The use of Java in the NetSolve project

H. Casanova
University of Tennessee
Department of Computer Science, 104 Ayres Hall
Knoxville, TN 37996-1301, U.S.A
e-mail: casanova@cs.utk.edu

J. Dongarra
University of Tennessee
Department of Computer Science, 104 Ayres Hall
Knoxville, TN 37996-1301, U.S.A
e-mail: dongarra@cs.utk.edu
and
Oak Ridge National Laboratory
Mathematical Science Section, PO Box 2008, Building 6012
Oak Ridge, TN 37821-6367, U.S.A
e-mail: dongarra@msr.epm.ornl.gov

Keywords: Internet Computing, Scientific Computing, User Interfaces, Load Balancing, Fault Tolerance, Computational Servers, Virtual Libraries.

ABSTRACT

The NetSolve project, underway at the University of Tennessee and Oak Ridge National Laboratory, allows users to access computational resources, such as hardware and software, distributed across the network. Thanks to these resources, the user can easily perform scientific computing tasks without having any computing resource installed on his/her computer. There are many research issues involved in the NetSolve system, including Internet computing, fault-tolerance, load balancing, user-interface design, computational servers, and virtual libraries. After providing an overview of the project in its latest development stage, this paper emphasizes two particular aspects of our work. First, we describe our computational server paradigm and how these servers implement virtual libraries. The computational server creation/update process is facilitated by a Java applet. Second, we describe a new graphical interface to NetSolve since a constant concern of the NetSolve project is to provide different interfaces for a large number of target users. This graphical interface has been written in Java as well.

INTRODUCTION

Scientific computing has been a major part of both research centers and industry for many years. As such, it has been the object of many investigations which have led to the development of numerous software products. These products can be classified into different categories. Some numerical tools, like MATLAB [1] or Mathematica [2], have enjoyed great success. These tools generally provide an interactive interface as well as the possibility of writing scripts to perform computation.

Another class of products falls under the category of numerical libraries. Numerical libraries are less convenient than interactive tools, since the user generally is required to write a C or Fortran program. Nevertheless, they offer several advantages. A very large number of such libraries exists and they cover diverse fields of computational science. Moreover, unlike interactive tools, numerical libraries are often freely available.

A third class of tools comprises runtime packages whose goal is to help the user perform some specific type of built-in, scientific computation. Like numerical libraries, such packages are usually freely available. One example is NEOS [3], which is focused on linear programming and optimization.

Users wanting to solve a numerical problem are thus confronted with a dilemma. They can purchase a commercial product and take the risk that it might not be suitable for future use on different kinds of problems, or they can try to locate and download free libraries and write programs in terms of specific functions or subroutines. In this paper, we address the second situation, where the user is confronted with many difficulties.

The first task the user must undertake is to look for the appropriate library or set of libraries he/she needs for their own computational problem. Usually, such libraries can be found in software repositories. A well-known repository, for example, is Netlib [4] which is maintained through the collaborative effort of several institutions and universities. Software repositories present some intrinsic difficulties for the inexperienced user : (i) they are generally very large and (ii) contain very different types of libraries. Once located, the appropriate library must be downloaded and installed. Depending on the nature of the software, this step might be nontrivial, especially for a user not used to this kind of task. However, the biggest steps still remain; learning how to use the library itself and how to write a program in terms of its component. Such a task can be formidable and time-consuming (even without regard for the debugging phase).

These considerations motivated the establishment of the NetSolve project. NetSolve is a client-server application designed to solve computational science problems over a network. A number of different interfaces have been developed for the NetSolve software so that users of C, Fortran, MATLAB, or the Web can easily use the NetSolve system. The underlying computational software can be any scientific package, thereby ensuring good performance results. Moreover, NetSolve uses a load-balancing strategy to improve the use of the computational resources available. The following section gives an overview of the NetSolve system.

OVERVIEW OF NETSOLVE

Architecture

The structure of the NetSolve system organization is depicted in Figure 1. To solve the challenges highlighted in the introduction, NetSolve provides the user with a pool of **computational resources**. These resources are in fact computational servers that provide run-time access to arbitrary numerical libraries. The user can use one of the different NetSolve **client** interfaces to send requests to these servers. The user requests, however, are not sent directly to the computational resources, but are instead processed by another component of the system: a NetSolve **agent**. The agent decides which computational server should handle the user request and assigns the request to that server.

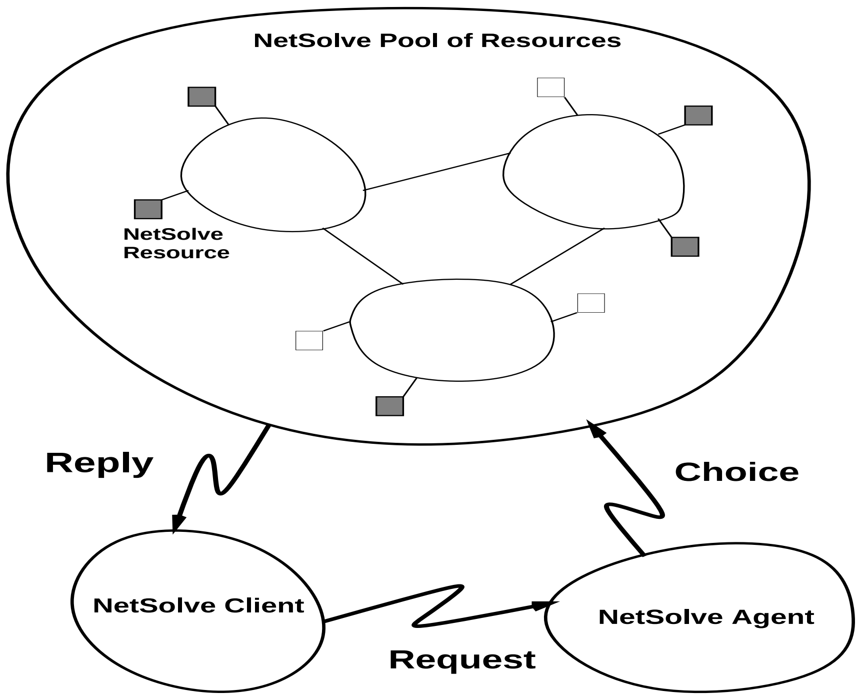

Figure 1: The NetSolve System

The different hosts that participate in the NetSolve protocol may be located anywhere on the Internet. In fact, they may be administered by different institutions. NetSolve does not assume any centralized control over the different hosts in the system. On the contrary, each process (computational server or agent) is an independent entity: it can be stopped/restarted safely at anytime without putting the integrity of the system in jeopardy. Furthermore, a NetSolve system can contain several instances of the NetSolve agent. Suppose, for example, the set of computational resources spans several local area networks and that users on each of these networks want to use NetSolve to perform scientific computations. It is then possible to start a NetSolve agent on each network, so that user requests always go to the "closest" agent to be processed. Different instances of the NetSolve agent can then have different views of the set of computational resources, reflecting the fact that certain clients are closer to certain computational resources.

Load Balancing

The main role of the agent in the NetSolve system is to perform load balancing among the different computational resources. NetSolve is inherently a multi-request system. Several users can compete for the resources by contacting the same agent or different agents managing the same pool of resources. In fact, even a single user can send multiple asynchronous requests at once as explained in the description of the user interfaces. For each incoming request, the NetSolve agent has to choose a computational server where the numerical computation will be performed. For each server, the agent can use information contained in the user request (type of computation, size of the problem, ...), static information about the server (speed of the host, numerical server available, etc.), and **predictions** about the workload of the server's host and the distance to the server's host over the network. These different pieces of information are then combined to obtain an estimate of the time required to process the user request on each computational server (including network time and CPU time). For each request, the NetSolve agent sorts the appropriate computational servers according to these estimated times and is then able to process the request accordingly. More details on the way the agent performs this load balancing can be found in [5].

Fault Tolerance

As previously mentioned, the hosts in the NetSolve system can be located anywhere on the Internet and can therefore be administered by different institutions. This is the reason why NetSolve does not try to impose any control on the different resources. This approach is of course very flexible, but it requires that NetSolve implement some kind of fault tolerance mechanisms. Indeed, any resource can become unreachable at any moment, perhaps because of a network failure, a host failure, or simply a system administrator rebooting a host.

The NetSolve system ensures that a user request will be completed unless every single resource has failed. When a client sends a request to a NetSolve agent, it receives a sorted list of computational servers to try. When one of these servers has been successfully contacted, the numerical computation is started. If the contacted server fails during the computation, then another server is contacted and the computation is restarted. This whole process is transparent to the user. If all the servers have failed, then the user is notified that the computation can not be performed at that time. This strategy is only a first step towards fault tolerance and will be improved in future versions of the software.

Multiple User Interfaces

A main concern of the NetSolve design was to provide several interfaces for a wide range of target users. For now, NetSolve provides a C interface, a Fortran interface, a MATLAB interface and a graphical Java interface. The Java interface has been added quite recently and is the object of the last section of this paper.

Another concern was keeping the interfaces as simple a possible. For example, the MATLAB interface contains only 2 functions : `netsolve()` and `netsolve_nb()`. The first function allows the user to send *blocking* requests to NetSolve whereas the second one sends *non-blocking* requests. In fact, every interface provides non-blocking calls to NetSolve. When several non-blocking requests are sent to a NetSolve agent, they are dispatched between the available computational resources according to the load balancing schemes implemented by the agent. With minimal effort, this is how the user can achieve some degree of parallelism. More details on the various interfaces as well as examples can be found in [6].

COMPUTATIONAL SERVER PARADIGM

As mentioned in the introduction, the user is confronted with several difficulties when wanting to use freely available, numerical software. NetSolve tries to remedy almost all of these difficulties. The solution comes in the form of computational servers that have the following abilities :

- **Uniform access to the software**: The servers give users the illusion that they have access to a uniform set of subroutines/functions.
- **Configurability**: The servers are not limited to any particular software and must therefore use a general framework to integrate new functionalities in an easy way. NetSolve will then have the ability to extend and encompass new numerical applications at will.
- **Preinstallation** : Numerical software is of course pre-installed on the servers' sites. The user is not responsible for installing or maintaining any numerical software at all.

To make the implementation of such a computational server model possible, we designed a machine-independent, general way of describing a numerical computation. The main component of this framework is a descriptive language that is used to describe each separate numerical functionality of a computational server. Files written in this language can be compiled into actual computational servers which are executable on any UNIX platform. The NetSolve software contains a compiler that performs this translation. In the following paragraphs, we briefly describe the general concepts behind the language, as well as how we use a Java applet to easily generate description files, and thus computational servers.

Problem Specification

We call a *problem* any self-contained, numerical computation that can be performed by a computational server. As such, it has *input* and *output*. Output is a function of the Input according to the *algorithm* of the numerical computation. For example, a problem could be defined as follows: a double-precision real matrix in input, a double-precision complex vector as output, and a double-precision eigensolver as algorithm. This problem computes the eigenvalues of a real matrix. The input and output objects can be of any Fortran data-type. At the moment, NetSolve supports the following data structures of the data types: (i) two-dimensional arrays (with leading dimension), (ii) one-dimensional arrays, and (iii) scalars. The description language possesses constructs to describe the inputs and outputs to a problem. This description is sufficient for the high-level interfaces of NetSolve (that is the Java and the MATLAB interfaces) since an object can be represented by a single identifier in these interfaces. The low-level interfaces (C and Fortran), however, cannot handle high-level objects directly. They need a more detailed way of accessing a computational functionality. In fact, low-level interfaces need a specific *calling sequence* description so that the user can call NetSolve from these low-level languages.

Arbitrary Calling Sequence

The description language also provides constructs to formally describe the calling sequence that users should use from the low-level interfaces for each problem. This calling sequence can be totally arbitrary, so that the flexibility of our system is increased. It is then possible, for example, to exactly match the calling sequence of the underlying numerical software. This can be interesting if code that is already written in terms of existing numerical libraries is to be converted to NetSolve. The conversion can be instantaneous if the NetSolve administrator setting up the server decides to preserve the original calling sequence. This is the policy that we have followed for the numerical software we have integrated in the NetSolve computational servers running at the University of Tennessee. Users can effortlessly convert their legacy code to a NetSolve implementation using those servers. Thus far, the following numerical packages have been successfully integrated with NetSolve : BLAS [7, 8, 9], LAPACK [10], ITPACK [11], FFTPack [12], MinPack [13], FitPack [14]. More software is constantly being integrated to expand NetSolve's range of applications.

Problem Descriptions

A problem description consists of a *problem specification*, a *calling sequence* description for the low-level interfaces, and a *pseudo-code*. The first two components have been described. The pseudo-code is the segment of code that the computational server executes when calling the underlying numerical software. This segment of code must take care of memory management details, possible error checking, etc. It is described in the pseudo-code section of the problem descriptions and is written with a mixture of C and predefined macros. It proved to be the most difficult part to write in the entire problem description when we integrated new numerical software with NetSolve. This is the reason why we have developed a Java applet that automatically generates description files.

Server creation/update with a Java interface

One of the strengths of NetSolve is that any arbitrary numerical software can be integrated in a computational server. NetSolve can therefore satisfy the changing needs of users performing numerical computations. This strength, however, is effective only if the process of creating new problems and adding them to a computational server is reasonably straightforward. For this reason, we developed a graphical interface to handle the generation of the description files. The interface performs all sorts of error checking of the user input which mostly consists of mouse clicks and choices in menus. Using the interface is much easier than creating a description file manually, especially as the complexity of the problem to be described increases.

Not only is this interface graphical, but it is also written in Java. Several factors motivated this choice. First, Java allows one to write GUIs (Graphical User Interfaces) very easily, thanks to its built-in widget classes. Second, Java is object oriented and therefore provides a good degree of modularity and data encapsulation. These features are important to us because we might have to modify the syntax of the language in the future to describe wider classes of problems. Third, Java is Web-enabled. This interface can thus be downloaded as an applet and users setting up NetSolve computational servers can create their description files directly from within Web browsers. These files can then be downloaded from the Web browser, already compiled into NetSolve computational servers thanks to the compiler provided by the NetSolve server software.

Something important to note here is that a description file for a given functionality of a given numerical library has to be created only once! This file can indeed be re-used by any NetSolve administrator to integrate the given functionality in his/her computational servers if desirable. This allows collaboration between different institutions and leads us to an idea that was envisioned in [15]. A description file repository could be created on the Web. From that repository, description files could be downloaded at will to set up computational servers. The actual numerical software should also be available to make the creation of computational servers almost immediate. The idea would then be to add a complementary repository of description files to a regular software repository (like Netlib [4]).

JAVA CLIENT INTERFACE

Why a Java interface?

For NetSolve to be accepted by many users, it needs to provide several distinct ways to access computational resources, mainly different user interfaces. Before the development of the Java interface, the only interfaces that were available were MATLAB, C, and Fortran, leaving the MATLAB interface as the only high-level and interactive way of calling NetSolve. However, MATLAB is not freely distributed and many users are not likely to have it installed on their workstations. We needed a freely available interactive interface so that users not familiar with scientific computing could still use NetSolve for their first steps in the field. Furthermore, Java allowed us to effortlessly design a graphical interface making it even more straightforward for the first-time users, even though an experienced user is more likely to use the Fortran or MATLAB interface if available.

Interface Design

Let us now review some of the major aspects of the design of the Java interface. We already said that it is entirely graphical. A set of available computational functionalities is presented to the user who can point and click to select the appropriate one. The user then supplies input by way of files, URLs, or interaction. Computation is handled as usual by a computational server chosen by the NetSolve agent and the result is returned to the user. The results can be viewed and saved if necessary.

The interface is multi-threaded. Using Java's built-in thread facilities, it is easy to assign one separate thread to each computation. Every request sent by the Java interface to the NetSolve computational resource is represented by an independent window. Opening several windows at the same time, the user can then send multiple requests simultaneously and these requests will be handled by different computational servers thanks to the agent's load balancing strategy. This is very similar to the non-blocking calls to NetSolve from C, Fortran, or MATLAB.

The pool of NetSolve resources can be a heterogeneous set of hosts, i.e., two hosts can have different internal data representations. To support heterogeneous environments, the agents, servers, and C, Fortran and MATLAB interfaces use the XDR protocol. The Java interface may then contact servers which expect XDR-encoded data. This is why we developed a Java XDR-encoder, enabling our interface to communicate with any server on any platform.

One of the features of Java is that the Java virtual machine can enforce a strict security scheme. Typically, the virtual machine embedded in a Web-browser does not allow file access, network connection except to the Web server, etc. Presently, the Java interface to NetSolve cannot be used as an applet within a Web browser because of these security restrictions. Indeed, our interface is supposed to contact many different servers scattered on the Internet and therefore perform multiple network connections. It is possible to work around this problem by setting up an appropriate daemon on the Web servers, but this would not be a viable solution in the long term, considering the mass of data that would transit through the Web server. However, the interface is fully functional as an application, running on a system which has a Java virtual machine. Considering the increased interest in Java, most systems already have an implementation of the Java Development Kit (JDK) that includes the Java virtual machine and the Java byte-code compiler. Our interface should therefore be usable on virtually any platform and future versions of Web-browsers are sure to make the applet approach feasible.

FUTURE DEVELOPMENTS

NetSolve is still a young project: at the time of this writing the BETA version is being released. Several new features are sure to be added to the agent, servers, and interfaces. When the project reaches a certain maturity, we will add several security features, perhaps using a Java approach, including user accounting or authentication mechanisms, data encryption, etc. The agent design is still experimental and can be greatly improved based on the experience we are accumulating on agent-based computation over the Internet, thanks to the present version of NetSolve.

The mechanism for setting up computational servers and integrating numerical software at will has proven to be reasonably efficient for a number of scientific libraries. The Java applet that will help advanced users to create new servers will undoubtedly be modified intensively as soon as we receive feedback from its users. Since it will be an applet, it will be very easy to maintain and we will be able to modify it constantly, if necessary, without preventing users from accessing it. These modifications will aim to make it even easier to use, without changing its semantics; we will ensure that any description file created with any version of the applet is a valid description file.

In a long term perspective, we will investigate the possibility of an alternate repository of description files **and** numerical software. Such a repository could then be accessed as a *NetSolve computational server database* and used to set up every possible NetSolve computational server on the Internet. Of course, this repository would be incremental in the sense that new description files would be constantly tested and incorporated. This kind of project raises numerous, difficult issues such as software authentication and testing. NetSolve will have to evolve to a new level of maturity before such a repository can be activated.

The Java interface seems to have reached quite a stable point in its development. We are now thinking of distributing a Java class Library Interface. Users could then call NetSolve from their Java programs since it is possible to do it in C or Fortran with the appropriate NetSolve interfaces. The Class Library Interface would use the same mechanisms as the Graphical Interface, and could allow users to perform efficient numerical computations from Java programs.

REFERENCES

[1] Inc. The Math Works. *MATLAB Reference Guide.* 1992.

[2] S. Wolfram. *The Mathematica Book, Third Edition.* Wolfram Median, Inc. and Cambridge University Press, 1996.

[3] J. Czyzyk, M. Mesnier, and J. Moré. Neos : The Network-Enabled Optimization System. Technical Report MCS-P615-1096, Mathematics and Computer Science Division, Argonne National Laboratory, 1996.

[4] S. Browne, J. Dongarra, E. Grosse, and T. Rowan. The Netlib Mathematical Software Repository. *D-Lib Magazine*, sep 1995. Accessible at http://www.dlib.org/.

[5] H Casanova and J. Dongarra. NetSolve: A network server for solving computational science problems. In *Proc. of Supercomputing'96, Pittsburgh.* Department of Computer Science, University of Tennessee, Knoxville, 1996. (to appear in *The International Journal of Supercomputer Applications and High Performance Computing*).

[6] H. Casanova, J. Dongarra, and K. Seymour. Client user's guide to netsolve. Technical Report CS-96-343, Department of Computer Science, University of Tennessee, 1996.

[7] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Transactions on Mathematical Software*, 5:308–325, 1979.

[8] J. Dongarra, J. Du Croz, S Hammarling, and R. Hanson. An extended set of fortran basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, 14(1):1–32, 1988.

[9] J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, 16(1):1–17, 1990.

[10] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide, Second Edition.* SIAM, Philadelphia, PA, 1995.

[11] D. Young, D. Kincaid, J. Respess, and R. Grimes. Itpack2c: a FORTRAN package for solving large sparse linear systems by adaptive accelerated iterative methods. Technical report, University of Texas at Austin, Boeing Computer Services Company, 1996.

[12] P. Swarztrauber. FftPack : Documentation file accessible at :
"ftp://ftp.ucar.edu/ftp/dsl/lib/fftpack/readme".

[13] J. Moré, B. Garbow, and K. Hillstrom. Minpack : Documentation file accessible at :
"http://www.netlib.org/minpack/readme".

[14] A. Cline. Scalar- and planar-valued curve fitting using splines under tension. *Communications of the ACM*, 17:218–220, 1974.

[15] S. Browne, H. Casanova, and J. Dongarra. Providing access to high performance computing technologies. In J. Wasniewski, J. Dongarra, K. Madsen, and D. Olesen, editors, *Lecture Notes in Computer Science 1184*, pages 123–134, Berlin, 1996. Department of Computer Science, University of Tennessee, Knoxville, Springer-Verlag.