Interactive and Dynamic Content in Software Repositories [*]

Ronald F. Boisvert
National Institute of Standards and Technology
boisvert@nist.gov

Shirley V. Browne
University of Tennessee
browne@cs.utk.edu

Jack J. Dongarra
University of Tennessee and Oak Ridge National Laboratory
dongarra@cs.utk.edu

Eric Grosse
Lucent Technologies
ehg@bell-labs.com

Bruce Miller
National Institute of Standards and Technology
bruce.miller@nist.gov

February 26, 1997

[*]This work is supported in part by the Defense Advanced Research Projects Agency under contract DAAH04-95-1-0595, administered by the U.S. Army Research Office.

Software repositories have traditionally provided access to software resources for particular communities of users within specific domains. For example, our Netlib [1] [6] and GAMS [2] [4] repositories provide access to collections of mathematical software, while our National HPCC Software Exchange (NHSE) [3] [5] provides access to high performance computing resources. The growth of the World Wide Web has created new opportunities for expanding the scope of discipline-oriented repositories, for reaching a wider community of users, and for expanding the types of services offered. Reaching a wider community of users has created a need for increased automated assistance in locating appropriate resources and in understanding and making use of these resources.

The goal of our software repository research is to improve access to tools for doing computational science for both expert and non-expert users. We are exploring the use of emerging Web and network technologies for enhancing repository usability and interactivity. Technologies such as Java, Inferno/Limbo, and remote execution services can interactively assist users in searching for, selecting, and using scientific software and computational tools. This paper describes various related prototype experimental interfaces and services we have developed for traversing a software classification hierarchy, for selection of software and test problems, and for remote execution of library software. After developing and testing our research prototypes, we deploy them in working network services useful to the computational science community.

Although the focus of our work has been on software repositories, we believe that many of the results are applicable to other types of digital libraries, especially with respect to the provision of interactive and dynamically generated content.

1 GAMS and HotGAMS

GAMS, the Guide to Available Mathematical Software, is a cross-index and virtual repository of mathematical and statistical software useful in science and engineering [2]. The majority of software indexed by GAMS represents subprograms for mathematical problems which commonly occur in computational science and engineering, such as solution of systems of linear algebraic equations, computing matrix eigenvalues, solving nonlinear systems of differential equations, finding minima of nonlinear functions of several variables, evaluating the special functions of applied mathematics, and performing nonlinear regression. All cataloged problem-solving software modules, of which there are over 10,000, are assigned one or more problem classifications from the GAMS 736-node tree-structured taxonomy of mathematical and statistical problems. Users can browse through modules in any given problem class. To find an appropriate class, one can utilize the taxonomy as a decision tree, or enter keywords which are then mapped to problem classes. Search filters can be declared which allow users to specify preferences such as computing precision or programming language. In addition, users can browse through all modules in a given package, all modules with a given name, or all modules with user-supplied keywords in their abstracts.

Classification systems have long been used to give structure to large bodies of information. A well-formulated system can improve understanding of the information as well as ease access to it, thus making the information more useful. To be effective, a software classification system must have the following properties:

- Problem-orientation

  It must classify the problems which can be solved by computer software. Other orientations, such as classification by algorithm or classification by software package, are of less interest to end users.

- Variable-level tree structure

  A tree structure is the most natural for a classification system. Allowing arbitrary levels of refinement permits the system to adapt to both mature and young subject areas. In young subject areas little software is available, and hence little refinement is necessary. In mature areas where much software is available, increased refinement is necessary to distinguish among the choices.

[1] `http://www.netlib.org/`
[2] `http://gams.nist.gov/`
[3] `http://www.nhse.org/`

The concept of *filters* helps users narrow down the number of desired software modules. Partitioning more than 10000 problem-solving software modules using a 736-node taxonomy necessarily leads to classes populated by a large number of modules. Differentiating among modules in a single class can then be quite tedious. Filters allow the user to specify additional preferences in a number of areas: language (e.g., Fortran, C), precision (e.g., single, double, multiple), access (e.g., free, proprietary), package, and repository. When presenting the user with modules in a given class, GAMS screen out those that do not satisfy the current set of filters.

Two interfaces to GAMS are available: an HTTP gateway and a Java-powered Web client. The native *gams* (command-line) and *xgams* (X11) client programs are still available for downloading from *math.nist.gov* but are obsolete and are being phased out. Although Web browsers provide a universal client interface, implementing sophisticated user actions is often awkward and sometimes impossible. For example, the *xgams* client requested information from the GAMS server about existing query filters and their possible values on startup. *xgams* could then present these to the user on demand, without any additional server interaction. By setting these filters, the user provided a profile that *xgams* used to qualify each subsequent query. Because implementing a persistent user profile in the context of HTTP would be at best awkward and inefficient, GAMS filters have not been implemented in the HTTP gateway.

The functionality that was lost in moving from the *xgams* client to a Web interface via the HTTP gateway has been regained through the use of Java, and additional functionality has been added. HotGAMS is a Java-powered GAMS client [4]. Taking advantage of Java allows us to improve interactivity in conducting searches and exploring the problem hierarchy. Currently two version of HotGAMS are operational. They both require a Java enabled browser. They differ essentially in how they present the main interaction window, and in where the referenced URLs (e.g., for documentation, subroutine sources, etc.) are displayed in the browser. The *HotGAMS* version appears on its own Web page and displays referenced URLs on the "next" page. *HotGAMS in a Frame* is a compact HotGAMS that appears within a Netscape Frame, with referenced URLs displayed in a separate frame in the same window. The frame version allows the user to select a sequence of URLs while keeping the HotGAMS applet visible and operational. With either version of HotGAMS, you may specify preferences (i.e., filters) in advance about the software you're interested in, such as the source language or numerical precision. You may also refine your selections after the fact to reduce the set of apparently appropriate modules to the truly relevant ones.

The HotGAMS applet itself consists of a set of tab cards across the top which provide ways of selecting or manipulating a set of software modules which are listed in the lower half of the applet. Clicking the tab selects the associated card. The following describes the functioning of the various cards.

- **Help Card**. The Help Card displays a welcome message and a set of buttons linking the user to various documents about the GAMS project and to the help text.

- **Classes Card**. This card shows the GAMS Problem Classification Hierarchy. A Class corresponds to a general class of mathematical problems. Clicking the mouse on a class opens it, revealing a set of software modules or the class's subclasses, or both. Clicking the class again will close it, hiding its parts. The subclasses represent more specific problems. Thus the hierarchy can be used as a decision tree, to find the most specific class that describes the problem the user wants to solve. The modules set represents the set of modules which solve problems of this class. Opening the set gets the modules and displays them in the Modules panel at lower half of the applet, where the user may then examine and/or download them.

- **Packages Card.** This card shows all software Packages known to the Gams server. A package is a collection, commercial or otherwise, of related software modules. Clicking a package opens it to reveal a module set (as in the Classes card) and a set of documents relating to the package.

- **Search Card**. The search card allows the user to search for Classes, Packages or Modules that match a given name. There is also an apropos function to find problem classes. This search results in a list

[4]`http://math.nist.gov/HotGAMS/`

of phrases in the index which match the given keyword, along with the class or classes appropriate to those phrases. The user opens the class corresponding to the phrase closest to his intended meaning to view that class in the Classes card. Other suggested keywords may also be shown; opening them performs a recursive apropos.

- **Preferences Card**. The preferences card allows the user to specify preferences for using HotGAMS. It provides a couple of switches to control the display, and it provides a set of global filters to restrict the set of modules shown.
- **Refine Card**. This card allows the user to refine a set of modules, once they have been selected from one of the other cards. Currently, the user interface is much the same as for the filters in the Preferences card, but the only attribute values shown are those in the currently selected set of modules (those passing global preferences, if any). This can be useful when a search has generated a list of too many modules; It shows the user what attributes differentiate the modules, and by selecting desirable properties the user may reduce the set to a more manageable size. In further research, we intend to explore including additional problem-dependent 'facets' which will further distinguish modules from one another. An example would be the type of boundary conditions on a differential equation solver.

The next step for systems such as GAMS is to provide expert-level advice on software selection. We are particularly interested in the situation where many similar, but not identical, pieces of software appear to meet the user's criteria. We are focusing on extensions to the filtering mechanism that support both simple knowledge representations and effective user interactions [1]. This issue of refining the search will be the subject of further research, with HotGAMS serving as the testbed.

2 Matrix Market and Dynamically Generated Content

The decomposition, solution, and eigenanalysis of systems of linear equations remain important problems in scientific computation for which new algorithms and software packages are continually being developed. In order to make reliable, reproducible quantitative assessments of the value of new algorithmic developments, it is useful to have a common collection of representative problems through which methods can be compared. For sparse matrices the Harwell-Boeing Sparse Matrix Collection has served this purpose for some time. However, one of the difficulties with such collections is that their size and diversity makes them unwieldy to manage and use effectively. Consequently, the Harwell-Boeing collection has not been used as much as it should, and new matrices have not been regularly added to the collection. Recent developments in Web technologies are opening up new possibilities for improving the access to and usability of test corpora of this type.

The Matrix Market [5] provides convenient Web access to a repository of test data for use in comparative studies of algorithms for numerical linear algebra. Matrices and related data from problems in linear systems, least squares, and eigenvalue calculations in a wide variety of scientific and engineering disciplines are provided. Tools for browsing through the collection or for searching for matrices with special properties are included. Additional background on the project can be found in [3]. In this paper, we discuss work on the user interface to Matrix Market.

Matrices in Matrix Market are gathered together into *sets*. Matrices in a set are related by application area or are contributed from a single source. Sets can be further grouped into *collections* managed by a single group, such as the Harwell-Boeing collection. Individual matrices may be stored explicitly as dense or sparse matrices, or may be made available via a code which generates them. For each matrix, we provide a summary page in HTML format outlining the properties of the matrix and displaying a graphical representation of its structure. Clicking on the GIF image providing the view of the nonzero structure retrieves another image at a finer level of detail. Similarly, we have developed an HTML page for each set, which gives its background (e.g., source and application area), references, as well as a thumbnail sketch of the nonzero

[5] `http://math.nist.gov/MatrixMarket/`

pattern of each matrix in the set. Clicking on the matrix identifier or thumbnail retrieves the home page for that particular matrix. A separate database contains all the information in a highly structured form, allowing us to manipulate the data in various ways. All the matrix and set HTML pages are automatically generated from this database. The database also supports both structured and free-text retrieval. The matrix collection may also be browsed by collection, application domain, or contributor.

Matrix Market provides a *Database Query* tool with an HTML form interface that allows the user to specify a large variety of attributes describing selection criteria for matrices. Lists allow selection based on predefined attributes in the database, such as linear algebra problem type, arithmetic field (real, complex, pattern, symmetry property, definiteness, type of nonzero structure, storage mode, and shape. Text fields allow the user to specify the minimum and maximum number of rows, columns, and nonzeros. Additional buttons allow the user to request that right-hand sides, exact solution, or initial vectors be made available. A text field allows the user to specify patterns to match in the database text descriptions of the matrices. Help text is provided for each of the available selection criteria. Only matrices that satisfy all of the requirements specified by the user are retrieved. Matches are returned on an HTML page organized by matrix set. Links to the home pages of individual matrices as well as the sets are provided.

We intend for the Matrix Market to provide a real marketplace for the exchange of test data. For this to occur there must be a convenient way for users to submit their own matrices to the collection. Thus, we have provided a set of HTML forms that allow users to describe their matrices and provide the URL that can be used to retrieve them. Submitting the form constructs a prototype database entry for the matrix and notifies the maintainers of the submission. It is still necessary to review contributions to determine if they satisfy our selection criteria before inclusion, however.

Many of the matrices found in Matrix Market were obtained by running application codes that generated them. In many cases these codes are parameterized so that a wide range of matrices can be produced. We plan to incorporate programs which generate matrices useful for testing linear algebra software into our collection. These may be made available either as links to downloadable codes or access to remote execution servers which exercise the generation software on demand. The latter method could provide access to proprietary matrix generators, with permission of the authors. One way to supply matrix generation software is to develop Java applets which present a class of matrices whose individual members can be computed on demand inside a Web browser. We are currently experimenting with such tools. This is a more scalable approach than providing a remote execution service, but presents some challenges in that there has yet been little experience in developing portable, reliable, and efficient floating-point applications in the Java environment.

A related service that also provides dynamically generated content is the source code request service for the NIST Sparse BLAS developed by Karin Remington and Roldan Pozo [6]. The Sparse BLAS are a set of computational kernels for use in building portable high performance software for numerical linear algebra. Because of the many operations and storage formats supported by the BLAS, would be some 1,300 possible routines, representing more than 100,000 lines of code, much more than the average user is interested in for any particular application. The source code generation service provides a form in which the user selects a particular matrix storage format, the BLAS operation, the type of scaling, the number of right hand sides, and special values for scalar factors. The service then generates a routine optimized for this particular set of criteria and returns it to the user's browser.

3 Inferno and ApproxWizard

Inferno(tm) is a new network operating system and programming environment for delivering content in a rich environment of heterogeneous networks, clients, and servers [7]. The Inferno system includes the Inferno kernel, the Limbo(tm) programming language, reference APIs that include interfaces for networking and graphics, network protocols, security and authentication, and various toolkits. Inferno was developed by members of the Computing Sciences Research Center of Bell Laboratories, the research arm of Lucent Technologies.

[6] `http://math.nist.gov/spblas/`

[7] `http://inferno.bell-labs.com/inferno/`

Although the focus of Inferno is interactive media, its portability across hardware and operating platforms, its relative simplicity, and its strength in distributed computing make it attractive for distributed scientific computing as well. Inferno can run either on bare hardware or on top of another operating system such as Windows95 or Unix. Programs for Inferno are written in the Limbo language and compiled to machine-independent object files for the *Dis* virtual machine, which is then implemented with runtime compilation for best performance. The floating point environment provided by Limbo includes tight rules on expression evaluation, binary/decimal conversion, exceptions and rounding, and an elementary function library.

Limbo may be viewed as alternative to Java, but while Java is just a programming language, Limbo is supported by the Inferno full network operating system which includes security and authentication, naming protocols, directory services, and network interfaces. Inferno will eventually support other programming languages such as Java. Since Inferno developers plan to support Java, Inferno is more a complement to Java than a competitor.

As an example of the use of Limbo, ApproxWizard is an applet, developed in Limbo, that helps users select an approximation code. Instead of doing a keyword or hierarchical search, the user provides sample data and an objective. The ApproxWizard applet interacts with the user by doing calculations, either on the client or remotedly on servers, on sample user data sets that reside on the client disk. The Wizard looks for problem features and tries different algorithms. The applet uses dynamic loading and unloading so that all of the Netlib approximation code collection is potentially part of one program.

4 NetSolve

An ongoing thread of research in scientific computing is the efficient solution of large problems. Various mechanisms have been developed to perform computations across diverse platforms. The most common mechanism involves software libraries. Unfortunately, the use of such libraries presents several difficulties. Some software libraries are highly optimized for only certain platforms and do not provide a convenient interface to other computer systems. Other libraries demand considerable programming effort from the user, who may not have the time to learn the required programming techniques. While a limited number of tools have been developed to alleviate these difficulties, such tools themselves are usually available only on a limited number of computer systems. MATLAB is an example of such a tool.

These considerations motivated the establishment of the NetSolve project. The NetSolve system, developed at the University of Tennessee, is a client-server application designed to solve computational science problems over a network [8]. A number of different interfaces have been developed to the NetSolve software so that users of C, Fortran, MATLAB, or the World Wide Web can easily use the NetSolve system. The underlying computational software can be any scientific package, thus helping to ensure good performance through choice of an appropriate package. Figure 1 shows the conceptual picture of the NetSolve system. In this figure, a NetSolve clients sends a request to the NetSolve agent. The agent chooses the "best" NetSolve resource according to the size and nature of the problem to be solved. Several instances of the NetSolve agent can exist on the network. Every host in the NetSolve system runs a NetSolve *computational server*, also called a *resource*. The NetSolve resources have access to scientific packages such as libraries or stand-alone software systems. More information about the NetSolve system may be found in [7]. In this paper, we describe recent work on the Web interfaces to Netsolve.

An HTML forms interface is available from the NetSolve home page that allows a user to contact a NetSolve agent to obtain information about software and hardware resources for an instance of the NetSolve system. The user enters the name of a host running a NetSolve agent in either the Software Resources or the Hardware Resources form. The Software Resources form returns a list of all the problems that can be solved on the NetSolve system. The list contains all the information about what type of input data is needed and what type of output will be produced. The Hardware Resources form returns a list of all the agent or computational servers in a NetSolve system. The list contains information such as Internet addresses and host and server status. Although the HTML forms cannot be used to submit problems for solution, they can be used to obtain information prior to using one of the programming interfaces to NetSolve.

[8] `http://www.cs.utk.edu/časanova/NetSolve/`

The Java interface to NetSolve provides a user-friendly graphical tool for accessing the NetSolve system. Because the Java interface should be runnable from many Web browsers, it also provides the opportunity to solve problems without downloading or compiling any source code. However, the current Web browsers that support Java impose restrictions on the networking capabilities of applets. At this time, it appears to be impossible to open sockets to remote hosts, making the NetSolve Java interface from a Web browser. Future version of Web browsers will undoubtedly alleviate this problem. In the meantime, the Java NetSolve client must be run as a standalone Java application.

Let us now assume that the user has started the Java interface, either as an applet (via the Web) or as a stand-alone application. Figure 2 shows the initial screen, which consists of several components:

- Agent Selection Box
- Problem List
- Problem Description Box
- Input List
- Input Description Box
- Output List
- Output Description Box

To contact an agent, the user can enter the hostname in the *Agent Selection Box* and then click on the "Contact/Update" button. In some cases, the user may have already contacted an agent, but just wants to update the list of problems. If so, clicking on the "Contact/Update" button without changing the text in the *Agent Selection Box* will reload the problem list. Once the list of available problems has been loaded it is then displayed in the *Problem List*, located in the upper left region of the interface.

To find out more about any problem listed, the user may click on that problem and view pertinent information displayed in the *Problem Description Box*, the *Input List*, and the *Output List*. The *Problem Description Box*, located in the lower left region of the interface, contains a short description of the selected problem. The *Input List* contains a list of the input objects required to solve the selected problem. Similarly, the *Output List* contains a list of the output objects that are returned by the server. When the user clicks on any item in the *Input List*, the interface updates the *Input Description Box* with text describing the selected input object. Likewise, clicking on any item in the *Output List* updates the *Output Description Box* with text describing the selected output object.

4.1 Solving a Problem

To solve an instance of some problem, the user must first select a problem from the *Problem List* and then click on the "Solve" button. A new window will appear allowing the user to input data for each input object required by the problem. Figure 3 shows the *Data Input Window*, which consists of the following components:

- Input List
- Input Description Box
- Filename (or URL) Selection Box
- Data Input Box

The *Input List* contains a list of the input objects for which the user must supply data. The *Input Description Box* contains text describing the selected input object (this text is the same as the text displayed in the *Input Description Box* of the initial screen).

For each input object, the user may choose to enter the data manually into the *Data Input Box* or to specify the name of a file containing the data in the *Filename Selection Box*. Next to the *Filename Selection Box* is a "Browse" button which allows choosing the file using a graphical file browser. Those users accessing the NetSolveClient via a Web browser will have a *URL Selection Box* (instead of a *File Selection Box*) in which they may type in the URL for their data file. This allows NetSolve to access the user's local data files over the network. Just above the *Data Input Box* is a "Sample Data" button which fills the box with some numbers appropriate to the type of the input object (for example, if the input object is a vector of integers, clicking on the "Sample Data" button will generate a vector of integers). Note that even though the interface allows having text in both selection boxes simultaneously, only one box may be "active" at any time and anything in the "inactive" box will be ignored.

The title bar of the *Data Input Window* contains some noteworthy information: the name of the problem, and a *Request Number*. The problem name listed on the title bar is the same name from the initial screen, minus the path. For example, if the full name as shown on the initial screen is `/Blah/blah/prob`, then the name on the title bar is `prob`. The *Request Number* is a number which uniquely identifies each *Data Input Window* so that the user may easily relate the *Output Windows* (see Section 4.2) to the *Input Windows* from which they originated.

Once all inputs have been fully specified, click on the "Compute" button, located in the lower left region of the *Data Input Window*. If there are any errors in the data and/or files, an informational window will appear describing the nature of the errors and for which input object(s) the errors apply. All errors must be corrected before the data may be sent.

If the data and/or files specified are acceptable, the values are sent to a computational server which performs the computations and returns the output objects.

4.2 Viewing the Results

Once the computational server sends back the results, a new window appears allowing the user to browse the results. Figure 4 shows the *Output Window*, which consists of the following components:

- Output List
- Output Description Box
- Data Box

The *Output Window* is arranged like the *Data Input Window*, with a list of objects on the left, a data box on the right, and a description box on the bottom. When the user clicks on any item in the *Output List*, the *Output Description Box* is updated with text describing that object and the *Data Box* is updated with the results of the computation. Above the *Data Box* is a "Save" button which allows users of the stand-alone application to save the text in the *Data Box* to a file. Note that the data saved is that for the selected output object only, not all output objects.

Like the *Data Input Window*, the title bar of the *Output Window* also contains the problem name and a *Request Number*. However, the *Request Number* is slightly different in this window. It consists of two numbers separated by a "." (period). The first number is the *Request Number* from the *Data Input Window* from which this output originated. The second number uniquely identifies this window so that it can be distinguished from other *Output Windows*. Here's an example of how the numbers are assigned: the user chooses a problem, "ddot" perhaps, on the initial screen and clicks "Solve". The *Data Input Window* corresponding to that problem will have *Request Number* "1". Then the user chooses a different problem, "matmul" perhaps, and clicks "Solve". The *Request Number* corresponding to that problem will be "2". The number is incremented each time a new input window is opened. The user enters data into the "matmul" window and clicks "Compute" three times to solve three instances of that problem. Soon three output windows will appear with *Request Numbers* “2.1”, “2.2”, and “2.3” corresponding to the first, second, and third instance of the problem, respectively.

5 Java Linpack

Although Java technology opens a wealth of opportunities for distributed scientific computing, the performance of Java needs to be evaluated to determine its suitability for numerical applications. A Java version of the Linpack Benchmark is available from Netlib [9] The Linpack Benchmark is a numerically intensive test that has been used for years to measure the floating point performance of computers. The Java applet allows users to submit results for Java Linpack by filling in a form with information about the operating system and CPU of his local machine, as well as more detailed information about memory and processor speed, and whether or not the user's browser uses just-in-time (JIT) Java compilation. The user then presses a button of run the benchmark on his machine. The user's timings are then to the Java Linpack developers by email, who then update the timings and the graphical display that appear on the Java Linpack home page.

The test results are more a reflection of the state of Java systems than of the floating point performance of the underlying processors. Some Java systems do line by line interpretation and others perform “just in time” (JIT) compilation. As can be seen from the results, the JIT systems perform better, perhaps by an order of magnitude. The Linpack Java Benchmark allows the scientific user community to track the numerical performance of Java implementations over time over a range of architectures.

6 Future Work

One objective of our research is to make the process of networked scientific computing seamless. This requires that there be an operational transparency, at the level of the user, to the existence of the network. The user's view should be that of accessing a single computational resource. To this end, we seek to create *Problem Solving Environments* (PSEs) that are network and evolution aware, and abstract network and system details from the user, thereby making the system effectively transparent. We plan to evolve GAMS into a PSE that enables network-based computing in the scientific software area. An important task of the PSE is to accept some “high level” description of the problem from the user, and then, taking the user's computing environment and other constraints into consideration, automatically select appropriate computational resources (hardware, software) to solve it. Clearly this task requires the use of “intelligent” techniques, with knowledge about the problem domain and reasoning strategies that enable the system to locate appropriate software components. The PSE also needs to be aware of the underlying network and its capabilities and how to integrate various software components across the network.

We plan to do further benchmarking studies and comparisons of the Java and Limbo programming languages to determine their suitability for distributed applications. In addition, because of the security risks involved, we are working with other researchers in the repository and agent technology communities to define requirements for safe execution environments for agent and applet programs. An execution environment provides program interpretation and run-time support as well as relocation and communication services. However, the execution environment must also be secure to ensure that code from untrusted sources does not harm the host system, gain unauthorized access to files, or usurp resources. After determining the requirements for such an environment, we plan to implement a facility for remote execution of user code in the Netsolve system.

[9] `http://www.netlib.org/benchmark/linpackjava/`

References

[1] R. F. Boisvert. The architecture of an intelligent virtual mathematical software repository system. *Mathematics and Computers in Simulation*, 36:269–279, 1994.

[2] R. F. Boisvert, S. E. Howe, and D. K. Kahaner. The Guide to Available Mathematical Software problem classification system. *Comm. Stat. - Simul. Comp.*, 20(4):811–842, 1991.

[3] R. F. Boisvert, R. Pozo, K. Remington, R. F. Barrett, and J. J. Dongarra. Matrix Market: a Web resource for test matrix collections. In *The Quality of Numerical Software: Assessment and Enhancement*, pages 125–137. Chapman & Hall, 1997.

[4] R. F. Boisvert, J. L. Springmann, and M. L. Strawbridge. The GAMS virtual software repository. In *Proceedings of the Thirtieth Semi-Annual Meeting*, pages 68–72, Gaithersburg, MD, September 1992. Cray User Group, Five Point Editorial Services.

[5] S. Browne, J. Dongarra, S. Green, K. Moore, T. Rowan, R. Wade, G. Fox, K. Hawick, K. Kennedy, J. Pool, R. Stevens, R. Olsen, and T. Disz. The National HPCC Software Exchange. *IEEE Computational Science and Engineering*, 2(2):62–69, 1995.

[6] S. Browne, J. Dongarra, E. Grosse, and T. Rowan. The netlib mathematical software repository. *D-Lib Magazine*, Sept. 1995. Accessible at http://www.dlib.org/.

[7] H. Casanova and J. Dongarra. Netsolve: A network server for solving computational science problems. In *Supercomputing '96*, Pittsburgh, PA, Nov. 1996.

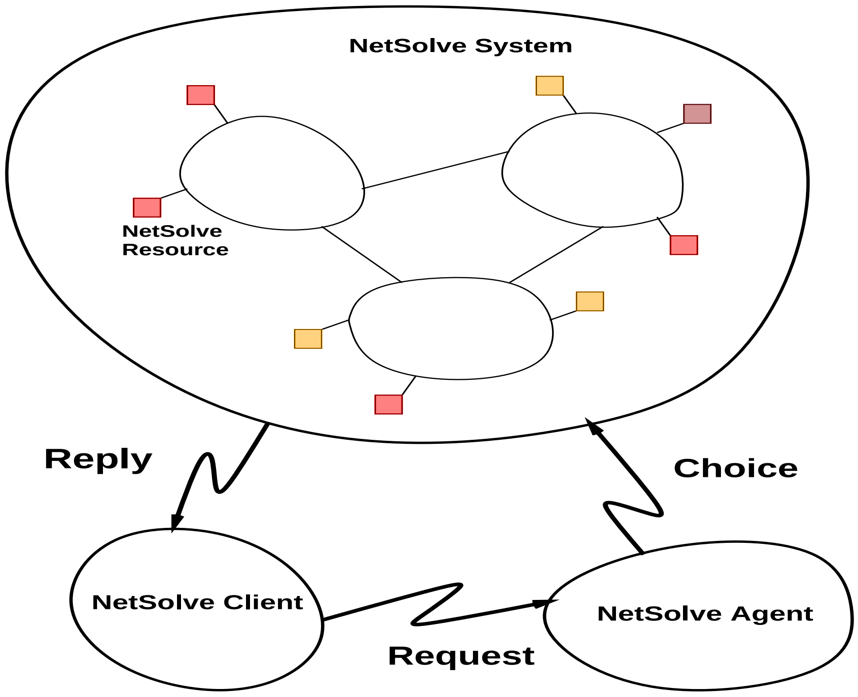
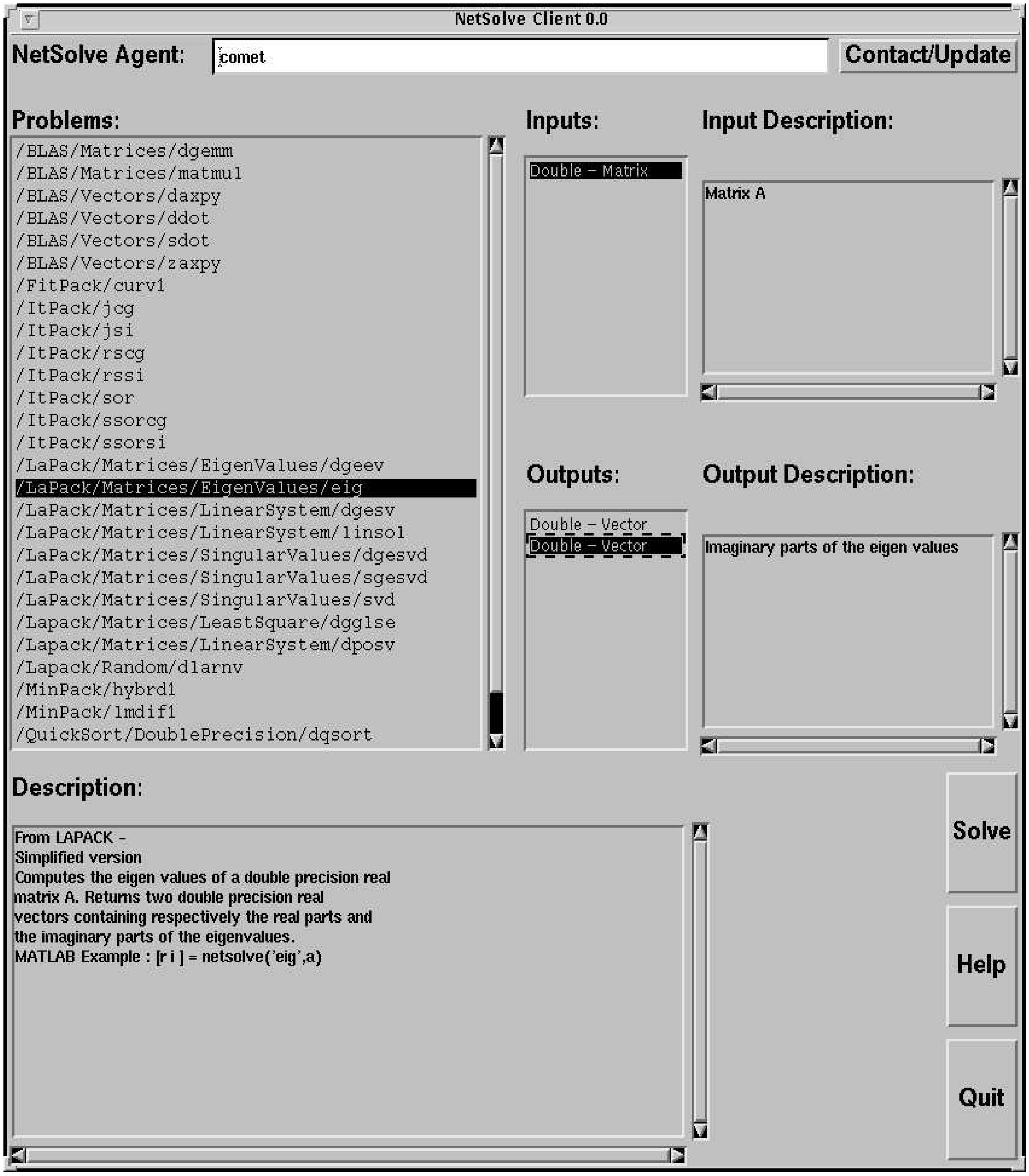


Figure 1: The NetSolve System



NetSolve Agent: comet

Problems:

```
/BLAS/Matrices/dgemm
/BLAS/Matrices/matmul
/BLAS/Vectors/daxpy
/BLAS/Vectors/ddot
/BLAS/Vectors/sdot
/BLAS/Vectors/zaxpy
/FitPack/curv1
/ItPack/jcg
/ItPack/jsi
/ItPack/rscg
/ItPack/rssi
/ItPack/sor
/ItPack/ssorcg
/ItPack/ssorsi
/LaPack/Matrices/EigenValues/dgeev
/LaPack/Matrices/EigenValues/eig
/LaPack/Matrices/LinearSystem/dgesv
/LaPack/Matrices/LinearSystem/linsol
/LaPack/Matrices/SingularValues/dgesvd
/LaPack/Matrices/SingularValues/sgesvd
/LaPack/Matrices/SingularValues/svd
/Lapack/Matrices/LeastSquare/dgglse
/Lapack/Matrices/LinearSystem/dposv
/Lapack/Random/dlarnv
/MinPack/hybrd1
/MinPack/lmdif1
/QuickSort/DoublePrecision/dqsort
```

Inputs: Double – Matrix

Input Description: Matrix A

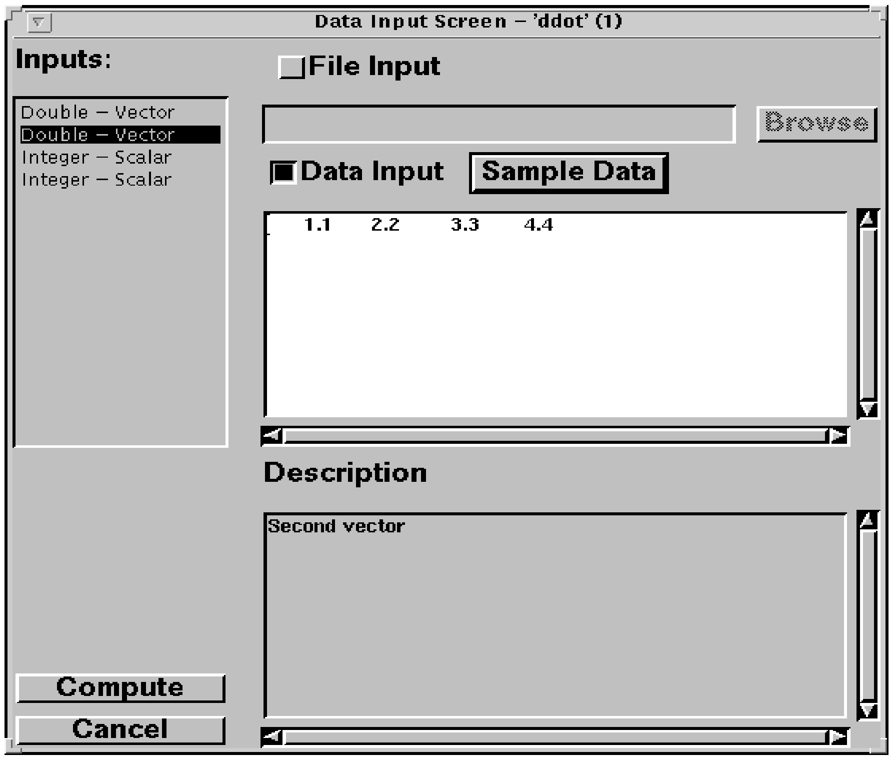
Outputs: Double – Vector, Double – Vector

Output Description: Imaginary parts of the eigen values

Description:

From LAPACK -
Simplified version
Computes the eigen values of a double precision real
matrix A. Returns two double precision real
vectors containing respectively the real parts and
the imaginary parts of the eigenvalues.
MATLAB Example : [r i] = netsolve('eig',a)

Figure 2: The Initial Screen



Double – Vector
Double – Vector
Integer – Scalar
Integer – Scalar

1.1 2.2 3.3 4.4

Second vector

Figure 3: The Input Screen

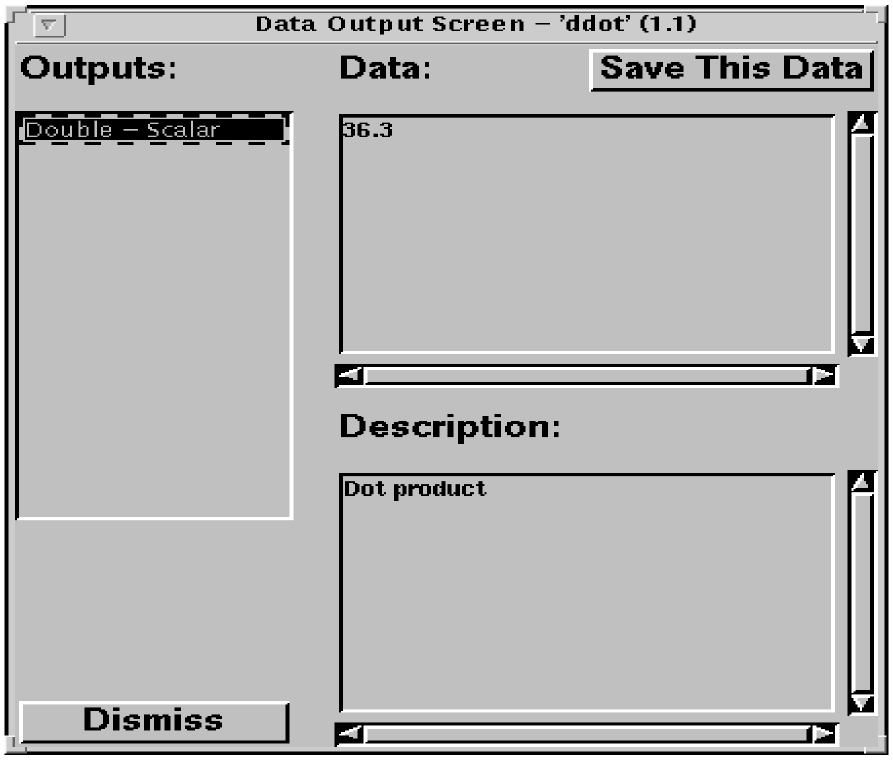


Figure 4: The Output Screen