Scientific Computing via the World Wide Web: The Net //ELLPACK PSE Server

Shahani Markus, Sanjiva Weerawarana, Elias N. Houstis and John R. Rice
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907-1396, USA.

{markus,saw,enh,jrr}@cs.purdue.edu

Abstract

The World Wide Web is now the *defacto* environment for providing (electronic) information to the worldwide user community. With the availability of programming languages like Java, Web browsers now can handle documents with programs embedded in them. This new functionality presents opportunities for *scientific computing* on the World Wide Web while its realization requires addressing several new research issues including *user interface and protocol design, legacy software encapsulation, security, software delivery in directly usable form, and networked computational servers.* In this paper we address these issues in the context of PDE (Partial Differential Equation) network computing supported by the well known problem solving environment (PSE) //ELLPACK and its networked, Web accessible, counterpart which is referred throughout as the Net //ELLPACK PSE. Using Net //ELLPACK one can solve complex PDE problems using any Web browser that supports Java applets. The design of Net //ELLPACK includes a Java applet that serves as the graphical user interface, a stateful text-based protocol, and Net //ELLPACK server(s) running on some machine(s) anywhere on the network. In addition to its native protocol, the Net //ELLPACK servers also support the Web HyperText Transfer Protocol (HTTP). This approach allows the reuse of existing public domain or proprietary applets to visualize or post-process the computation results. We present and justify the overall design of the system and present some data on solving a partial differential equation problem via the Web to illustrate the effectiveness of the Net //ELLPACK architecture.

1.0 Introduction

High performance computing and the Internet make it possible to greatly increase the problem solving power that a scientist or engineer has at his finger tips. Examples of this are reusable software components, monolithic problem-solving systems, software testing systems, and widely distributed compute servers. This power is not easily accessible now to the application scientist or engineer because using these resources requires skills and knowledge that most do not have.

Enabled by advances in hardware, networking infrastructure and algorithms, highly compute intensive problems in many areas can now be successfully attacked using networked scientific computing. In the network-based paradigm, vital pieces of software and information used by a computing process are spread across the network, and are identified and linked together only at run time. This paradigm is in contrast to the current software usage model where one purchases a copy (or copies) of a general-purpose, monolithic software package for use on local hosts, possibly distributed on a collection of local hosts. With network-accessible software repositories and networked computing, the view of software changes from a product to a service. A network-accessible repository provides access to up-to-date copies of software components on an as-needed basis, so called “disposable software.” With networked computing, the software developer provides a computing service to interested parties over the network. The raw computing

power to run this service may be purchased from the software service provider, provided by the end user, or even purchased from a third party computing service provider. The information needed to use this service might be available only from disparate sources.

The disposable software and networked computing models do not apply to all computing services; basic operating system and network access software, as well as low-level mathematical routines that are tuned for the particular machine architecture, will be permanently resident on the user machine. One advantage of the networked computing model is that as software is improved upon by the software provider, there is no need to release new versions and upgrades. The user simply sees an improved service. The analogy could be to the phone system - changes in the software of the local switch are completely transparent to the user, save for the availability of additional or enhanced functionality. Similarly, the service provider can upgrade the hardware without affecting the user. We envision that the network-based paradigm of software usage will eventually become fully automated and effectively transparent to the user. It is the aim of this research effort to demonstrate this in the context of PDE computing and problem solving environments (PSEs) utilizing the existing Web technologies and infrastructure.

Realizing Web based computing involves several research issues including user interface and protocol design, legacy software encapsulation, security, software delivery in directly usable form, and network computational servers. In this paper we address these issues in the context of the well known PSE //ELLPACK [1] and its Web enabled counterpart referred throughout as Net //ELLPACK.

A PSE is a computer system that provides all the computational facilities necessary to solve a target class of problems [2]. It uses the language of the target class of problems and provides a “natural” user interface, so users can run them without specialized knowledge of the underlying computer hardware or software. A PSE exploits modern technologies such as interactive color graphics, powerful processors, and networks of specialized services. The main design objective in Net //ELLPACK is to provide the //ELLPACK interface to remote //ELLPACK users in an effective, secure and efficient manner.

Once the Net //ELLPACK user has selected a solution path through interactions with //ELLPACK, one or more library software modules need to be downloaded from a repository and used locally, or the problem is sent to a computational server with an implementation of that algorithm. In both cases, there is a high-level interface between the PSE user interface and the library routines it invokes. In the case of downloaded software modules, there is also a low-level interface between the library routines and the low-level machine-dependent math and (in the case of a parallel machine) communications facilities that are resident in the user’s local computing environment. In order for the network-based computing paradigm to succeed, both low-level and high-level library interfaces must be standardized. Standards permit the effort of developing and maintaining bodies of mathematical software to be leveraged over as many different computer systems as possible. This issue is addressed in the context of //ELLPACK interfaces [1].

One of the biggest obstacles that applications programmers face in using library routines downloaded from software repositories is the correct installation of these routines. These routines typically need to be compiled and linked with the user’s program and with local libraries before they can be executed. Users often spend hours to days trying to run down unresolved references or name conflicts that occur during the compile and link process.

When software is to be used only occasionally, or when a software library needs the performance or specialized facilities provided by a specific platform that is not available to the user locally, it is advantageous for the user to send his problem and data to a remote computational server that can perform the computations and return the result over the network. Remote computational servers are also useful for software that is difficult to install or is frequently updated, or that authors do not wish to distribute for other reasons. Users may also find paying for time on remote computational servers to be more economical than purchasing one or more high performance platforms themselves.

In the work reported here, we concentrate on the compute server approach: once the solution algorithm is determined, the solution computation is performed on a remote server. The user interaction with the compute server is entirely via existing Web based technologies. In an on-going collaboration with the National Institute of Standards and Technology and the University of Tennessee-Knoxville, the problem of transporting code from software repositories on to the user’s machine in directly usable form is investigated.

There are many other projects that attempt to address the issues related to scientific computing under the network-based or network-centric paradigm. The NEOS optimization system [3] provides electronic mail, Web or customized X-based tools as interfaces for its NEOS server, providing computational resources and access to state-of-the-art optimization software. Utilization of the graphical user interface for NEOS requires the explicit downloading and installation of the submission tool. The computational results are available via the Web or electronic mail. There are many sites that provide access via the Web to servers running specialized mathematical software systems such as Mathematica [4], Maxima [5] and Maple [6]. However, at these sites the user can interact only in a non-session oriented, request-response manner, making it difficult to accomplish any complex problem solving tasks. The Web //ELLPACK system [7] provides a session-oriented interface to a remote //ELLPACK computational server. However the performance of the remote user interface over a wide area network is poor. In the Net //ELLPACK system, we attempt to address all these problems and provide viable solutions utilizing currently available Web technologies and infrastructure.

This paper is organized in 6 sections. The software architecture of //ELLPACK is presented in Section 2. In section 3 we present four different scenarios for building Web-based PSEs by Web enabling legacy components of existing PDE software. Section 4 presents an overview of the Web technologies needed to design and implement Web-based PSEs, particularly Net //ELLPACK. Section 5 defines the Web computing research issues associated with the realization of Web based PSEs. The software architecture and components of Net //ELLPACK are presented in Section 6. Its accessibility together with some examples is outlined in Section 7. The justification of network computing is summarized in Section 8.

2.0 //ELLPACK: A Distributed PDE Problem Solving Environment

//ELLPACK is a problem solving environment for solving PDE problems on high performance computing platforms as well as a development environment for building new PDE solvers or PDE solver components. It currently supports the solution of steady-state and time dependent *field and fluid mechanics* 2-D and 3-D PDE problems.

The main design objective of //ELLPACK is to create an intelligent software environment where both sequential and parallel PDE solvers can be implemented in a reasonable time. //ELLPACK presents application users with a high level environment to abstractly specify PDE problems and build solvers for them using intrinsic solver components. Knowledgeable users apply //ELLPACK's solver development facilities to build new solvers which are then available as intrinsic solver components for application users.

The software architecture of the system is depicted in Figure 1. It consists of four layers of software. The top layer is a collection of graphical tools supporting the PDE problem and solution specification and all the required pre- and post- processing phases. Each tool consists of a GUI editor and some are supported by appropriate libraries to support their computational objective. The outcome of the usage of these tools is saved in the //ELLPACK main session editor in some predefined high level language. This language can be used to specify both the PDE problem and algorithm including the pre- and post- processing computations required by the PDE solvers. The third layer consists of the scripting code required to *make* and *execute* the specified PDE computation and define the *execution platform* which can be any machine or a cluster of workstations in a LAN facility. An instance of the user interface to this software layer, called execution tool, is depicted in Figure 4. This tool supports the execution of //ELLPACK programs on MPI and PVM virtual parallel machines running on clusters of workstations, nCUBE II, and Paragon.

The fourth layer in Figure 1 supports well-defined data structures, functions, and file formats on which the //ELLPACK library has been implemented. This layer allows users to easily integrate their own specialized solvers and solver parts into the environment and "foreign" PDE solvers. The new code must be modified to access the required data from //ELLPACK data structures. Its name and any user-specifiable parameters are then placed in the //ELLPACK software repository, and the modified code is installed in the //ELLPACK library system. In this way, "software parts" such as mesh generators, discretizers, and solvers can be added to the system with minimal effort. Some software parts with complex input requirements take significantly more effort to integrate. For example, some are defined through FORTRAN functions. //ELLPACK can generate these functions through the graphical interface, using its symbolic manipulation capabilities which are available through the equation specification tool.

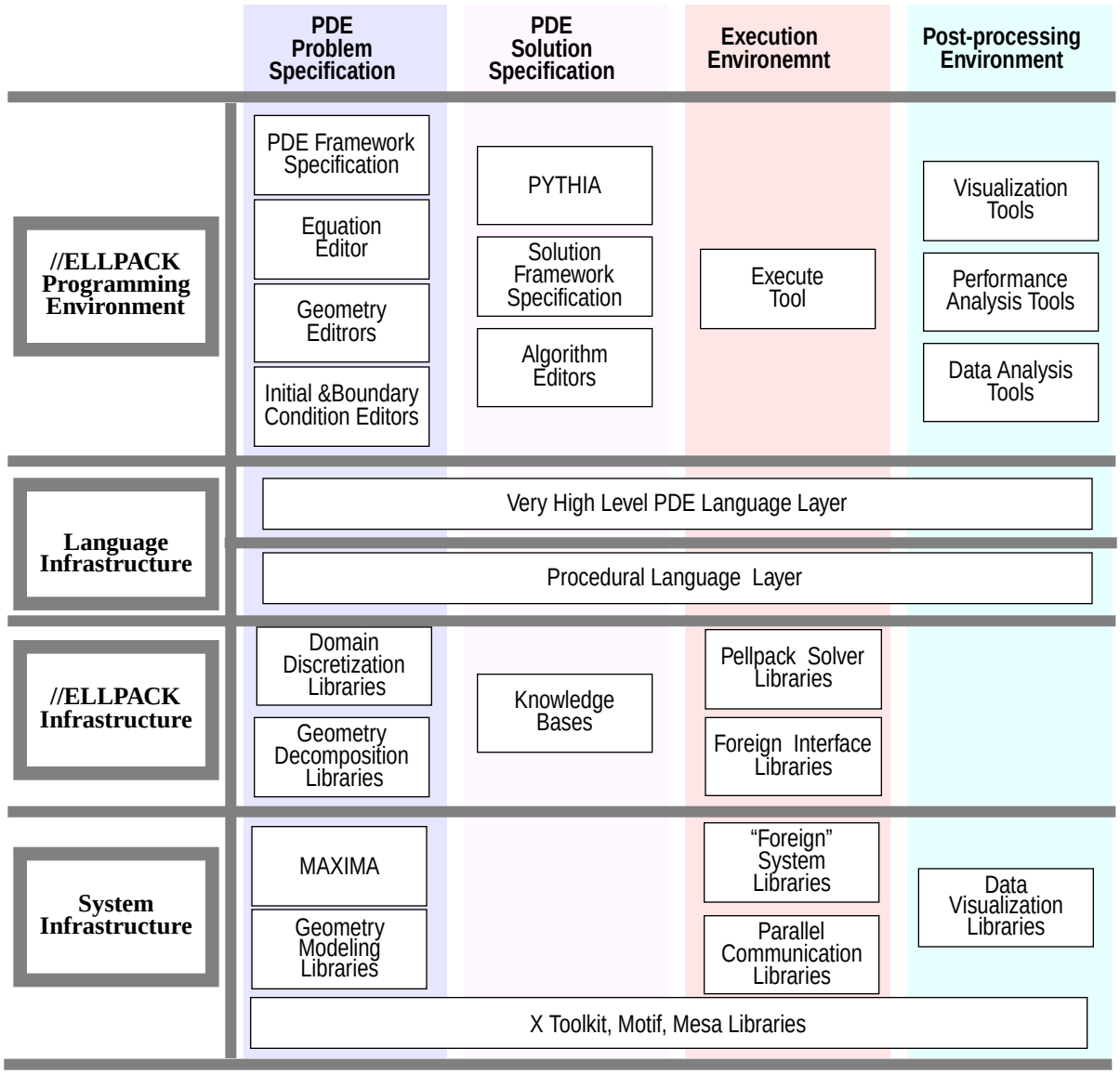

FIGURE 1. The software architecture of //ELLPACK.

The libraries of solvers available in //ELLPACK include both finite difference and finite element solution schemes. In addition to the solver components that we have developed locally, several publicly available PDE solvers have been integrated into //ELLPACK. Some of the "foreign" systems we have integrated, include VECFEM [8], FIDISOL [9], PDECOL [10], PARC [11]. For solving the PDE discrete system of algebraic equations, //ELLPACK includes several of the widely used linear system solvers including ITPACK, LINPACK, SPARSKIT and NSPCG. //ELLPACK supports three different parallel domain decomposition methodologies [12]. One of the domain decomposition approaches, including a tool for controlling it, allows the user to reuse the sequential discretization solvers and to compute in parallel the PDE discretization systems. For this we have parallelized the ITPACK library and implemented it using VERTEX, MPI, PVM, PICL and PARMACS communciation libraries [12]. These parallel libraries have been implemented for distributed memory parallel machines as well as networks of workstations. The code has been ported to and tested on an nCUBE/2, an Intel iPSC/860, an Intel Paragon and on Ethernet and ATM based workstation networks. In addition we have integrated the MGGXX [13] library running on the nCUBE II and the integration of several other parallel linear solvers is underway.

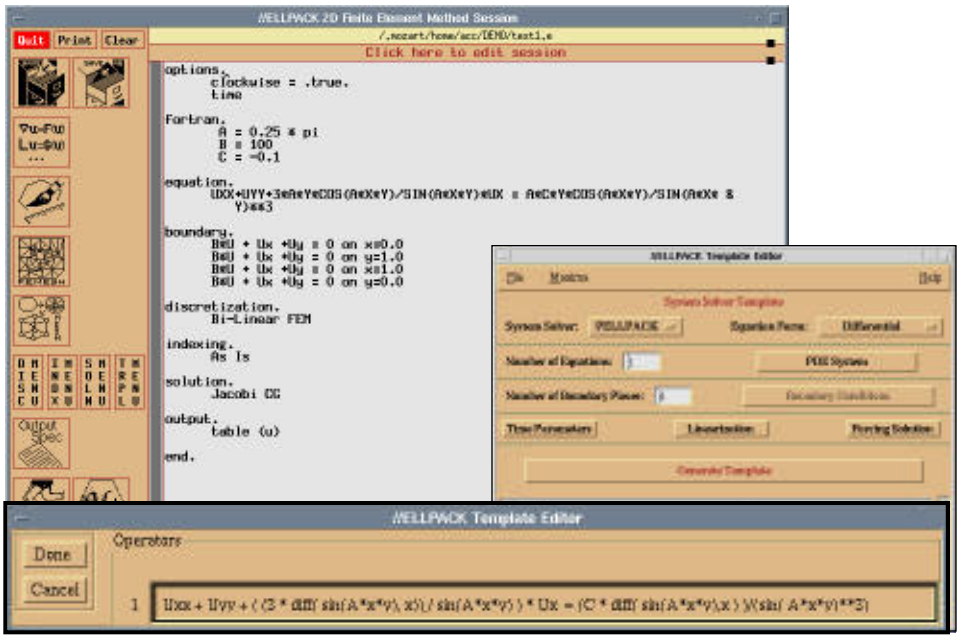

FIGURE 2. The //ELLPACK session editor and the equation specification editor..

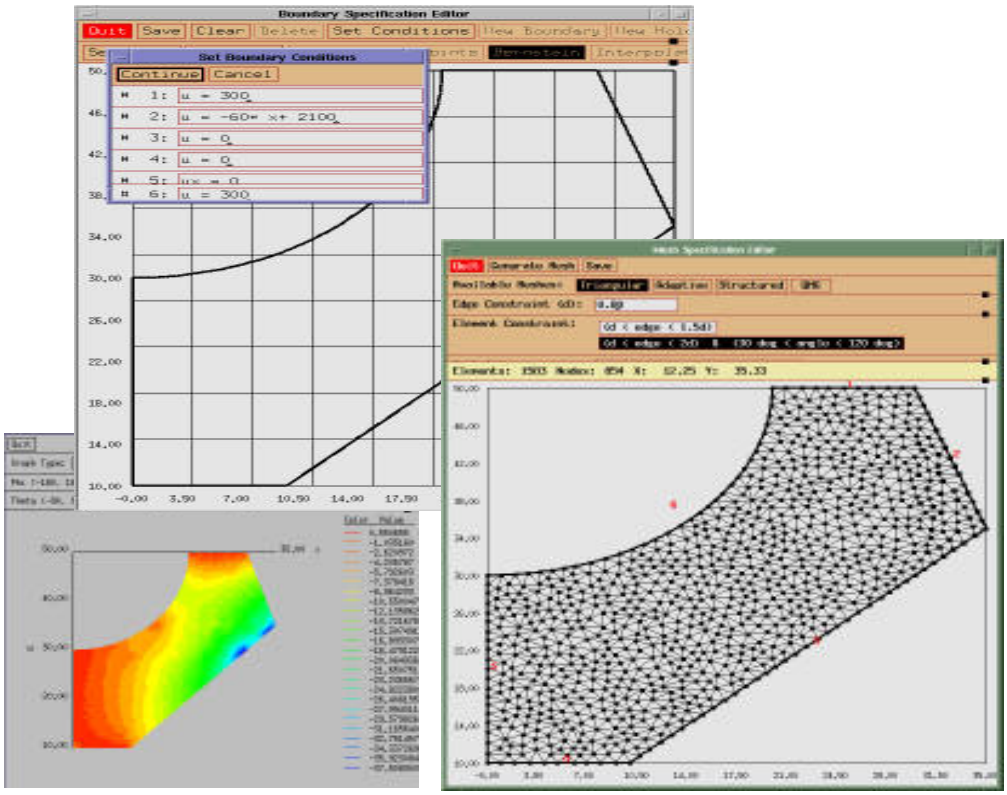

FIGURE 3. The //ELLPACK boundary specification editor, mesh generator and solution visualizer.

FIGURE 4. The //ELLPACK Execute Tool

3.0 Web-based Problem Solving Environments

While many of the issues and solutions discussed above are applicable to general Web computing systems, there are some that need to be addressed in particular towards the realization of scientific PSEs on the Web. We consider PSEs with the following structure: A GUI for pre-processing, a run-time system consisting of a driver program which invokes various libraries for the scientific problem "solution" and a GUI for post-processing. Since both the pre- and post-processing GUIs are similar in structure, we consider those to be all part of one GUI environment (Figure 5). The driver program consists of a collection of calls to library routines with data passed between the routines via the *software bus* in the driver. For the simplest case, the software bus is the global and local variables (memories) of the driver program. Based on this model for PSEs, we identify four different scenarios for building Web-based PSEs by Web enabling legacy components of existing PSEs.

The first approach is to take the PSE and make it available over the Web in its entirety (Figure 6). Web //ELLPACK [7] is an instance of this scenario. The Web //ELLPACK service allows remote users to access and use the //ELLPACK PSE in a safe and secure manner. The re-engineering required was mainly in building a suitable security mechanism to implement a user account based system for remote access. Feedback from this project has led us to conclude that while this approach is feasible on a high speed local area network, it is not practically viable due to very slow response speeds of graphical software operating over the Internet.

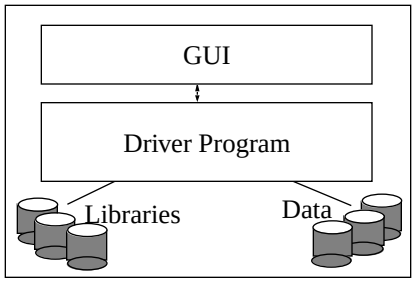

FIGURE 5. Model for Scientific PSEs

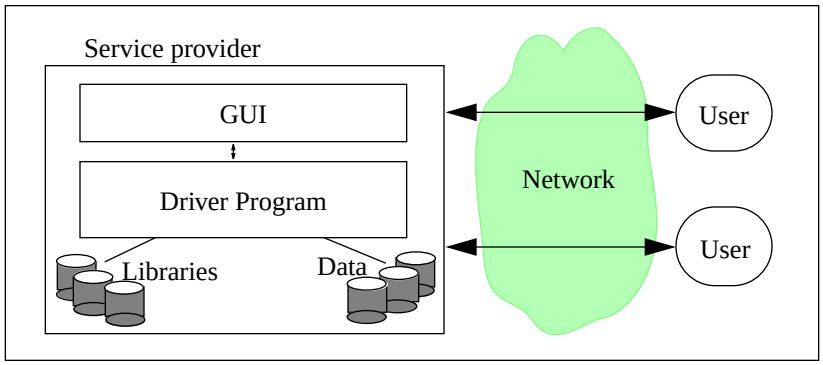

FIGURE 6. Distributed users accessing a central service

The second approach is to use a networked software bus to distribute the libraries to multiple service provider sites. We call such libraries *virtual libraries*. This scenario is illustrated in Figure 7. A networked software bus could be implemented by using either standard network communication technologies such as RPC [14] or CORBA [15] or by incorporating a proprietary or custom built network communication system such as DCOM [16] or the PSEBus software bus [17]. The realization of virtual libraries would also require the use of intelligent software agents. They would be aware of the functionalities, characteristics and interfaces of the remote libraries and would provide assistance in accessing and linking these virtual library modules. High performance computation servers would be utilized to remotely execute specialized library modules. This approach can also encompass the goal of software delivery in directly usable form. The directly usable software in this case would be virtual library modules that may be either statically or dynamically linked to a program on the remote user's machine. These modules may be dowloaded onto the user's machine during compilation or at runtime. The NetSolve project [18] which is being jointly developed by University of Tennessee and Oak Ridge National Laboratory is an instance of this scenario.

The third approach is to distribute components of the PSE GUI to multiple service provider sites. GUI components may include PSE tools such as domain builders, mesh generators and scientific data visualizers. This approach may involve the distribution of parts of the driver program to these sites as well. The remote user would access the distributed GUIs for problem specification and/or post-processing. A driver program at the central service site would coordinate and gather the problem specification data, solve the problem, and provide access for distributed post-processing of the solution data. The problem solving computations may be done on high performance parallel

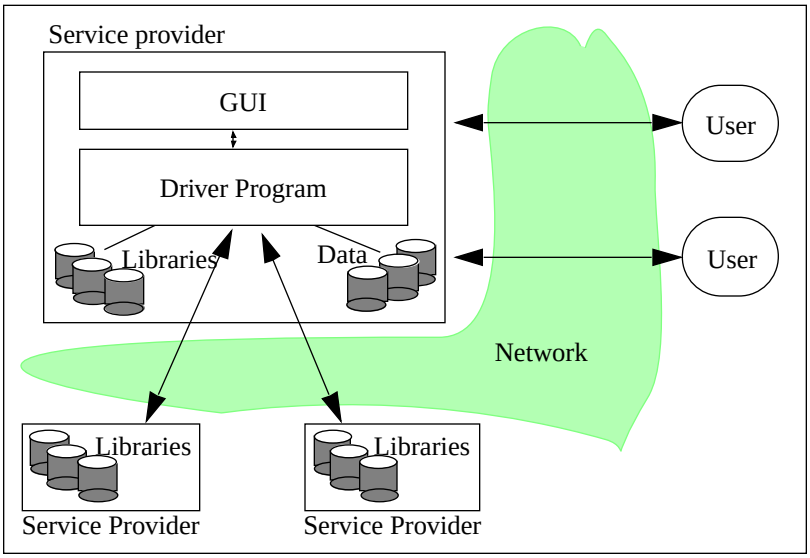

FIGURE 7. Distributed users accessing a central service which uses virtual libraries.

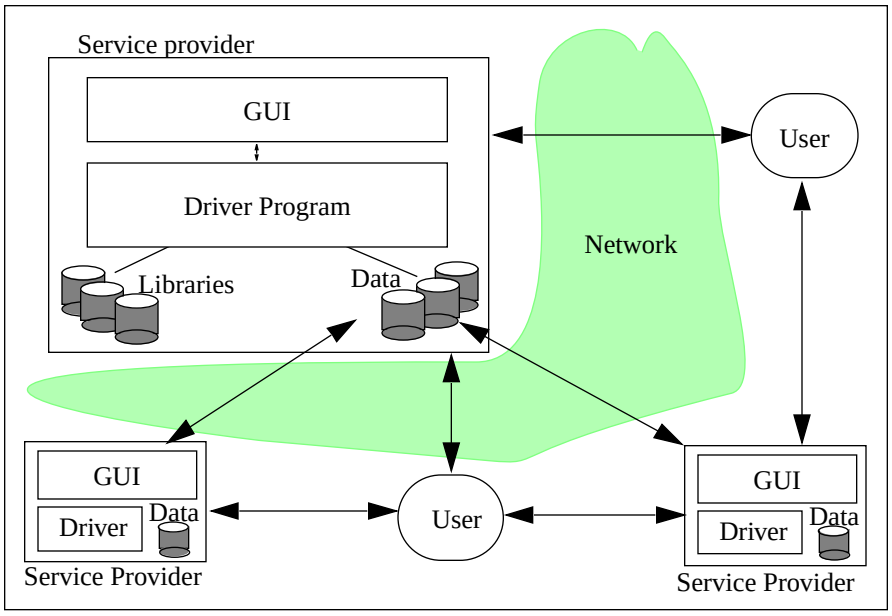

FIGURE 8. Distributed users accessing distributed PSE GUI component services coordinated by a central service.

machines or may be distributed over a virtual network of cooperating compute servers. This scenario is illustrated in Figure 8. This approach would also require the use of intelligent software agents. They would assist the user in selecting and locating the appropriate GUI component sites for problem specification and post-processing as well as negotiate communication protocols between components. They would also intelligently provide any data filtering necessitated by various legacy components adhering to different formatting standards and would transparently move the pertinent data to the target compute servers. The Net //ELLPACK system described in this paper is an instance of this scenario.

The final approach is a fully distributed, fully collaborative, multi-user, virtual library based, networked, Web accessible intelligent PSE environment where one can freely combine components from participating service providers to analyze and solve the problem at hand. On-going work at Purdue and elsewhere is moving in this direction.

4.0 Web Infrastructure

The World Wide Web is a distributed information system build on a hypertext model where pieces of information are linked to one another in a natural way. Information is provided to the Web by Web servers which communicate with clients seeking information (browsers) using the Hyper Text Transfer Protocol (HTTP). Information (documents) on the Web is identified / addressed by Uniform Resource Locators (URLs). The Web also now subsumes other information sources such as files made available to the network via the File Transfer Protocol and the Usenet news system.

The World Wide Web has now firmly established itself as the defacto standard for publishing (electronic) information to the global community. Web browsers, the software that provide point-and-click type graphical interfaces for navigating the Web, have evolved recently to go beyond being just Web browsers: they now seamlessly integrate other networked communication / interaction protocols and media such as file transfer, Usenet news and electronic mail. In addition to static content, the Web also supports dynamically generated content; i.e., information in a “document” that is generated just when someone requests such a virtual document. In a separate dimension, the evolution of executable content has brought forth a new sense of dynamism to the Web: documents can now have programs embedded in them. Executable content allows a document author to embed a program written in some portable language understood or supported by the client browser in the document itself. When a client visits the document, the program is transparently downloaded into the client’s browser and executed by the browser. Obviously security considerations are the primary issue here: blindly downloading and running others’ programs is clearly a high-risk activity. How-

ever, barring security implications, such programs allow users to perform useful computations on the Web by downloading pre-compiled programs into their Web browser and running them with appropriate input.

In 1995 Sun Microsystems announced the release of the Java programming language [19], a language designed specifically for writing executable content so that someone can safely download it and execute it. Java has since gained wide acceptance and is now available on most hardware and software platforms. While Java does not solve all the security problems, it provides a solid foundation upon which applications and client browsers can build the level of security they desire.

4.1 HTTP: The Web Communication Protocol

The primary protocol that drives the Web, HTTP, is a simple, text-based, request-response type stateless protocol. A browser requests a document from an HTTP server by giving a URL and the server responds to that request by providing the document. The document is typed using the Multipurpose Internet Mail Extensions (MIME) type system and browsers use this type to invoke the appropriate handler for the information received from the server. The request made by a browser may result in a program being executed on the server to generate the information requested - this allows a degree of dynamicism where the requested information may be generated dynamically. Programs executed by the server interact with the server via the Common Gateway Interface (CGI), a standard which specifies how the server provides input to it and how the program should provide output to the server.

HTTP has a flexible challenge-response type security model to control access to documents on the Web. When an access controlled document is requested, the server refuses the document and requests that the browser provide a response to a given challenge and resubmit the query. Multiple security schemes can be provided with the core requirement being that both the client and the server must support them. The protocol itself defines a security scheme called “Basic” where the server issues a challenge to which the client must respond by providing a user name and a password as the authorization for a document. Some servers also support other forms of access control, such as (dis)allowing requests from certain domain names / IP addresses, but these are server specific and not part of the HTTP protocol.

4.2 Executable Content

While the ability to have the server run programs in response to a client query is a form of Web computing, the primary enabling technology for truly interactive Web computing is executable content. The idea behind executable content is simple: embed a program within a document and when a client browser visits that page the program will be run automatically by the browser on the client’s machine.

Many issues must however be resolved to fully realize executable content in an efficient, safe and secure manner. Given that browsers may be running on a variety of hardware / software platforms, the programs must be written in an extremely portable language in order for them to be useful to a wide audience. Interpreting high level languages is a common solution to portability questions, but efficiency is a primary concern too: if the language is not run very efficiently, then the usefulness of the language to write non-trivial applications will be limited. Security is a crucial issue: if a program is run automatically by just visiting some Web page, how can the client ensure the program will not perform any malicious activities? Several languages / environments for executable content have been developed recently [19][20][21][22][23], but the Java programming language and environment stands out both in terms of the language itself and in terms of the wide support it has received in a very short time.

4.3 The Java Programming Language and Environment

Java is a new programming language developed by Sun Microsystems, Inc. and was first released to the public in the summer of 1995. Sun describes Java as a simple, object-oriented, distributed, interpreted, robust, safe, architecture neutral, portable, high performance, multithreaded, dynamic language. Syntactically it is very similar to C++ and hence immediately familiar to many users. Java programs are executed by compiling the source into bytecodes which are then interpreted by the Java virtual machine. Since Java bytecodes are defined in a portable, architecture-neutral manner, any Java program is immediately and automatically ported to any platform to which the Java virtual machine

has been ported. The bytecodes and the virtual machine provide a fairly efficient execution environment for Java programs. The language includes support for distributed computing and for graphics and graphical user interfaces. Object-oriented and multithreadedness as well as the ability to dynamically extend the run-time system makes Java powerful environment for Web computing.

Most Web browsers now have the Java virtual machine embedded in them. Barring implementation bugs, the embedded Java virtual machine enforces a set of strict security rules which prevent a downloaded program (applet) from adversely affecting the client's machine or from communicating with Internet hosts randomly. Some of the security measures enforced include disallowing file I/O on the client machine and disallowing network access to any host other than the host the applet was downloaded from. Downloaded programs (bytecode) are also checked for correctness to ensure that they follow the language rules before they are executed by the virtual machine.

5.0 Web Computing: Issues and Solutions

As described in the above section, the Web infrastructure and the advent of executable content provide a mechanism to develop Web computing systems. We have chosen Java as the means of realizing executable content, mainly due to its current popularity within the Web community. In this section we consider the research issues in designing new software and re-engineering legacy software to make them available as Web computing systems. While some of the solutions to these issues are dependent on the features and restrictions of the Java environment, most are general. Since security is an underlying concern in all of the other issues, we do not treat it separately but treat it along with the others. Also, while some of the issues and solutions are targetted towards scientific problem solving in particular, others are of general scope.

5.1 Architecture of Java-based Web Computing Systems

Java and Java-enabled Web browsers provide a mechanism to write programs that users anywhere in the world could execute safely using any hardware platform. However, what is not clear is how to architecture new software and how to re-engineer existing software to make them available in this form. The Java-based Web computing environment is basically a client-server world where the client is a Java applet that is downloaded into a Java-enabled Web browser and run within it. The Java security framework constrains the client (applet) to interact only with the Web server host from which it was downloaded.

The simplest design for a Web computing system is to re-write the entire system as an applet. However, there are many reasons for this being a bad idea. The applets cannot perform file I/O on the local host. Hence, while the applet can run and do many computations, it cannot save any state or load any state. Clearly for non-trivial applications this is an unrealistic constraint. In addition to applets being I/O constrained, if an entire system were to be rewritten as an applet then the latency in downloading the system into the browser would be prohibitive. However, browser caching will all but eliminate this problem after the first time, so the initial latency may be somewhat acceptable. On the other hand, for all the advantages of Java and Java-like languages, there is a significant performance hit in even the most optimized Java code; typically a factor of at least 2 when compared to compiled imperative languages such as C and FORTRAN. In non-trivial applications a factor of 2 performance loss in the compute intensive parts of the application would be unacceptable. Hence, it is clear that one generally should not have compute-intensive components of a system be downloaded and run directly within the browser; some separation of user interface from the compute-intensive portions of an application is needed. Web computing systems must therefore follow a client-server architecture with an appropriate protocol for communication between them.

5.2 Client-Server Communication Protocol

There are both advantages and disadvantages in choosing the Web protocol, HTTP, as the communication protocol for a client-server Web computing system. A significant advantage is in the ease of use of the system across a Web traffic compliant computer network firewall. In the case of a customized communication protocol, additional software would have to be installed on the gateway to facilitate communication across the firewall or a suitable method of bundling the custom protocol within the HTTP protocol would have to be devised. On the other hand, since the client

(applet) cannot maintain any state, all application state must be maintained by the server. Because HTTP is a stateless protocol, if one were to use the HTTP server and HTTP for communication then state would have to be encapsulated into the messages themselves. Also, since HTTP is a request-response model, each communication between the client and server would effectively be a different connection, which also requires that significant state be transferred in each message. Messages would be acted upon in this scheme by the HTTP server invoking programs on the server machine. While this approach is a feasible one to implement a Web computing system, it is clear that all client-server communication will need to be shoehorned to fit the HTTP mold. The security model in HTTP also does not fit very well with the needs of applications. The alternative is to use a custom server and a private protocol between the client and the server along with a custom security model.

When designing a private protocol for communication between the user interface and the remote computational components, many issues need to be carefully considered. For instance, both stateless and stateful protocols have their merits and demerits and must be evaluated on a case-by-case basis based on the frequency and granularity of communication between the two entities as well as other factors. For the Net //ELLPACK case, assuming that all compute intensive components are found on the server and that the entire user interface is the client, the communication between client and server would occur when defining a problem, when defining a solution scheme and when analyzing a computed solution. Some of the problem specification components involve heavy computing, and hence would require server communication. Some problem components are small (a short string) while some are quite large (several hundred kilobytes). Also, problem components may depend on one another. Based on these observations, we chose a stateful protocol for the Net //ELLPACK system.

Another consideration is whether to use a text-based protocol or a binary protocol. Text protocols (for example, HTTP) have several benefits including human readability, hardware platform independence and ease of testing (using telnet). The cons for text protocols include cost of parsing (messages must be parsed), inaccuracy of representing numerical quantities (printing numbers to ASCII and reading them back may change some bits in the number) and inefficiency of space usage for numerical data (for better number representation numbers must be printed using a large number of digits (bytes)). The pros for binary protocols include fast parsing and efficient and accurate number representations while cons include debugging difficulties and portability problems (differences in binary representations between heterogenous hardware). This design choice must also be evaluated on a case-by-case basis. For the Net //ELLPACK case we chose a text-based protocol as we felt that the benefits outweighed the costs. The biggest negative (cost and inaccuracy of representing numbers) was not a real concern since many of the existing pieces of software we wanted to reuse required a text representation of the data.

Protocol security is a very important issue: the communication protocol must allow applications to provide the desired level of security to users. Since in our architecture state is maintained at the server which may be simultaneously accessed by different users throughout the world, it is important that users be separated from one another. The server uses an account model with a login identifier and a password to separate one user from another and also to provide security. We use an Internet-unique string (such as one's e-mail address) as a login identification for each user. The user is allowed to specify a password when the account is first created. Later accesses require this password. Authentication is once per session; i.e., after a user authenticates himself or herself, the authentication is valid for the duration of the session. A session is defined as the duration of a computation and may be named so that it can be returned to later (with valid authentication). To improve security we will eventually use authentication certificates in conjunction with some standard certificates server.

While an ad-hoc communication scheme such as this is certainly a feasible approach for implementing large scale Web computing systems like Net //ELLPACK, the approach that best utilizes today's technologies would be to use a Common Object Request Broker Architecture (CORBA [xxx-corba]) compliant Object Request Broker (ORB) for the client-server communication. Using such ORBs (which intrinsically use standard binary protocols) would eliminate some of the negatives of binary protocols (since CORBA defines a machine independent type system with which types and interfaces may be defined) and also significantly ease the development effort. Two Java-based CORBA-compliant ORBs are currently available (HORB [xxx-horb] and JOE [xxx-joe]). A related feature of Java, the Java Distributed Object Method (JDOM) [xxx-jdom] and the Remote Method Invocation (RMI) [xxx-rmi] scheme, is also available. However, as these systems are still in their early stages of evolution we have chosen not to use them in the current version of Net //ELLPACK.

5.3 Designing Applet User Interfaces

Large scale software systems tend to have large, complex GUIs for controlling and manipulating them. In addition to the cost of downloading such large GUIs, it is not clear that the original GUI design would be appropriate for Web applications. The GUI building facilities in Java (and similar languages) do allow one to build user interfaces that are comparable to ones built using X or Microsoft Windows. But our intuition is that an applet downloaded from the Web should not have the same look and feel as an installed application on one's machine. Due to the dearth of large scale Web computing systems today, it is difficult to evaluate the validity of this intuition. We expect to use the Net //ELLPACK system as a testbed for user interface design ideas after it is fully deployed.

5.4 Reusing Existing Components

An important feature of systems such as Net //ELLPACK that build Web computing systems from existing software is reuse of legacy code from the original system. Since in the Web enabled version the user interface will run on the client end, that part must clearly be re-engineered and re-developed using the GUI capabilities of the language / environment used. For computational components, the custom server must be able to invoke legacy code components, either locally via subroutine calls or remotely via remote invocation techniques. This requires that the server be aware of the interfaces to these legacy components.

Similarly, there are hundred and perhaps thousands of applets being produced by developers the world over that one must not preclude a Web computing system from using. The most useful of these from a scientific computing standpoint are probably scientific data visualization applets. Such applets typically display a given data set and support interactive manipulation as well. The data sets are given to these applets as URLs, rather than as files in the typical application scenario. However, in the design of Web computing systems we advocated in Section 4.1, all the state is maintained by the server which used a custom protocol to communicate with the applet. This would preclude using any applets which require the data to be available on the Web unless the server supported the HTTP protocol as well; i.e., the server must have a facility for exporting some or all of its data to the Web.

6.0 Net //ELLPACK

Net //ELLPACK is a Web computing system based on the //ELLPACK PDE problem solving environment. This Web computing system was designed along the guidelines discussed earlier. The design incorporates the security and other constraints of the Java programming environment. The current prototype implementation allows the user to graphically define a PDE problem domain, textually specify symbolic equations for the PDE operator, boundary condition and true solution, define the mesh/grid generator parameters and select an appropriate discretizer, indexer and linear solver. In terms of functionality, the version of Net //ELLPACK described here supports solving 2-dimensional, elliptic problems sequentially using available solvers only. The java based //ELLPACK graphical interface applet presented to the user, consists of the minimum functionality necessary to define a PDE problem to the //ELLPACK compute server. This minimizes the size of the java classes that the client applet needs to download from the server.

Our on-going work is addressing issues in using multiple Net //ELLPACK servers to form a virtual parallel machine to run parallel solvers as well as issues in allowing users to introduce their own solvers to the system as is possible with the //ELLPACK PSE. The limitations on the types of problems solved (i.e., restricting to only 2D problems) is not a technical one but a practical one; we do not have the person-power to produce a completely general version of Net //ELLPACK.

The architecture of this system includes Java-enabled Web browsers, Net //ELLPACK servers (NetPPDs), a custom communication protocol (NetPP) and a standard communication protocol (HTTP). When a user initially visits a Web site offering the Net //ELLPACK service, a Java applet with a choice of NetPPDs at various geographic locations is downloaded and presented to the user. The user can either select a preferred server, or the applet may suggest a selection based on the network connectivity distance between the client and a potential server. Once a NetPPD is selected, the Net //ELLPACK interface applets are downloaded to the Web client from the selected server's HTTP compliant component. The Net //ELLPACK interface (GUI components) use the NETPP protocol to communicate with the NetPPD server during the problem specification and solution stages. The user's solution data is made available via an

HTTP compliant NetPPD component, permitting for instance, the use of foreign data visualization applets for solution analysis. In the rest of this section we describe each component of this architecture in more detail.

6.1 The Net //ELLPACK Web Server

The Net //ELLPACK Web server offers users a selection of Net //ELLPACK servers located at many sites worldwide. This selection is dynamically configurable to permit the addition of new servers and the deletion of out-of-service or inaccessible servers.

When a user visits this website and selects a NetPPD location, it invokes a CGI script on the Net //ELLPACK Web server. The script sends an HTTP redirect response which forces the client Web browser to contact the selected NetPPD server using the HTTP protocol. When initially contacted by a client, the HTTP compliant NetPPD component responds with a NetPP login applet. The NetPP client applet and NetPPD establish a reliable stream connection and communicate using the NetPP protocol.

6.2 NetPP: The Net //ELLPACK Communication Protocol

The NetPP communication protocol is a simple, text-based, request-reponse type, session-oriented, stateful protocol. Although it is broadly similar to the HTTP protocol, it is designed for communication over a single reliable byte stream connection. The protocol is based on the interaction pattern between a user and a PSE. It is designed to shield the user as much as possible from some of the inherent problems associated with remote applications: slow response time, bandwidth limitations, loss of connectivity and client memory limitations. It is also designed to protect the server host from any potentially malicious intrusions.

Each authenticated user is assigned a non-hierarchical directory space on the NetPPD server host. After successfully logging in, the remote user's client applet generates a NetPP request-response sequence with the NetPPD server to setup a computation session. The user specifies a session name to either start a new problem solving session or to restart a previously saved session. A subdirectory within the user's space on the NetPPD server host is associated with each session.

For security purposes, the NetPP protocol does not permit any indiscriminate browsing of the user spaces on the server host. The protocol also prevents any downloading or uploading of files between the client and user space on the NetPPD server host. However for solution analysis, remote users are allowed to download solution data files via an HTTP compliant NetPPD server component. Supporting the HTTP protocol in this manner requires precautionary security measures. Web security protocols are meant for human interaction and are not designed for Web computing system client interactions. It is also unintuitive for a client applet to request HTTP security information solely for the solution analysis phase while the other problem solving phases interact with a different security model. Our solution to this security concern is to use the private, secure NetPP protocol to leverage the security in the HTTP protocol compliant NetPPD server components. To this end, the NetPPD server creates a random path containing the solution data and communicates this path (URL) to the client applet using the NetPP protocol. The client may then supply this URL to foreign data visualization and analysis applets and assume that the supplied data is safe. The user's solution data is made available in this manner only as long as the user is logged on to the NetPPD server. Once the user logs out, this temporary location is made inaccessible. The NetPP protocol also provides users with the capability of selecting and specifying the format of the solution files that are to be made Web accessible. For instance, when a client sends a NetPP request for a solution file in a particular format, the NetPPD server performs the requisite conversion, makes the file Web accessible and sends a response message back to the remote NetPP client with the corresponding temporarily accessible URL.

To address the problem of slow response time, the NetPP communication protocol has been designed to minimize the interactions between the remote client and the server. For instance in the PDE problem specification phase, communication between the client and server occurs only when the user presses a submit button after each definition stage. The user can request periodic checkpointing to allow for easy recovery in the event of a network failure. If requested, checkpointing occurs after each client-server communication stage and the client's current state is saved on the NetPPD server host in the architecture-independent XDR format (xxx). PDE problem specifications often result in a large amount of data. Due to possible bandwidth limitations and slow connectivity, the NetPP protocol has been

designed to refrain from the movement of large data files to and from the client and server. For example, when the server generates a mesh for a user-defined PDE domain, it extracts and transmits only the control points neccessary for visualization back to the client. The client applet then reconstructs the mesh structure and displays it to the remote user.

6.3 NetPPD: The Net //ELLPACK Daemon

The //ELLPACK compute server (NetPPD) is a stateful, concurrent server with the ability to handle multiple NetPP clients simultaneously. NetPPD provides the important interface between remote NetPP GUI clients and the //ELLPACK PSE backend. It is connected to the //ELLPACK PSE components and the //ELLPACK libraries via a software bus. NetPPD has two communication components operating on separate ports; a NetPP request handler and a limited HTTP request handler. The HTTP handler is restricted to honor only the HTTP GET/HEAD requests. In addition to its communication components, NetPPD also has logging and security modules. Figure 9 illustrates the overall software architecture of the NetPPD server. Due to the compute intensive nature of its operation, NetPPD was designed to be lightweight. It is even possible to run multiple copies of NetPPD on a single machine with each communication handler listening on different ports. This design feature has been useful in our on-going research on using Net //ELLPACK in a distributed, collaborative scientific agent system.

The compute server requires a fairly large disk space allocation. It provides non-hierarchical, session-based directory spaces for each user. All intermediate files generated during the problem specification and execution stages are stored in these locations. This includes potentially large mesh files that are generated during the session. The user's checkpointing data is also stored in this location, enabling the user to restart a computation or problem specification after a break in connectivity. NetPPD provides randomly named, temporary, Web accessible locations for the solution data. This space is reclaimed at the end of each session. The GUI client applet class files are stored in the Web accessible area in NetPPD. These Java class files are archived to minimize the initial class download time. Archived Java classes are downloaded in a single HTTP request-response sequence unlike regular class files which result in multiple HTTP request-response sequences.

The intrinsic problem specifications that NetPPD receives from the remote GUI client are stored in specialized PDE data structures [xxx-PDELab paper]. For optimum performance, these data structures are kept in-core. However, they may be serialized and saved if the user requests checkpointing. Some problem specifications require the invocation of external programs incorporated within the //ELLPACK system. For instance, the //ELLPACK PSE includes several finite element mesh generators. When the remote client requests a mesh to be generated for its domain specification, NetPPD's //ELLPACK interface component invokes the external mesh generation library module with the necessary

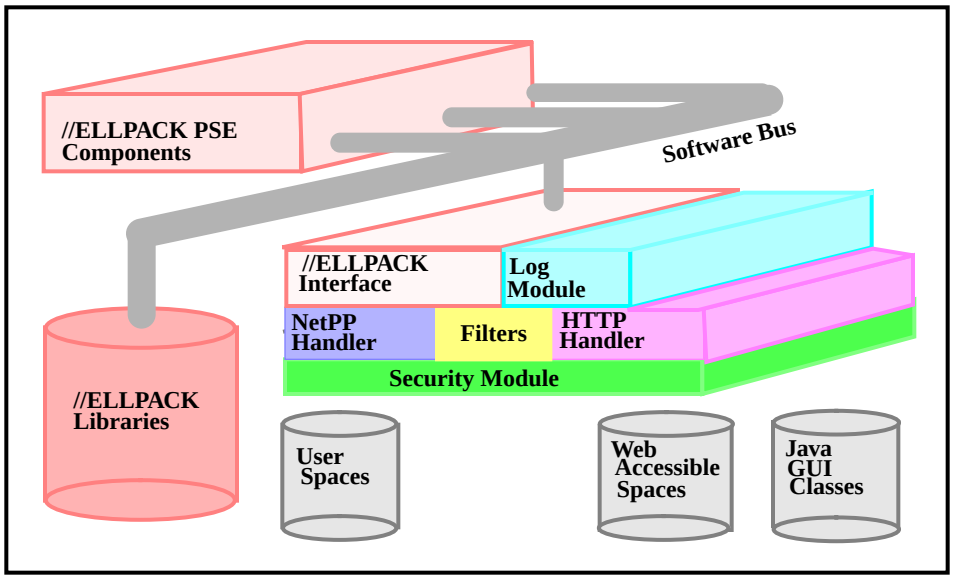

FIGURE 9. NetPPD Software Architecture

parameters. NetPPD then extracts the mesh display points from the generated mesh specification and transmits them to the client GUI. The generated mesh is saved in a file for future reference. To reduce the communication cost, some problem specification tasks are performed entirely by the GUI client which transmits just the resulting pertinent information back to NetPPD. For instance, if the remote user requests a grid based domain decomposition, the grid is generated and displayed by the GUI client applet and only the grid parameters are transmitted to the compute server.

Once the user has completed the PDE problem specification and requests the problem be solved, NetPPD converts and saves the specification data from either the latest checkpointed memory state or from its current memory state into the //ELLPACK natural PDE specification language. This PDE language specification is saved in a text file known as the .e file. NetPPD then invokes //ELLPACK compute engine with this problem specification file (.e file) as input. If requested by the user, NetPPD transmits the //ELLPACK compute engine execution status messages via the NetPP protocol, to be displayed on the remote clients log window.

The //ELLPACK PSE includes a comprehensive, graphical execution tool (Figure 3) to assist in the program compilation, execution and post-processing. This execution tool does the necessary access and availability checks for all the data files required during the computation. In the case of parallel execution, the execute tool transparently starts the program execution on the user specified parallel architecture using a user defined communication library. NetPPD has the necessary intelligence built in to handle this complex execution process. NetPPD has the capability to initiate a parallel execution on a high speed local area network using a communication library such as MPI or PVM, or on a parallel machine using a native communication library. At the end of the parallel or sequential computation, the PDE problem solution and the execution timing information are stored in the user's session specific storage space. Depending on the remote user's selection, this solution is filtered into a particular visualization format and moved into a temporary Web accessible location for analysis and visualization. NetPPD sends the solution URL to the GUI client which then accesses it via the specified foreign scientific visualization applet or a Web browser helper or plug-in application such as a VRML viewer (xxx).

The design of NetPPD is not dependent on the communication channel between itself and the remote GUI client. It would be possible to incorporate an emerging Web technology to substitute the current connection-oriented TCP level communication. For instance the Castenet channel transmitter system (xxx) along with its eventual Web browser bundled tuner could be used to provide the communication layer. The Net //ELLPACK GUI client could then be stored on the remote client machine and transparently updated via the Castenet update mechanism, greatly reducing the initial start-up communication cost. It would also provide the added benefit of an existing proxy server to enable Net //ELLPACK access to remote clients behind firewalls. The automatic update mechanism within the Castanet system could be extended to the solution anaysis process also by transparently downloading the computed solution in a user specified format onto the remote client's machine. Any foreign visualization applets or applications could also be downloaded at the same time.

The ease of seamlessly and effectively Web enabling the //ELLPACK PSE via the //ELLPACK interface component of NetPPD can be attributed to //ELLPACK's loosely coupled component architecture. On tests conducted so far, the design and implementation of NetPPD has proven to be fast, effective, safe, secure and scalable. NetPPD itself was designed with inherent characteristics of a PSE and is extensible and loosely coupled. Its overall architecture and framework could be used to effectively Web enable any other scientific PSE with suitable modifications to the private communication protocol.

6.4 The Net //ELLPACK GUI Components

The //ELLPACK system has a complex graphical user interface (Figures 2, 3 and 4 show an instances of it). The design used for the //ELLPACK GUI is along the lines of a graphical editor per PDE problem / solution component. For example, one editor allows users to specify the differential equation to be solver. Another editor allows the user to specify the domain on which the equation holds. Using the //ELLPACK language, a separate editor shows the problem components that have so far been specified. While this is a powerful organization of a GUI of a PDE computing system, this structure results in a large amount of code being used to provide this interface. In a separate issue, the //ELLPACK GUI is designed with the intention that the user is willing to use most of his/her screen real estate for the //ELLPACK environment's operation.

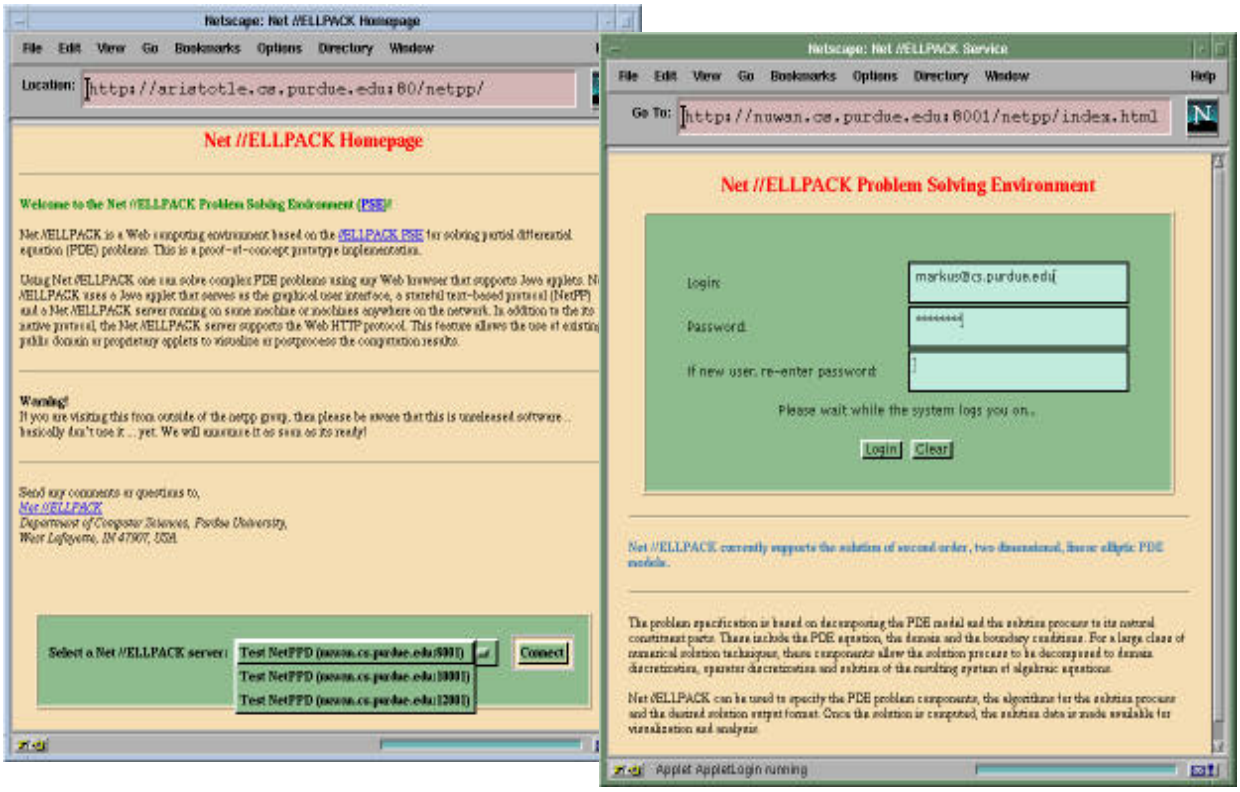

FIGURE 10. The Net //ELLPACK Web server homepage and a NetPPD login applet

In the case of applications run from the Web, both of these issues must be re-evaluated. First, the structure of user interfaces must be designed so that they consist of many small pieces which can be brought over on demand. Having such a modular structure allows the GUI to become operational quickly and also to only bring over just the pieces of code that the user is using in the current session. For example, the //ELLPACK equation editor has several templates for specifying the equation. The user selects one of these based on some properties of the problem at hand and on the solution scheme to be applied. In the applet case, rather than downloading all the functionality initially, we have restructured the interface so that only the framework of the user interface (the part that allows the user to select a template) is initially downloaded. The code for a certain template is downloaded only when that is selected by the user. A similar example is present in the domain editor: the editor has many options for how boundary pieces will be constructed. The algorithm and code for interpolating control points for each of these options will be downloaded only when that option is selected. In addition to the micro-structure issues, we also restructure the macro organization of the GUI in a similar manner: a basic framework which provides the user with the ability to bring over the needed functionality on demand. From the user's point-of-view, this has resulted in our changing the interface model to one that follows the object being constructructed rather than the process being used to construct that object. The entire GUI is now built around a graphical monitor/editor that shows the problem and solution being defined. The functionality available at any point (in terms of menu actions and buttons etc.) is based on the activity in progress (defining the equation, drawing the domain etc.) and is changed by a set of controls that select the current activity. Since in our system design the state is maintained at the server, once an activity is completed it must be "committed" to the server before one is allowed to change the activity.

This approach of having the GUI be based on a framework that follows the current activity implicity addresses the second issue we listed above of screen real estate usage. The new design uses exactly one top-level window for the entire GUI. The //ELLPACK GUI in comparison uses many windows.

The design of GUIs for Web computing system is very much of an open research issue. Due to the lack of real applications that operate in this form, there have been very few lessons learnt as yet. We expect to use the Net //ELLPACK system's GUI as a testbed to test various GUI design approaches in order to develop the most natural and effective interfaces for such Web computing systems.

7.0 Example: Solving a PDE Problem with Net //ELLPACK

In this section we present a typical PDE problem solving session within Net //ELLPACK. A sample set of response timings and computation timings for remote access via the internet from several geographical locations is presented in Table 1.

In this example, we use the widely available Netscape Web browser to access the Net //ELLPACK Web server. The initial NetPPD selection page and the subsequent login page from the selected NetPPD are shown in Figure 11.

When the remote user is succefully authenticated and logged in, the NetPP client applet is displayed. The user may then begin by selecting the "new session" option within the file menu and entering a new session name. Once the new session has been succesfully negotiated, the Net //ELLPACK PDE problem specification applet is displayed to the remote user. The user first defines the domain of the PDE problem by clicking the control points counterclockwise on the canvas and completes the last edge by <shift> clicking (Figure 11). Similar to the //ELLPACK PSE domain editor, the Net //ELLPACK domain specification editor also allows the user to enter the desired X and Y direction ranges.

The user then decides on the general solution strategy by selecting the domain discretization scheme. If a grid discretization is selected, then the subsequent templates presented to the user by the GUI client will be restricted to finite difference operator discretization methods. If a mesh discretization scheme is selected, then the user will be presented

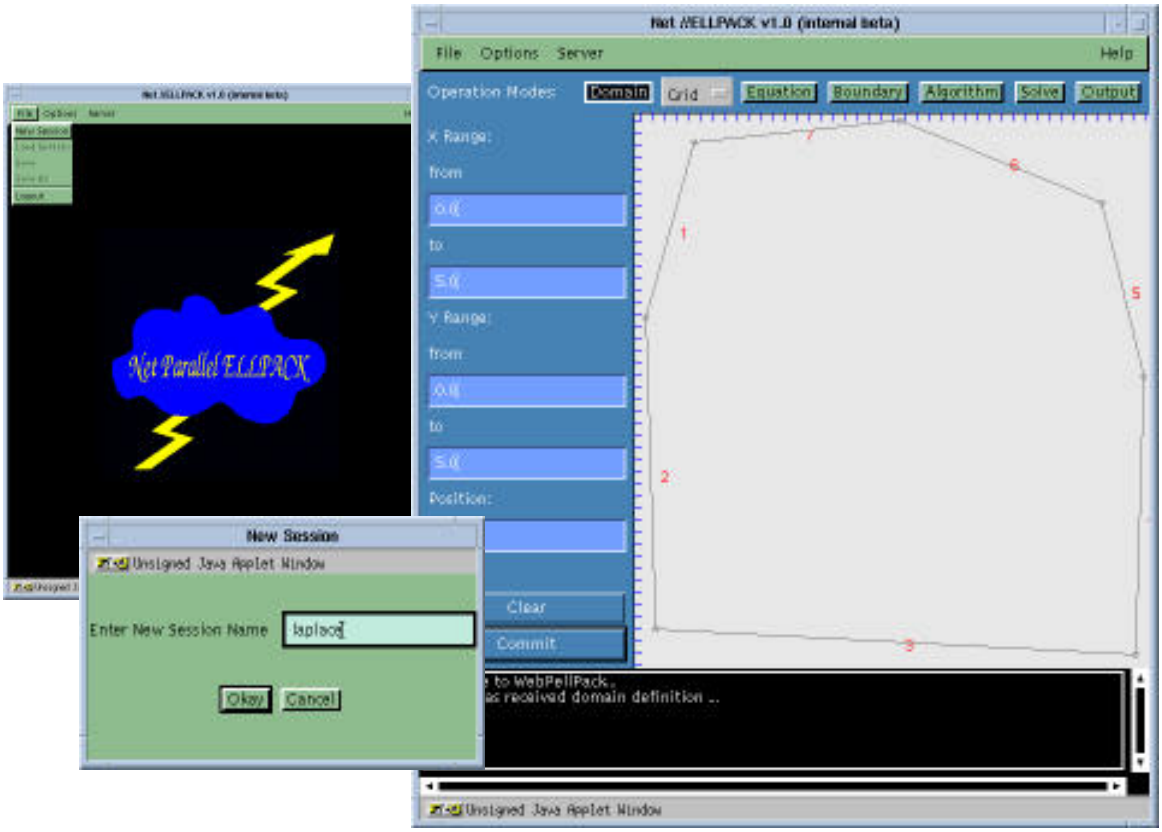

FIGURE 11. The NetPP startup client, session applet and the domain specification editor

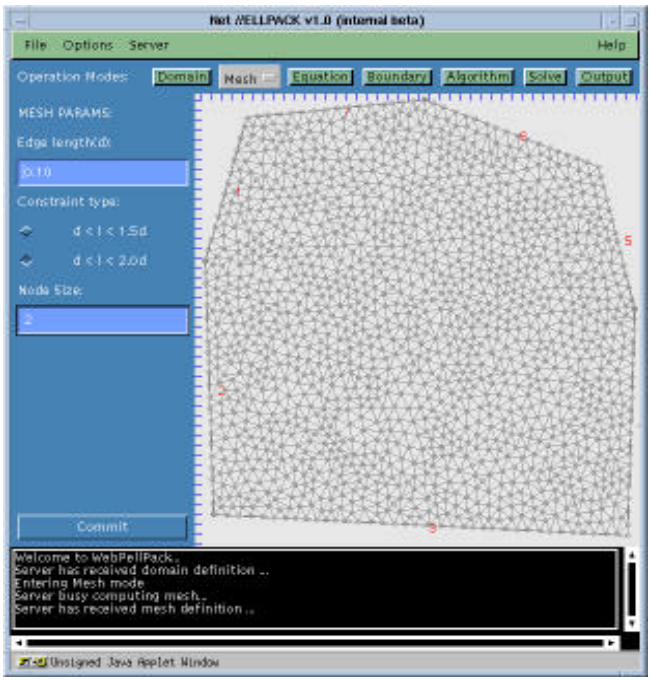

FIGURE 12. Net //ELLPACK mesh generator interface

with templates for finite element discretization methods. Figure 12 shows an instance of a mesh generation for the given domain definition. The mesh generator interface presented to the user has most of the options available within the standard //ELLPACK PSE mesh generator.

Figure 15 shows the equation and true solution specification interface and the algorithm specification editor. For this example, we have chosen a simple Laplace PDE problem with dirichlet boundary conditions and we force the true solution, $Sin(4x)Cos(4x)$. At each stage the user has to commit the problem specification component. Figure 13 shows the boundary condition specification interface. The true solution is specified as the boundary condition for each boundary piece since this is a PDE problem with Dirichlet boundary conditions. The boundary condition specification editor is tailored for the current boundary definition (in this example, a boundary with seven pieces).

Once the problem specification is complete, the remote user submits the problem for solution by the //ELLPACK PSE via the solve interface. When the problem solution is completed, a message is displayed on the remote client's log window. The remote user can then use the output specification interface to select the output format for solution visualization. Depending on the output format type, the remote client will display the problem solution either within the remote user's Web browser or as a Web browser helper or plug-in application. Figure 14 shows the output specification interface and the problem solution (in othe off [xxx] format) displayed within the remote user's Web browser. The solution visualization applet used in this case is a foreign scientific data visualization applet [xxx].

8.0 Conclusion

Internet and Intranet based servers have already become the common way of delivery services in many economical and goverment sectors today. The scenario of scientific computing servers and services updated and maintained transperantly and accessible through the internet has just appeared as a future research goal in many scientific circles. Our experience in building Net//ELLPACK, a Web-based PSE, has shown that net-centric scenario for scientific computing is feasible even with current technologies and bandwidths. The challenge of preserving the high level interactive user interfaces during remote computations has been overcome using the technology of applets and Java. Other (and maybe even better) solutions may be possible using upcoming net-centric computing platforms (e.g., inferno). We have not seen the beginning of the revolution yet!

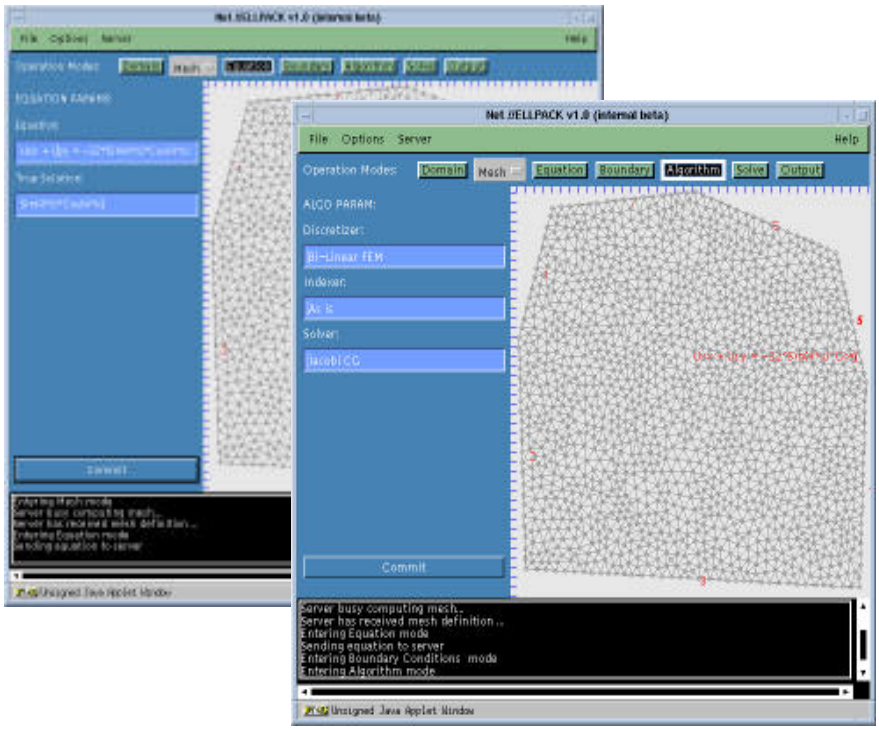

FIGURE 15. Net //ELLPACK equation & true solution specification interface and algorithm editor

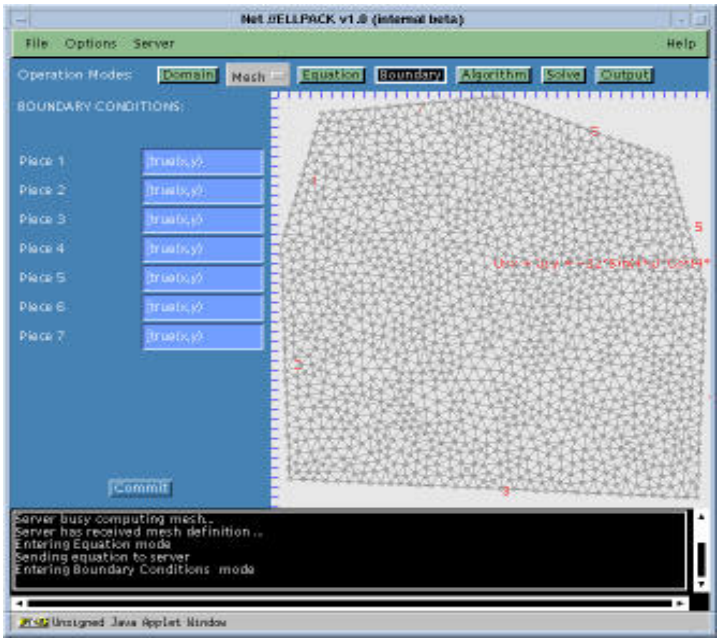

FIGURE 13. Net //ELLPACK boundary condition specification editor

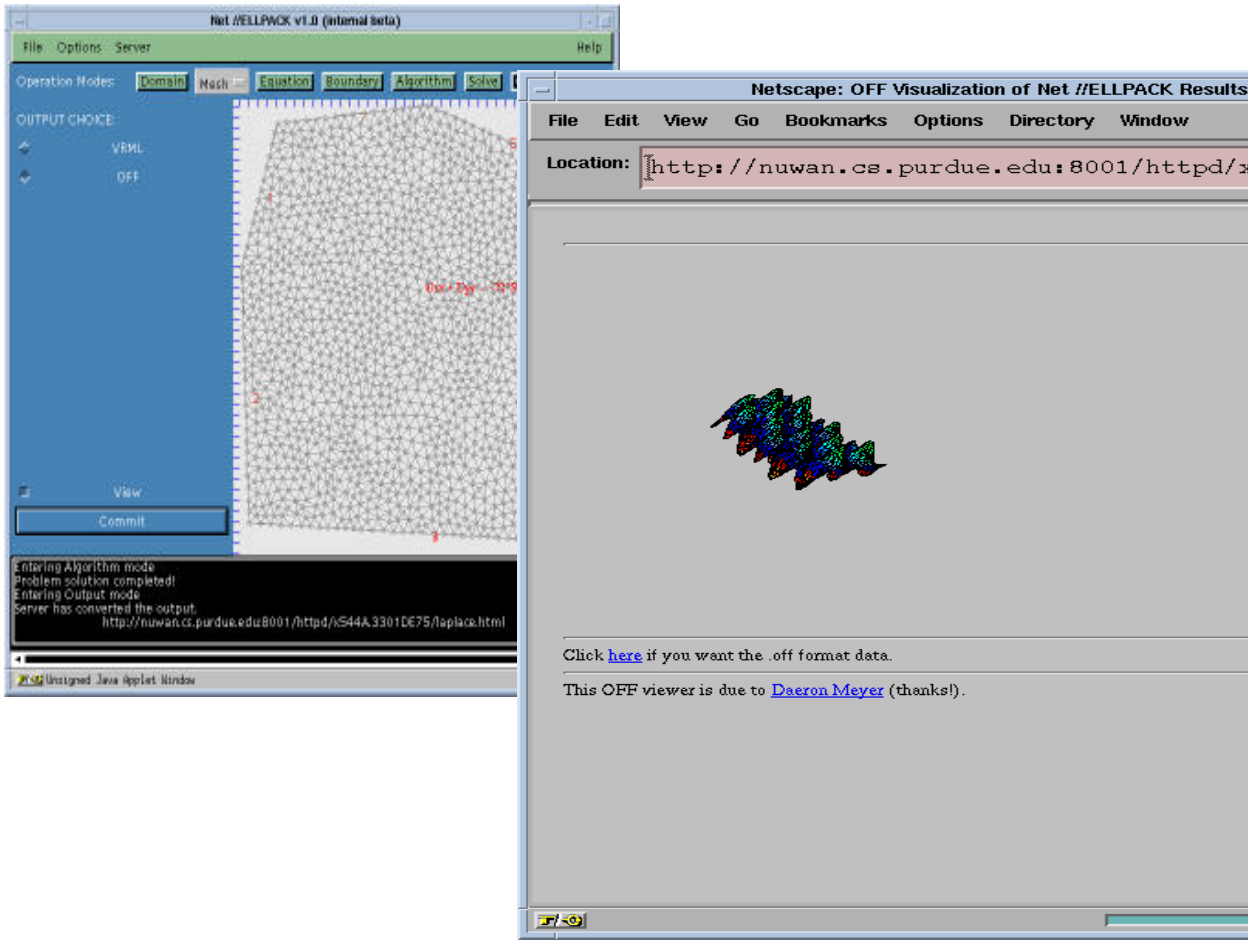

FIGURE 14. Net //ELLPACK solution specification interface and a foreign solution data visualization applet

9.0 References

[1] Houstis, E.N., Rice, J.R., Weerawarana, S., Catlin, A.C., Papchiou, P.N., Wang, K.-Y., Gaitatzes, M.G. 1996, Parallel (//) ELLPACK: A Problem Solving Environments for PDE Based Applications on Multicomputer Platforms. *CSD-TR-96-070*. Department of Computer Sciences. Purdue University.

[2] Gallopoulos, E., Houstis, E.N., and Rice, J. R. 1994. Computer as thinker/doer: Problem solving environments for computational science. *IEEE Comp. Sci. Engr., 1*, 11–23

[3] http://www.mcs.anl.gov/home/otc/Server/.

[4] Mathematica network server

[5] Maxima network server

[6] Maple network server

[7] Weerawarana, S., Houstis, E.N., Rice, J. R., Gaitatzes, M.G., Markus, S., and Joshi, A. 1996. Web //ELLPACK: A Networked Computing Service on the World Wide Web. *CSD-TR-95-011*. Department of Computer Science, Purdue University.

[8] Gross, L., Roll, C., and Schoenauer, W. 1993. VECFEM for mixed finite elements. *Technical Report Interner Bericht Nr. 50/9.* Rechenzentrum der Universitat Karlsruhe.

[9] FIDISOL ref

[10] Madsen, N.K. and Sincovec, R.F. 1979. Algorithm 540: PDECOL, general collocation software for partial differential equations, *ACM Trans. Math. Software, 5*, 326-351.

[11] PARC reference ?

[12] Kim, S. B., Houstis, E. N., and Rice, J. R. 1994. Parallel stationary iterative methods and their performance. Marinescu, D. and Frost, R. (Eds.), *INTEL Supercomputer Users Group Conference*.

[13] MGGXX (?)

[14] Stevens, W.R., *Unix Network Programming*, Prentice Hall, 1990.

[15] CORBA

[16] DCOM

[17] Weerawarana, S., Houstis, E.N., Rice, J. R., Catlin, A.C., Gaitatzes, M.G., Crabill, C.L., Markus, S., and Drashansky, T.T. 1996. The Purdue PSE Kernel. *CSD-TR-96-082.* Department of Computer Science, Purdue University.

[18] http://www.cs.utk.edu/netsolve/.

[19] http://www.javasoft.com/.

[20] http://surfit.anu.edu.au/SurfIt/.

[21] oblets

[22] inferno

[23] javascript

[24] http://ring.etl.go.jp/openlab/horb/

[25] joe

[26] java-idl

[27] RMI

[28] pdelab

[29] VRML

[30] http://www.marimba.com/

[31] off format

[32] off visualization applet url