Development of Grid-Enabled Problem Solving Environment for PDE Based Applications

Zhou Jun [1] Yukio Umetani [2]

1, Graduate School of Science and Engineering, Shizuoka University, Japan
2, Computer Science Department, Faculty of Informatics, Shizuoka University, Japan

zhoujun@cs.inf.shizuoka.ac.jp umetani@cs.inf.shizuoka.ac.jp

Abstract

This paper describes the functionality and architecture of a Grid-enabled partial differential equation based problem solving environment (PSE) for computational science and engineering. It integrates software tools and expert assistance into a Grid-enabled environment with a set of well-designed graphical user interfaces, which will guide the users to construct, run and analyze their physical simulation intuitively and with confident control. It allows more rapid prototyping of research ideas and enhances higher research productivity while relieves the researchers of the burdens associated with the details of computer science, as well as leaves them free to concentrate on their own research fields. It also provides the users opportunities to conduct remote simulations as well as to utilize Grid resources. Since this system is still under design and implementation, no real running example is given at this time in the paper.

1 Introduction and Motivation

Computation and simulation have become major driving forces in modern scientific research and engineering design, and have been the significant engines for increasing industrial productivity and saving development costs. Problem solving environments (PSEs) are increasingly used as natural platforms to enable collaboration and leverage expertise from different fields.

The definition of PSE was given in [1]: "*A PSE is a computer system that provides all the computational facilities necessary to solve a target class of problems.* These facilities include advanced solution methods, automatic or semiautomatic selection of solution methods and ways to easily incorporate novel solution methods. Furthermore, PSEs use a language natural for the problem class and allow users to solve problems without specialized knowledge of the underlying computer hardware or software. The goal of PSE technology is to exploit the existing enormous hardware and algorithm power to deliver 'cheap and fast' problem solutions".

The potential for very high level and powerful problem solving systems for science was recognized very early, when the DEQSOL [2] and ELLPACK [3] emerged in 1970s as two successful partial differential equation (PDE) solving system forerunners as an extension to Fortran. A wide variety of well-known phenomena in engineering and science governed by PDEs (e.g. structural mechanics, heat transfer, wave propagation, electromagnetic, etc.) had been solved by these two systems successfully and efficiently. In the late 1970s the DEQSOL was measured (compared to Fortran) to provide a decrease in programming efforts by a factor of about 100 and a decrease in execution time by a factor of about three [4].

In the past decades, many commercial software packages or research PSEs have been developed for rapid prototyping of ideas and improving productivity. Some of them are generic for a wide scope of applications but without network functionality, e.g. ANSYS [5], MATLAB[6], while others are application–specific or web-enabled such as BioPSE[7], WebPellpack [8], etc. For some sophisticated PSEs, the learning curve is long and sometimes cumbersome, especially for users who are often computer literate but not necessarily trained computer scientists, although such learning might be rewarded in the future. Some users might only need a lightweight process for validating their applications and conducting temporary

experiments for a short period of time, such as "what the results will be if this?" etc.

On the other hand, the communication between human and PSE is still poor. Friendly graphical user interfaces and run-time control still need to be carefully designed. Since the user-GUI interactions is not only just for simple parameter inputs but also for assisting users to perform simulations intuitively. Besides, users also want to be provided with multiple levels of information about his/her simulation for interactive control of the whole process. For example, during a computation intensive process, expanding and simplifying algebraic expressions sometimes causes an explosion in the size and complexity of the underlying data structures used for the computation. The user only knows that the package runs out of memory, or takes extremely long amounts of time but no clues of how to improve that situation [4]. Such low-level performance information should be collected and made available to the user.

The development of next generation PSEs is greatly influenced by rapid advances in the world of networking, dynamic and real time computations, state-of-the-art algorithms and human-machine interactions, especially in computational Grid [9]. Consequently, the need for a Grid-enabled user-friendly PSE that can assist scientists and researchers in performing physical simulations in a convenient and intuitive way is arising. Consequently, we propose the GridPSi system a Grid-enabled PSE for PDE based applications, which is the focus of this paper.

The reminder of this paper is organized as follows. Section 2 describes the overview of the GridPSi system. Section 3 presents the GridPSi client GUI. Section 4 shows the system architecture. Section 5 gives some related work Section 6 makes a conclusion remarks and discusses future work.

2 Overview of the GridPSi System

From an end user's point of view, The GridPSi system consists of a client and a server. This section describes the main features of it.

2.1 Design Principles

There are many well-known design principles [10] to which PSE developers and users agree. Firstly, it must be *problem-oriented*. The PSE should allow researchers to concentrate on their disciplines, without having to become experts in computer science. Secondly, it must be *reliable*. A system that responds with "unable to solve the problem" is much better than one that gives a wrong answer. The third is *persistence and robustness*. It should provide a persistent environment that allows the researcher to resume the solution process at a later time and recover from failure gracefully. There are still other requirements such as high-level abstraction, efficiency, etc. which we will not enumerate here.

By identifying the requirements above, we stick to the following principles when designing our system in an object-oriented manner.

Graphical and Visual. The use of graphical user interfaces and visualization can enhance the usability of the PSE. Friendly GUIs should appear natural to people in the application area, both in user interactions and ways of thinking, and also prevent them from making "errors".

User-centered Interactive. The ability to control the simulation process to as much an extent as possible is desirable for users. The GridPSi framework enables user to interactively prototype, modify parameters and steer in run time without interrupting the simulation.

Open and Flexible. Use component-based software as much as possible. Subsystems should be constructed of interchangeable, plug-and-play components with clear separations. It is flexible for both mature technologies and state-of-the-art components to be incorporated in novel or innovative ways.

2.2 Graphical Simulation Environment

The key feature of GridPSi system is to allow users to conduct PDE-based simulations in a graphical environment intuitively and conveniently. This is accomplished jointly by the PDE SimuGUI and PDE Wizards, which will be described in details in section 3.

2.3 Dynamically Generating Computational Scripts

GridPSi client allow users to dynamically generate declarative computation scripts for their applications at different stages, and submit them to server immediately or at a later time for execution. This is achieved by letting users input a set of information about their PDE-based problems through the client SimuGUI.

2.4 Web Services

Current trends show that most scientific users will wish to run applications via web [11]. For the GridPSi system to be useful to a large community of users and detecting

bugs at an early stage by real users, we provide a client-server based web computational service. Two kinds of service models, batch model and interactive model, respectively, can be provided.

The batch model is a model that a user performs problem specification locally first and then submits a generated computational script to the remote server, and the requested server will send the computed results back to the client in a later time. Interactions between the client GUI and remote server only occurs when a user is ready to do some remote submitting or retrieving jobs. This kind of asynchronous model is currently being used in scientific computation community.

The interactive model is to virtually map the client GUI to a remote machine on which both the GridPSi client and server programs are running. Some technologies available for this approach are VNC (Virtual Network Computing)[12] along with XML (eXtensible Markup Language)[13] and CGI (Common Gateway Interface)[14], etc., which are fairly mature for lightweight network applications.

Undoubtedly, the current Internet is not ideal for this interactive model service when conducting a remote scientific computation. The major bottleneck is the very huge and inefficient network traffics involved since each user-GUI interaction should go through the client-server communication. As a result, the user will suffer unbearable latency. However, the interactive model is promising since it can not only support a real-time simulation which is the long goal of PSE developers, but can also greatly reduce the size of a client program to make it more flexible and portable. We envision a future Internet, as advancements in hardware, software and network bandwidth will make the real-time simulation and visualization become realistic.

Moreover, this model is also studied to collect data necessary for analyzing the characteristics of scientific computation related network traffic. Which will help us provide a submit-once, "nearly interactive"-later model for the current Internet in a later stage

2.5 Grid Resource Accessibility

Computational Grids are becoming increasingly common, promising ultimately to be ubiquitous and thereby change the way global resources are accessed and used [10]. In order to offer users the opportunities to access Grid resources, we also design our system as Grid-enabled.

From a cost-effective point of view, we take an approach to integrate GridPSi client into Unicore client [15]. The Java-based GridPSi can be integrated into Unicore client as a plug-in gracefully via interfaces provided by Unicore client without manipulating the basic software of it. A user can invoke GridPSi client from Unicore client and submit jobs to any Unicore site on which the GirdPSi server is running while Unicore will take care of all the underlying security and other issues.

3 GridPSi Client GUI

The client GUIs resides on GridPSi client running on any java–enabled user workstation or PC, possibly somewhere on the Internet. The client is the sole interface to the end-user who uses it to connect to the GridPSi server. It enables the end-user to

- define PDE problems
- generate computational scripts
- connect to GridPSi server
- save/load files to/from disk
- inspect simulation processes
- submit/retrieve computational scripts/results to/from remote server
- analyze and refine the results

There is no separation between "local" and "remote" versions of the application code for the client. We provide a single client GUI with which PDE-based simulations can be performed.

3.1 Client Main Framework

We implement a main framework in order for further extensibility (e.g. easily adding plug-and-play visualization packages and other simulation toolkits etc. without change of the underlying structure of PDE SimuGUI). This framework is for the user to perform basic file I/O, edit and view operations as well as operations of remote server and visualization. Here we only describe the functionalities of the latter two since the functionalities of the others are nearly the same as all general-purpose GUI-based applications.

The *Remote* menu is used to connect /disconnect and submit/ retrieve computational scripts/results to/from a remote server as well as kill, resume and periodically check the remote process. (the default solve model is implemented on a local server when running SimuGUI). A user can also choose an interactive model by clicking

interact option from the pull-down sub menu when being connected to a remote server.

The *visualization* is separated from the PDE SimuGUI for the purposes that state-of-the-art visualization toolkits can be incorporated in easily in a plug-and-play way, and an interface for performing remote real-time visualization in the future is provided.

3.2 PDE SimuGUI

The PDE SimuGUI is a child framework of the main framework and can be launched from it. It includes a complete graphical user interface, which covers all aspects of the PDE simulation process.

From an end user's point of view, it consists of its own *menu bar*, a t*oolbar* with icon buttons providing quick and easy access to some of the most important functions, a multiple viewed *resource window*, an *active workspace* and a *watch window* as well as a variety of well-designed dialog boxes for user inputs. It guides users through the typical procedures for solving PDEs as below.

- Describing geometric domain
- Applying boundary conditions
- Inputting equation coefficients
- Generating mesh
- Choosing solution method
- Solving problems
- Post-run analysis

Before going further to the procedures listed above, we describe the functionalities of the resource window, active workspace and watch window.

The r*esource window* is a multi-viewed tree-like explore window in which all related resources of the current project are listed for quick view and edit, which is implemented as two view windows labeled as a domain view and a script view, respectively. The domain view window shows the geometric information of main domain (fringe region) and sub-domains (internal no-intercrossed sub-regions) of this project, which are composed of a set of boundary and/or internal border segments represented by the form of edges, arcs and splines in turn. The script view shows the generated computational scripts such as files about domain information, meshing method, and solving methods, etc.

The a*ctive workspace* is for drawing problem geometry (for simple 2D only) and displaying imported geometries (such as from CAD) as well as displaying script files in active.

The w*atch window* is for monitoring the simulation process and reporting errors.

Describing geometric domain. There are two kinds of model for user to describe problem geometry, drawing model and importing from file respectively. The geometric information about a problem domain described above will be automatically saved and displayed in the domain view window. A domain description files will be generated and exported by clicking the *export* sub-menu of geometry.

Applying boundary conditions. By double clicking each boundary segment on the geometry displayed in the active workspace, or the name representing that segment in the domain view window, a Boundary Condition (BC) dialog box will be invoked for users to set BCs, with standard Dirichlet or / and Generalized Neumann modes for scalar u for quickly selection. Likewise, the boundary condition file can also be generated and exported. (This same export function will not be mentioned in the following stages)

Inputting equation coefficients. Equation coefficient dialog boxes can be invoked for inputting necessitated parameters for PDEs involved

Generating mesh. The discretization method is the Finite Element Method (FEM). A user can select either automatically meshing mode (e.g. a fixed triangular mesh will be generated on a two dimensional domain) or choose different meshing mode on individual sub-domain with different shaped elements. The generated mesh can be refined adaptively within a domain or individual sub-domains.

Choosing solution method. Since most iterative methods involve certain parameters, such as maximum iteration steps, tolerance and convergence rate, etc. as well as few algorithms are robust or accurate for the entire range of applicable problems, a user can use this editor to specify his/her requirements correspondingly.

Solving problems. By choosing solve sub-menu, the problem will be solved locally if both client and server are running on the same machine. Solving tasks are actually done by the GridPSi server.

Post-run analysis is for users to analyze, visualize and refine solution results. A user can modify input parameters, methods and plot the results in different

models. The visualization part of post-run analysis can also be activated directly from the main GUI.

In addition, PDE SimuGUI ensures that a successor is executed only if all predecessors have been completed successfully and all necessary data sets are available. (e.g. a meshing option will be disabled until the geometry shape has been constructed.).

3.3 PDE Wizards

A PDE wizard is a multi-paged dialog box which guides the user to specify the basic information about his/her target problems before drawing the problem geometry.

There are eight predefined PDE wizards for the user to choose when he/she begins his/her simulation by clicking *File->New* sub-menu from the PDE SimuGUI. They are listed as follows.

- Generic Scalar Wizard
- Generic System Wizard
- Structural Mechanics Wizard
- Electrostatics Wizard
- Magnetostatics Wizard
- Heat Transfer Wizard
- Diffusion Wizard
- Conductive Media DC Wizard

In each wizard, there are well-designed pages for the user to input or choose basic information for the given problem, such as dimension and domain range on the domain page, nonlinear or linear boundary condition on the BC page, and Young's modulus and Poisson's ratio on the material page, etc. After setting these parameters, a workspace is created for this project, five script files with minimized information about problem domain, boundary conditions, mesh (at this time, only FEM is specified), governing equations and solving method are generated, along with information about project name and time of creation for each files.

3.4 A Simulating Scenario

In GridPSi system, each PDE simulation is treated as an independent *project* for which a permanent workspace will be provided upon creation. The PDE SimuGUI immediately guides a user conduct a simulation in the following scenario: First, after a user chooses *New* from the pull-down sub menus of *File,* a new dialog box is invoked which guides the user to declare the governing PDE to this project by choosing different wizard, as well as the name and location for this project. Then, the specified PDE wizard is invoked correspondingly for the user to perform the process of problem specification. After that, the user uses the PDE SimuGUI to detail the problem specification and his/her requirements. Finally, a server is chosen to solve the problem. (If the server is a remote one, the declarative computational script is submitted after connection.) Upon receiving the results, a post-run analysis will be performed.

4 GridPSi Architecture

This section describes the overall architecture of GridPSi system. The software architecture of a GridPSi system comprising one GridPSi client with a total of three GridPSi servers running on local machine, remote computer and Unicore site respectively, is shown in Figure 1.

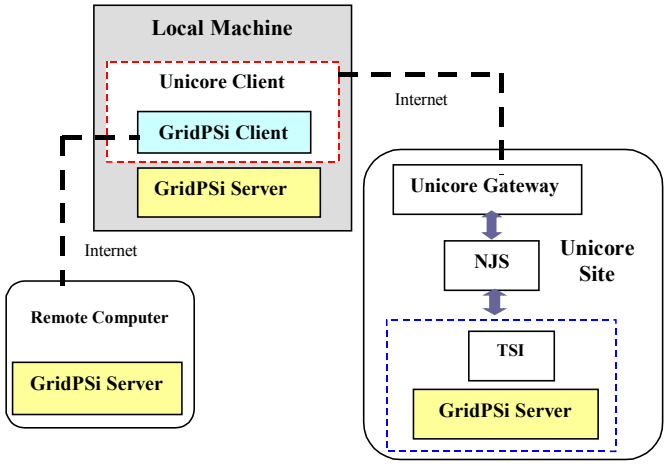

Figure1: GridPSi System Architecture

4.1 GridPSi Client

GridPSi client is implemented as a Java program to eliminate machine dependencies. It consists of the following components (see Figure 2).

Electronic Book provides a mechanism whereby the computational simulations conducted can be recorded and annotated, thereby constituting a record of what has been done. It assists the user to replay his/her experiments, and also serves as a safeguard against errors by tracking and logging system information.

Script Generator helps on generating computational scripts that will be sent to server for execution.

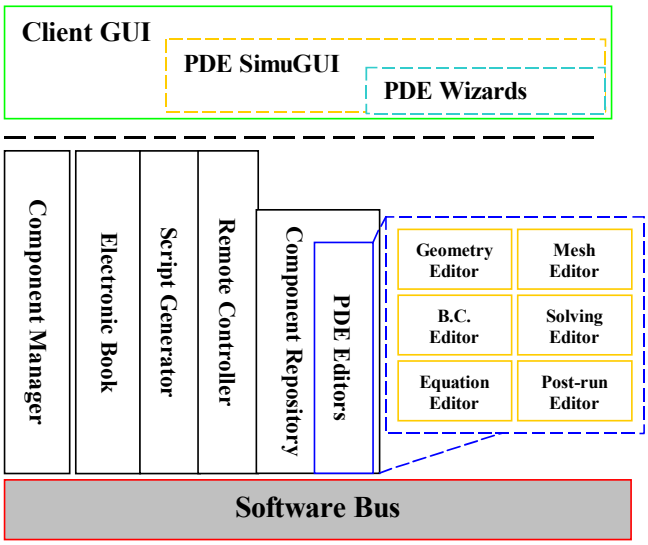

Figure2: GridPSi Client Architecture

Remote Controller manages remote server connection or disconnection and communication.

Component Repository is the database of PDE editors, visualization toolkits and geometric packages, etc. that are to be used to realize the functionalities provided by the client.

Component Manager is responsible for managing necessary components output and input for user and PSE components and implementing components' internal functionalities, as well as providing scheduling mechanisms.

Software Bus is the underlying "glue" to integrate the components of the client.

4.2 GridPSi Server

GridPSi server is also implemented as a Java application. Thus the components built with Java objects in both client and server component repositories can be shared. It is designed to provide service for both local and remote clients. For remote services, it supports multiple network clients by concurrent threads and communicates with instances of GridPSi clients running on client machines via Java Sockets. The server consists of the following components (see Figure 3).

Job Manager serves as a central manager for receiving and dispatching computational jobs, manipulating components for computing the solution, controlling the solving process and reporting the results, etc. when the server detects a remote job, it invokes the session creator which resides in the remote affair handler to create a computational session for it, and reports the completed results to it. It can also cache the results temporarily when necessary.

Script Translator. It is in charge of translating the computational scripts into high level programming language while checking errors before submission.

Solver Pool is a collection of PDE solvers. The solver part is designed to make use of some existing mature and proven numerical packages for the sake of software reuse and computational efficiency, such as PETSC [16], LAPACK [17] and BLAS [18] for examples. Since most native and legacy numerical oriented codes are not implemented in Java language, the wrapper collections are responsible to map the PDE solvers to corresponding standard application programming interfaces.

Result Collector serves as a recording system in collecting and saving computed results.

Software Bus is the underlying "glue" to integrate the components of the server.

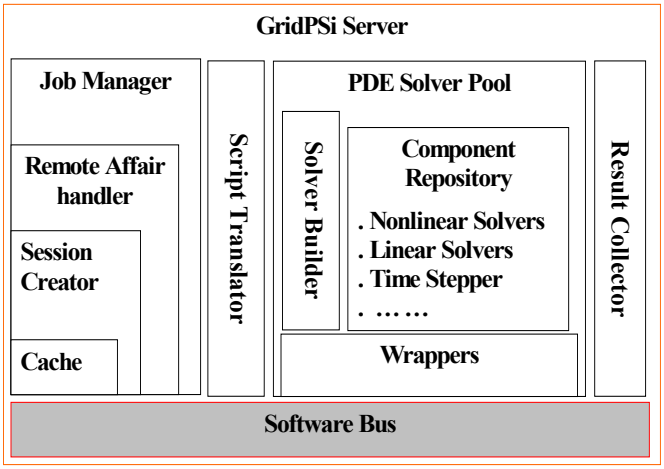

Figure3: GridPSi Server Architecture

5 Related Work

As mentioned earlier, for the sake of software reuse and not "reinventing the wheel", numerical libraries such as PETSC, LAPCK and BLAS are used to complement the numerical part of our system; third party software packages are plugged into our system for mesh generation and visualization; and Unicore client plug-in interface is utilized to feature our system with the Grid resource accessibility.

No PDE-based PSE so far has targeted the specification of PDE problems in a way that we have presented, although some of them also provide GUIs as well. Since

the problem specification process of PDE-based simulations involves a lot of delicate and error prone human-GUI interactions, even a tiny mistake can lead to a false computational results. Traditional GUI-based PSEs often lack the explicit methods to avoid occurrence of errors, and often provide general purposed dialog boxes with superfluous information that might confuse a user when the user is asked to input parameters. Take Matlab PDE Toolbox for an example, even if one boundary condition is nonlinear of a given geometric shape, the user has to keep in mind that he/she must choose a nonlinear solver in the later stage, otherwise, the system will produce a confusing result, a result solved by a default linear solver (it will be better if no result is given at all). However, with our system, a nonlinear solver is guaranteed after the user has specified the nonlinear boundary condition in the PDE wizard. If the user chose a linear solver by mistake or on purpose, a caution message will be generated automatically when the user chooses confirm button. Besides, after specifying a simulation type by choosing a corresponding PDE wizard, only necessary dialog boxes with reduced parameter options are to be invoked for the user's inputs.

Still, the computational script based remote service provided by our system is also unique. Most current existing PSE providers let users access their remote servers via a Java Applet based web portals to describe and define their PDE problems, such as WebPellpack[4]. Users need to log onto the remote server and describe their problem page by page and submit it until completion. Such kind of text-based description is often inadequate and incapable for dealing with complicated applications such as PDE-based simulations. However, in our case, the visual and graphical problem specification process is separated from the server (at the cost of a somewhat bulky client indeed), the server only needs to handle the solving process upon receiving a declarative computational script.

6 Conclusion and Future Work

In this paper we have described the functionality and architecture of the GridPSi system, which aims to provide a Grid-enabled, user-friendly environment for computational researchers to conduct PDE-based simulations intuitively and conveniently, hence will ease their prototyping processes and increase productivity. It also enables users to easily run powerful applications on the Grid, taking advantages of global resources such as supercomputers and so forth without having to care about the underlying environments and implementation details.

The GridPSi system is still being under design and implementation. The functions of individual component need to be carefully tested. The implementing details of interfaces still need to be considered. For future work, we plan to make use of VTK (Visualization ToolKit)[19] and caching technologies in our project to implement an interactive environment for visualizing and interacting with numerical simulations in real time.

References

[1] E. Houstis and J.R. Rice, Future Problem-Solving Environments for Computational Science, Computational Science, Mathematics and Software, pp: 93 - 114, 2002, ISBN: 1-55753-250-8.

[2] Y. Umetani, M. Tsuji, et al, "DEQSOL: A Numerical Simulation Language for Vector/Parallel Processors", Proc. IFIP WG2.5 Working Conference on PSE for Scientific Computing, North Holland, pp. 147-162, 1985.

[3] R.F. Boisvert, E.N. Houstis and J.R. Rice, A system for performance evaluation of partial differential equations software, IEEE Transactions on Software Engineering SE-5 (1979), pp.418-425.

[4] Report of workshop on problem solving environments and scientific IDEs for knowledge, information, and computing (SIDEKIC98) Santa Fe, NM, USA 1998

[5] http://www.ansys.com.

[6] http://www.mathworks.com.

[7] http://software.sci.utah.edu/scirun-biopse_1_8.html.

[8] E. N. Houstis, et al, PELLPACK: A Problem-Solving Environment for PDE-Based Applications on Multicomputer Platforms. ACM Trans. Math. Softw., Vol. 24, No. 1, pp. 30--73, March 1998.

[9] Ian Foster, The Grid: Computing Without Bounds, Scientific American Digital, April 2003.

[10] G. von Laszewski, I. Foster, et al, Designing Grid-based Problem Solving Environments and Portals, 34th Hawaii International Conference on System Science, 2001.

[11] Toyotaro Suzumura, Hidemoto Nakada, Satoshi Matsuoka, Henri Casanova, GridSpeed: A Web-based Grid Portal Generation Server, proceedings in seventh international conference on high performance computing and grid in Asia Pacific Region, Tokyo, 2004.

[12] Tristan Richardson et al, Virtual Network Computing, IEEE Internet Computing, Volume 2, Number1, January-February 1998.

[13] http://www.xml.org.

[14] P. Thiemann, "WASH/CGI: Server-side web scripting with sessions and typed, compositional forms," in Proc. 4th International Symposium on Practical Aspects of Declarative Languages, PADL '02, January 2002.

[15] http://www.unicore.org.

[16] http://www-unix.mcs.anl.gov/petsc.

[17] http://www.netlib.org/lapack.

[18] http://www.netlib.org/blas.

[19] http://public.kitware.com/VTK