released GridSim 2.0 toolkit. There are many similar tools like VM that generate source code, but are not related to Grid simulation in specific, such as SansGUI^{TM} [5] and SimCreator [6].

Other than the GridSim toolkit, there are several other tools that support application scheduling simulation in Grid computing environments. The notable ones include Bricks [7], MicroGrid [8] and SimGrid [9]. For a brief comparison of these tools with the GridSim toolkit, please refer to [3].

3 Architecture

VM adopts Model View Controller (MVC) architecture as illustrated in Figure 3. The MVC architecture is designed to separate the user display from the control of user input and the underlying information model. The following are the reasons for using MVC architecture in VM [10]:

- opportunity to represent the same domain information in different ways. Designers can therefore refine the user interfaces to satisfy certain tasks and user characteristics.
- ability to develop the application and user interface separately.
- ability to inherit from different parts of the class hierarchy.
- ability to define control style classes which provide common features separately from how these features may be displayed.

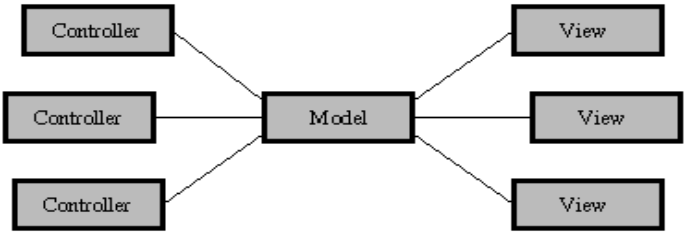

Fig. 3. The basic *Model View Controller* architecture

VM is programmed using Java as Java supports powerful Swing libraries that facilitate easy GUI programming. In addition, Java works well with the MVC architecture. VM support the following main features:

- enables the creation of many Grid testbed users and resources,
- generates the simulation scenario into Java code that the users can compile and run the simulation with the GridSim toolkit,
- saves and retrieves a VM project file that contains an experiment scenario in eXtensible Markup Language (XML) format, and
- works on different operating system platforms (as it is implemented in Java).

4 Design and Implementation

The MVC paradigm proposes three class categories [10]:

1. **Models** – provide the core functionality of the system.
2. **Views** – present models to the user. There can be more than one view of the same model.
3. **Controllers** – control how the user interacts with the view objects and manipulates models or views.

Java supports MVC architecture with two classes:

- **Observer** – any object that wishes to be notified when the state of another object changes.
- **Observable** – any object whose state may be of interest to another object.

4.1 Model

VM consists of three model classes, whose relationships are shown in Figure 4. The models are:

- **FileModel** – deals with UserModel and ResourceModel for saving and retrieving an XML file format for the VM project file.
- **UserModel** – stores, creates and deletes one or more user objects.
- **ResourceModel** – stores, creates and deletes one or more resource objects.

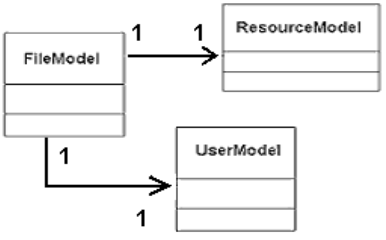

Fig. 4. Class diagram that contains the model classes of VM and their relationships

Both UserModel and ResourceModel contain objects that are hybrids of both model and view, i.e. they have their own widgets to display the stored values. This technique reduces the number of classes needed and the overall design complexity, thus saving significant development time. All models extend Java's Observable class, which provides the behaviours required to maintain a list of observers and notify them when there are changes to the model. The initialization for UserModel and ResourceModel is straightforward with both classes not requiring any parameters. However, FileModel needs to request a reference to each of the UserModel and ResourceModel object. It thus becomes an observer of UserModel and ResourceModel when dealing with the saving and retrieving project file.

4.2 View

Views are registered as dependents of a model. The model broadcasts a message to all dependents when it changes. The main views of VM are:

- **MenuView** – creates GUI components for menu bar.
- **IconView** – creates GUI components for icon toolbar.
- **DisplayView** – creates the main window to display the lists of Grid user and resource.

These three main views are created first, by constructing their widgets independently and then adding their sub-views. This design ensures that the sub-views are separate from their parents' view [10]. Each main view contains one or more references to the model since the model contains user/resource objects that also display widgets. This design reduces the number of method calls and any overheads associated with the interaction between the object and its view.

There are two possible approaches for creating user/resource objects: *with* or *without* their widgets. These approaches are tested by using VM to create 500 Grid users and 1,000 Grid resources. Table 1 shows that creating objects without their widgets requires much less time. However, Lines of Codes (LOC) inside the object class is slightly larger than that of the approach with widgets.

The approach without widgets requires less passing of reference objects and messages in comparison to the other method. In addition, the time taken and the amount of memory needed to create components and to register their event listeners are minimal. VM can thus supports fast creation of a large number user/resource objects at any one time. This is helpful since a Grid simulation model does not have a limited number of Grid users and resources.

Therefore, VM adopts the without-widgets approach by displaying its GUI upon the user's request. When the user closes its GUI window, the GUI components are retained so that they need not be created again for future access (similar to the cache concept in web browsers). This reduces the utilization of memory and enables fast display of GUI components repeatedly.

Table 1. Comparison for the creation of 500 Grid users and 1,000 resources using VM, running on a Pentium II 300 MHz with 64MB RAM

Approach	Time completion	Average LOC
Creating objects *without* their widgets	1 min	950
Creating objects *with* their widgets	20 mins	600

4.3 Controller

The controller needs to be informed of changes to the model as any modifications of the model will also affect how the controller processes its input. The controller

may also alter the view when there is no changes to the model. The primary function of the controller is to manage mouse/keyboard event bindings (e.g. to create pop-up menus). If an input event requires modification of application-specific data, the controller notifies the model accordingly.

In this implementation, controllers which relate to views have several responsibilities. Firstly, they implement Java's event-handler interfaces to listen for appropriate events, e.g. the icon toolbar controller detects the clicking of the save toolbar button and notifies FileModel to save a project file. Secondly, views delegate the construction and maintenance of button panels to controllers. Thirdly, controllers can broadcast semantic events (e.g. save file), to objects who have informed the controller that they are interested in these events [10].

5 Use Case Study – Grid Computing Environment Simulation

This section describes how a simulated Grid computing environment is created using VM. First, the Grid users and resources for the simulated Grid environment have to be created. This can be done easily using the wizard dialog as shown in Figure 5. The VM user only need to specify the required number of users and resources to be created. Random properties can also be automatically generated for these users and resources. The VM user can then view and modify the properties of these Grid users and resources by activating their respective property dialog.

Fig. 5. *Wizard dialog* to create Grid users and resources

Figure 6 shows the property dialog of a sample Grid resource. VM creates Grid resources similar to those present in the WWG testbed [1]. Resources of different capabilities and configurations can be simulated, by setting properties such as cost of using this resource, allocation policy of resource managers (time/space-shared) and number of machines in the resource (with Processing El-

ements (PEs) in each machine and their Million Instructions Per Second (MIPS) rating).

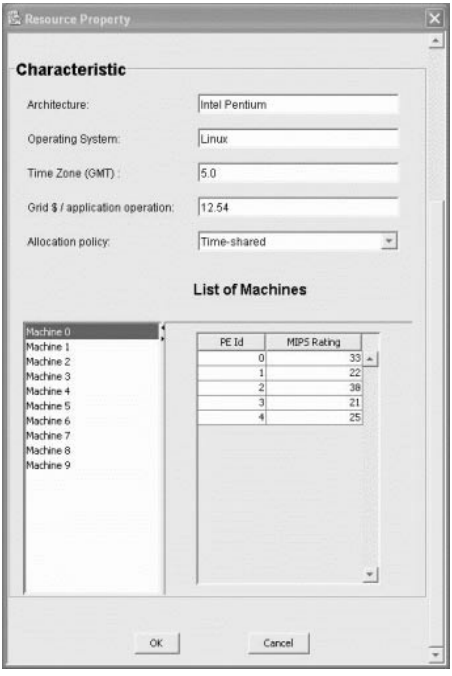

Fig. 6. *Resource dialog* to view Grid resource properties

Figure 7 shows the property dialog of a sample Grid user. Users can be created with different requirements (application and quality of service requirements). These requirements include the baud rate of the network (connection speed), maximum time to run the simulation, time delay between each simulation, and scheduling strategy such as cost and/or time optimization for running the application jobs. The application jobs are modelled as Gridlets. The parameters of Gridlets that can be defined includes number of Gridlets, job length of Gridlets (in Million Instructions (MI)), and length of input and output data (in bytes). VM provides a useful feature that supports random distribution of these parameter values within the specified derivation range. Each Grid user has its own economic requirements (deadline and budget) that constrains the running of application jobs. VM supports the flexibility of defining deadline and budget based on factors or values. If it is factor-based (between 0.0 and 1.0), a budget factor close to 1.0 signifies the Grid user's willingness to spend as much money as required. The Grid user can have the exact cost amount that it is willing to spend for the value-based option.

VM will automatically generate Java code for running the Grid simulation. This file can then be compiled and run with the GridSim toolkit packages to simulate the required Grid computing environment.

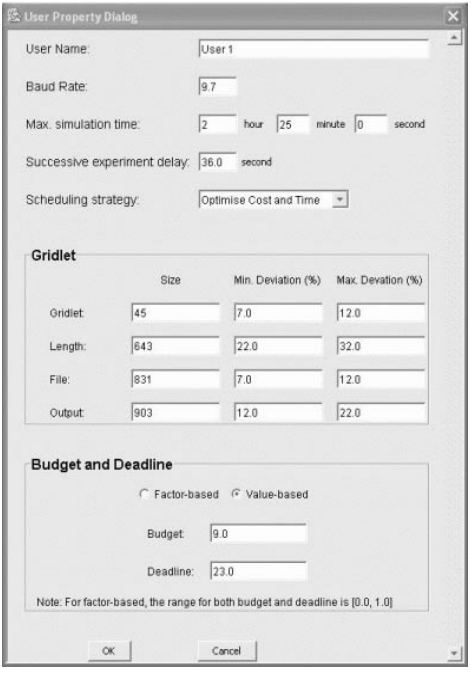

Fig. 7. *User dialog* to view Grid user properties

6 Conclusion and Further Work

This paper describes a Java-based GUI tool called Visual Modeler (VM) that facilitates GridSim users in creating and modifying Grid simulation models easily and effectively. It incorporates features such as easy-to-use wizards that enables users to create simulation models, and automatic source code generation facility that outputs ready-to-run simulation scenario code.

The implementation of VM in Java is ideal since Java supports powerful GUI components with its Swing packages. Moreover, the MVC architecture can be adapted in Java using Observable and Observer class. Hence, Java and MVC provide a perfect combination in creating a GUI for GridSim.

Possible improvements to be implemented include incorporating the GridSim run-time environment into VM, and generating visual diagrams such as graphs based on the GridSim simulation results. This will enable users to receive dynamic run-time feedback from GridSim through VM.

Acknowledgement. We thank Srikumar Venugopal for his comments on the paper. We also thank anonymous reviewers for providing excellent feedbacks.

Software Availability
The GridSim toolkit and VM software with source code can be downloaded from the following website:

`http://www.gridbus.org/gridsim/`

References

1. Buyya, R., Stockinger, H., Giddy, J., Abramson, D.: Economic Models for Management of Resources in Peer-to-peer and Grid Computing. SPIE International Conference on Commercial Applications for High-Performance Computing, Denver, USA (August 20–24, 2001)
2. Foster, I., Kesselman, C. (eds.): The Grid: Blueprint for a Future Computing Infrastructure. Morgan Kaufmann Publishers, USA (1999)
3. Buyya, R., Murshed, M.: GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. The Journal of Concurrency and Computation: Practice and Experience, Vol. 14, Issue 13–15. Wiley Press (2002)
4. Buyya, R., Abramson, D., Giddy, J.: Nimrod-G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid. Proceedings of 4th International Conference and Exhibition on High Performance Computing in Asia-Pacific Region (HPC Asia 2000). IEEE Computer Society Press, Beijing, China (May 14–17, 2000)
5. ProtoDesign Inc. SansGui^{TM}. `http://protodesign-inc.com/sansgui.htm`
6. Realtime Technologies Inc. SimCreator. `http://www.simcreator.com/simtool.html`
7. Aida, K., Takefusa, A., Nakada, H., Matsuoka, S., Sekiguchi, S., Nagashima, U.: Performance Evaluation Model for Scheduling in a Global Computing System. The International Journal of High Performance Computing Applications, Vol. 14, No. 3. Sage Publications, USA (2000)
8. Song, X., Liu, X., Jakobsen, D., Bhagwan, R., Zhang, X., Taura, K., Chien, A.: The MicroGrid: A Scientific Tool for Modelling Computational Grids. Proceedings of IEEE Supercomputing (SC 2000). Dallas, USA (Nov 4–10, 2000)
9. Casanova, H.: Simgrid: A Toolkit for the Simulation of Application Scheduling. Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001). IEEE Computer Society Press, Brisbane, Australia (May 15–18, 2001)
10. Mahemoff, M. J., Johnston, L. J.: Handling Multiple Domain Objects with Model-View-Controller. In: Mingins, C., Meyer, B. (eds.): Technology of Object-Oriented Languages and Systems 32. IEEE Computer Society Press, Los Alamitos, USA (1999) 28–29