The Architecture of an Intelligent Virtual Mathematical Software Repository System*

Ronald F. Boisvert
Computing and Applied Mathematics Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899 USA

Abstract

Much reusable software is available for solving routine mathematical and statistical problems. Unfortunately, locating this software is often quite difficult in current distributed computing environments. The Guide to Available Mathematical Software (GAMS) virtual software repository seeks to remedy this by providing users with convenient access to thousands of software modules physically distributed among several Internet repositories, including *netlib*. In this paper I will illustrate the use of GAMS, describe its implementation, and outline plans for the incorporation of expert advisory capabilities.

1 Introduction

Users of modern scientific computing facilities have substantial collections of mathematical and statistical problem-solving tools at their disposal. These may be installed for use on a local system or accessible from an external repository via anonymous FTP, a network server, or electronic mail. The average user with a particular mathematical problem to solve has a difficult time dealing with this bounty. Even if the user knows where within the labyrinthine computer network to look for

---

*This article appeared in *Mathematics and Computers in Simulation*, volume 36 (1994), pp. 269-279. References to commercial products in this paper do not imply recommendation or endorsement by NIST.

potential software, it is often a tedious and error-prone process to determine the subset to the problem at hand. Each software repository may have a different access mechanism, and primitive directory services at best. In addition, many potential users lack the expertise to translate their own sometimes vague problem descriptions into the precise jargon associated with mathematically oriented software abstracts. This terminology barrier also makes it difficult for users to discriminate among software for solving similar problems. Users need an easily accessible central information resource with effective directory services supported by readily available expert consultants.

The Guide to Available Mathematical Software (GAMS) is a centralized mathematical software resource maintained for scientists at the National Institute of Standards and Technology (NIST). GAMS contains information about thousands of mathematical and statistical software objects available from several physically distributed software repositories, including *netlib* [3]. The system supports search for software by problem, keyword, or name. Users can retrieve items such as abstracts, documentation, examples and source code using the system. Both command line and graphical user interfaces are available.

GAMS performs the functions of a search and retrieval system for a large central software repository. However, instead of maintaining the cataloged software itself, it provides a cross-index of and central access to multiple repositories managed by others. We use the term *virtual software repository* to describe such a system.

In this paper I outline the evolving architecture for the GAMS system. Both its current status as well as plans for the incorporation of automated domain consultants will be described.

2 The GAMS Software Repository Model

A virtual software repository is a collection of modules. A *module* represents a reusable software part, such as a Fortran subroutine, a C procedure, a self-contained program, or a command in a self-contained system. Modules may have alternate *versions*, identical except for the precision with which the computation is done. In addition, modules are characterized by a set of predefined *attributes*, such as precision and language, which can be used to establish user-specified *filters* for module selection. *Packages* are collections of modules that are related in some way; they may solve related problems or be distributed and maintained as a unit.

Figure 1: Topmost level of the taxonomy, with problem F expanded.

A.	Arithmetic, Error Analysis	G.	Optimization
B.	Number Theory	H.	Differentiation, Integration
C.	Elementary and Special Functions	I.	Differential and Integral Equations
D.	Linear Algebra	J.	Integral Transforms
E.	Interpolation	K.	Approximation
		L.	Statistics, probability
F.	Solution of Nonlinear Equations	M.	Simulation, stochastic modeling
F1.	Single equation	N.	Data handling
F1a.	Polynomial	O.	Symbolic computation
F1a1.	Real coefficients	P.	Computational geometry
F1a2.	Complex coefficients	Q.	Graphics
F1b.	Nonpolynomial	R.	Service routines
F2.	System of equations	S.	Software development tools
F3.	Service routines	Z.	Other

Associated with each module and package is a set of *components*. These are retrievable objects such as abstracts, documentation, source code, examples, and tests. Component types are not restricted to a predefined set, so that retrievable objects unique to a particular module or package are easily accommodated. Module and package components physically reside at one or more *repositories*. Each repository may have a different access mechanism, such as electronic mail, anonymous FTP, or a repository server.

An important feature of our model is a *taxonomy* of mathematical and statistical problems which is used to classify software modules. The GAMS problem classification system [2] is a 736-node tree-structured taxonomy extending to seven levels (see Figure 1). Each child node in the tree denotes a more specific instance of the mathematical problem represented by its parent. This taxonomy is now in widespread use, having been adopted by many library developers and scientific computing centers (see, e.g., [4] [7]). In addition, a two-level *index* of related terms is maintained. This index is mapped to the problem taxonomy, allowing one to translate alternate terminology into that used by the taxonomy.

Finally, the GAMS repository model supports a set of *services* which provide access to GAMS. These include operations such as: retrieval of information about module attributes; retrieval of information about problem classes; search for modules by class (subject to module attribute filters); search for modules by package or module name; module and package abstract retrieval; module and package component retrieval; and index search.

3 System Architecture

The architectural components of the GAMS system are (see Figure 2) :

GAMS Database. A relational database containing all information necessary to support the GAMS repository model.

Relational Database Interface. A set of procedure calls that provide standard relational database management services.

GAMS Interface. A set of procedure calls that implement the GAMS repository services.

GAMS Server. A network server implementing the GAMS repository services using a simple ASCII protocol.

GAMS Client Interface. A set of procedure calls that implement the GAMS repository services. These are functionally equivalent to the GAMS interface, except that they access the GAMS server.

User Interfaces. These manage all interactions with the user. Two versions have been developed: a command line interface (*gams*) and a graphical user interface (*xgams*).

All information about repository sites, problem classes, packages, modules, and components is contained in the GAMS database, with retrieval mechanisms provided by the relational database interface. At present, this database contains information describing 8545 modules in 66 packages at 4 repository sites (three at NIST and *netlib*). The database occupies about 10 megabytes on a Sun-4 system. It is currently implemented using the RIM Database Management System, a Fortran-based system available in both public-domain and commercially supported versions.

The GAMS interfaces and the GAMS server each implement the GAMS services. The GAMS interface provides these services through a set of procedure calls (both C and Fortran implementations are currently supported). These procedures obtain necessary data from the GAMS database using relational database interface routines.

The GAMS interface constructs module and package abstracts on demand based on information in the GAMS database. Other module and package components are not explicitly maintained within the GAMS system; to retrieve them the GAMS interface must rely on communication with external repositories. The database contains information on

Figure 2: The architecture of the GAMS system.

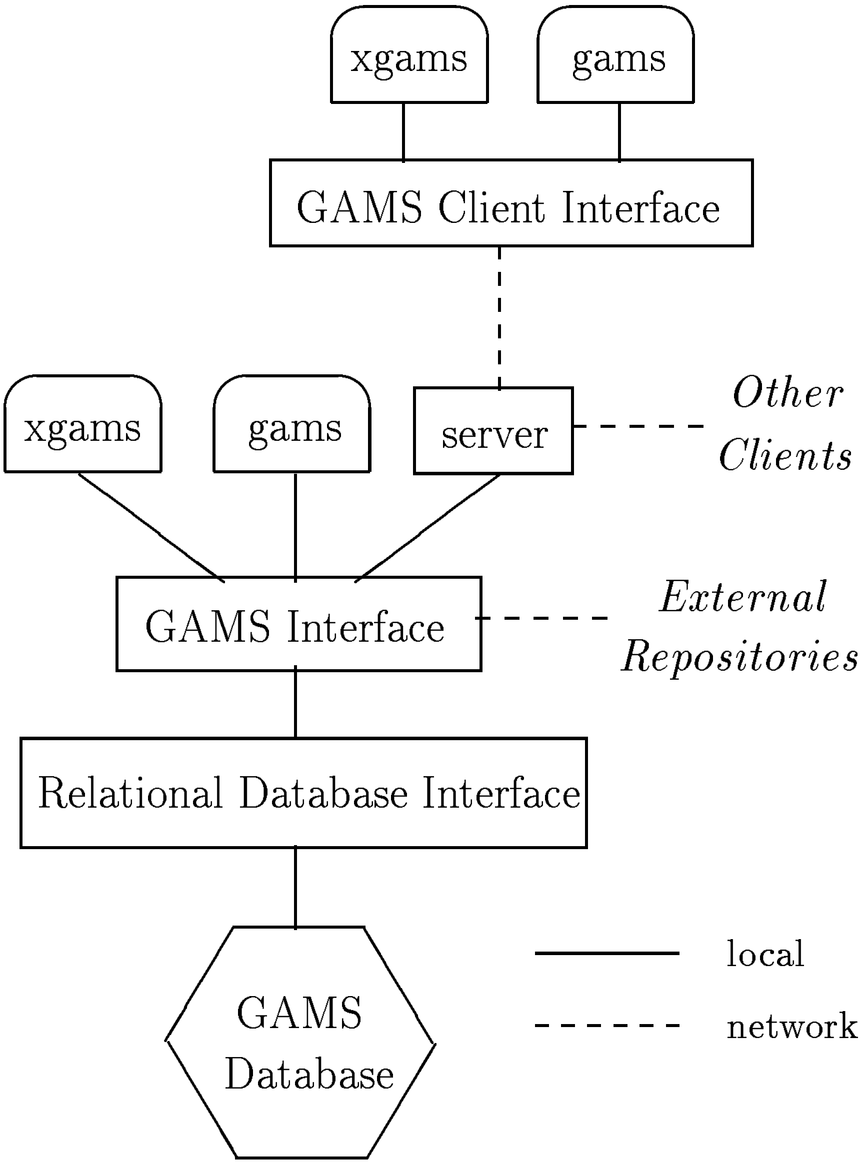


Figure 3: Example of GAMS server interaction. Lines representing server requests start with >, those representing responses start with <.

```
> telnet doc.cam.nist.gov 5555
Trying 129.6.80.2 ...
Connected to doc.
Escape character is '^]'.
> boisvert@nist.gov
> gams-serve
> class f
< F
< Solution of nonlinear equations
< 0 modules
< 3 subclasses
< F1
< Single equation
< F2
< System of equations
< F3
< Service routines (e.g., check user-supplied derivatives)
Connection closed by foreign host.
```

how to contact each repository and how to request components available there. Most types of communications mechanisms, such as network servers, anonymous FTP, and electronic mail, can be accommodated. For example, software from *netlib* is accessed in real-time using a local client program which makes a socket connection to a server at Oak Ridge National Laboratories developed for the *xnetlib* system.

Two distinct user interfaces to GAMS have been developed: a graphical user interface and a command line interface. Each provides user-friendly access to the GAMS services. The operation of the graphical user interface is described in the next section. The command line interface provides most of the services available in the graphical interface, but accepts input from the command line and displays output in ASCII.

The GAMS server provides network access to the GAMS services. It is a Unix daemon that responds to requests made via TCP socket connections using a simple ASCII protocol. The GAMS server is based on the server developed for the *xnetlib* project. The server is stateless, i.e., each request is independent of all other requests, and the server closes the connection as soon as the response is sent. The server operation is illustrated in Figure 3 where we simulate the interaction between a client program and the server using telnet. Here the client requests information about the GAMS problem class with tag F.

The GAMS client interface is identical to the GAMS interface except that it obtains GAMS services by making network connections to the GAMS server. When the GAMS user interface programs are compiled with this version of the library they become GAMS network clients.

4 The X11 Graphical User Interface

In this section I describe *xgams*, a graphical user interface to GAMS. It is based on the portable, network-transparent X Window System [9] and is implemented using the Motif widget set [8]. *xgams* has been successfully run on Sun-4 workstations using the twm, mwm and olwm window managers. It has also been run remotely using window servers on Sun-4, Hewlett-Packard and Silicon Graphics workstations.

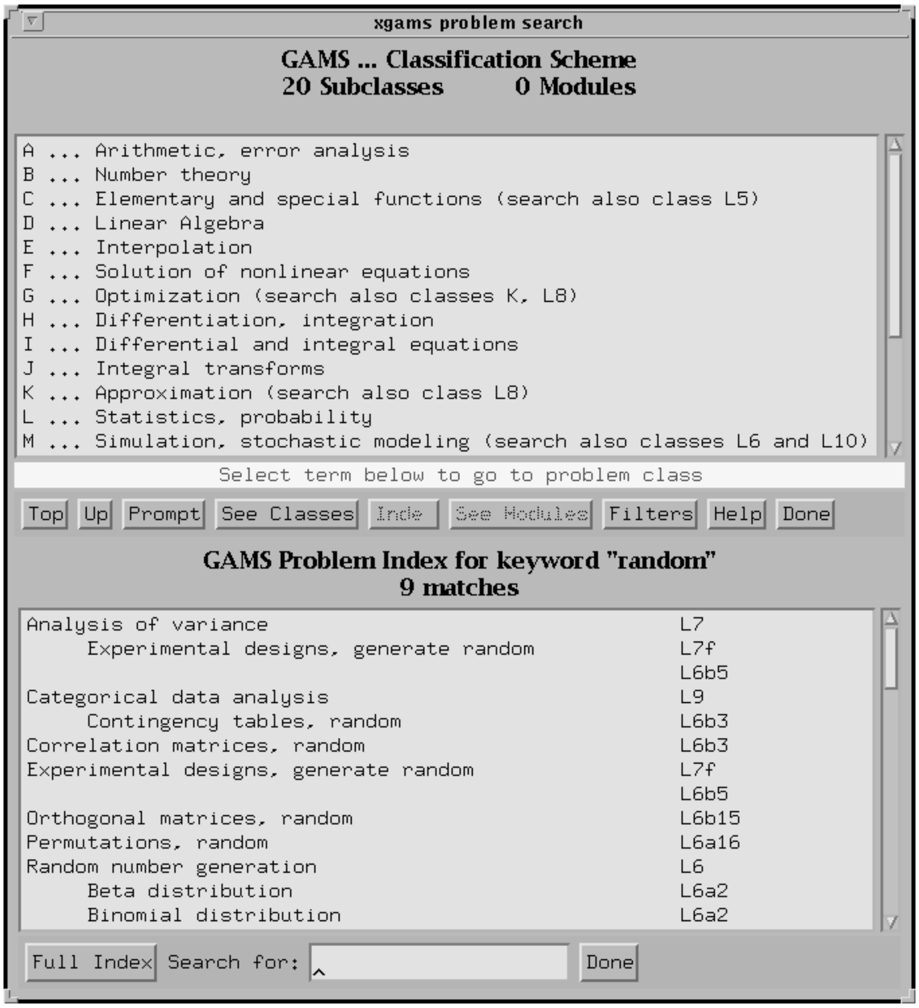
Each window in *xgams* contains the following components: a heading, one or more subwindows with scrollable text, a prompt line, and a row of buttons. The prompt line is context sensitive. When the cursor is moved over a button its purpose is displayed on the prompt line; otherwise, a state-dependent suggestion of what to do next is displayed. A Help button is available in each *xgams* window; pressing it exposes another window in which the user can view help text on various topics. The main *xgams* window provides an introduction to the system and allows one to select a basic search mechanism: search by problem or search by name.

When the Problem button is pressed in the main window the *problem search window* is displayed; see Figure 4. This is used to traverse the tree-structured problem taxonomy. At the top of the window is a description of the current problem class; its subclasses are listed in the scrollable text subwindow below. Users can move to a subclass by clicking on it with the mouse. Buttons allow one to move to the top of the tree (Top), to the parent class (Up), or directly to a class given its tag (Prompt).

Two additional navigation mechanisms are available in this window. The See Classes button exposes a subwindow containing the entire taxonomy in outline form; clicking on a class repositions the problem search window there. The Index button exposes a subwindow that allows the user to enter a keyword or phrase. Once entered, an index search ensues, with relevant problem classes displayed (see Figure 4). Clicking on these also repositions the problem search window.

The See Modules button exposes the *module window*, which allows one to view information about modules in the current class (see Figure 5). Clicking on a module or package name here causes an abstract de-

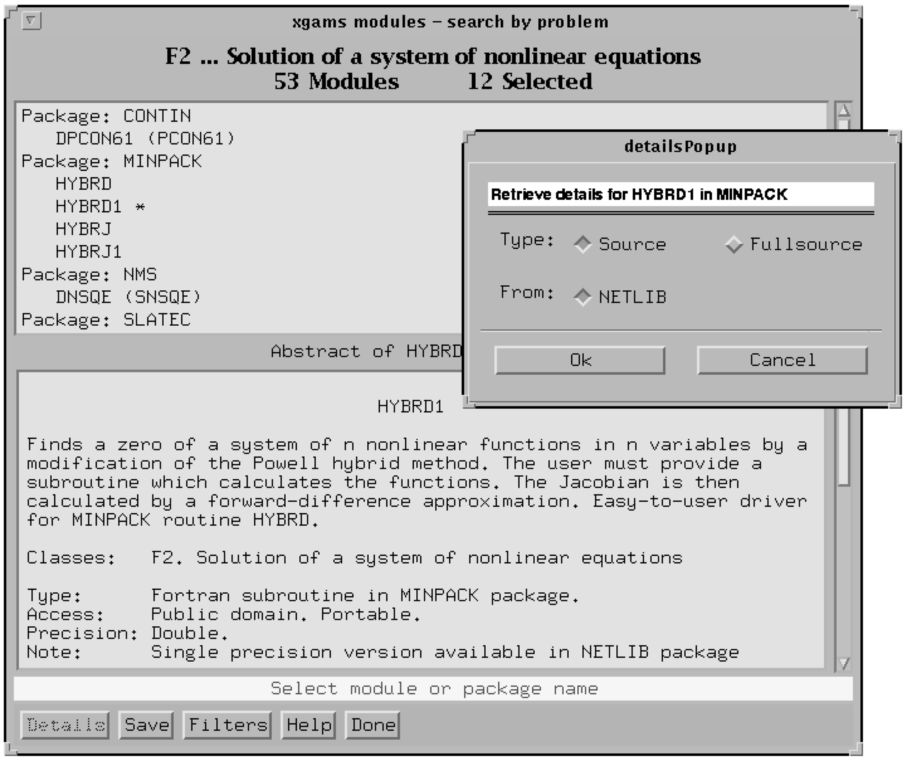
Figure 4: The *xgams* problem window.



scribing that item to be displayed in the text subwindow. Components of the currently displayed module can be retrieved by pressing the Details button. First, a popup appears which names the available components; when the user selects the desired component it is retrieved and displayed in the text subwindow. The Save button allows the user to obtain a permanent copy of the contents of the text subwindow — either in a local file or forwarded by electronic mail.

There are often many modules with the same problem classification.

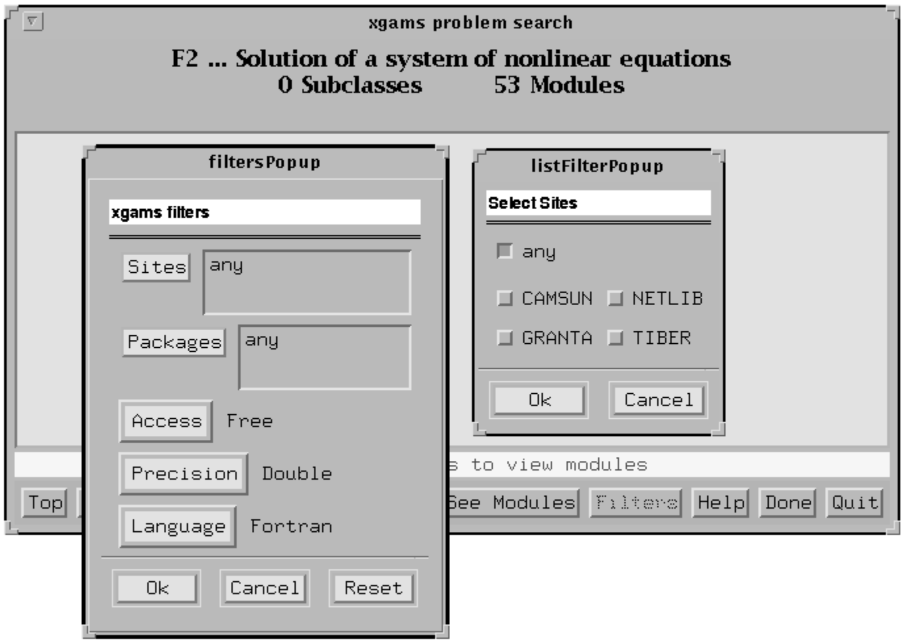
Figure 5: The *xgams* modules window.



Users can reduce the number displayed using the Filters button in either the problem search window or the module window (see Figure 6). The former is used to set global preferences while the latter is used to change preferences in the current module window. Only modules satisfying these preferences are selected for display. In Figure 5, for example, only public-domain, double precision, Fortran modules are displayed.

The second principal search mechanism in *xgams* is search by name, which is activated by pressing the Name button in the main window. This button causes the the *name search window* to be displayed, where one can enter a module name and/or select a package name. Pressing See Modules then brings up a module window with the selected modules. This facility allows one to find all modules with a given name, all modules in a given package, or a particular module in a given package.

Figure 6: The *xgams* filters facility.



5 Expert Extensions

The GAMS problem classification system provides a rudimentary means of codifying mathematical knowledge necessary for software selection. However, it is not detailed enough to deal with situations in which many similar software modules are available for a particular problem. Our goal is to build an automated advisory capability with the performance level of mathematical software generalist, such as someone providing computer user support at a large scientific laboratory. In particular, we are not attempting to provide the expertise of a numerical analyst specializing in any particular subfield. Advisory systems of the latter type require much deeper domain knowledge; see [5] and [6] for prototypes in the area of ordinary differential equations.

In order to further discriminate between modules, an advisory system must have information about each module's distinguishing characteristics. These are necessarily problem-dependent, and can only be ascertained after a detailed study of the software modules in a given class. In this section I provide an example of how one might do this.

Table 1: Some Extended Data for Problem E1a

Feature	Modules A	B	C	D	E	User
Class : additional data						
function bounds	-1	-1	-1	-1	-1	0
derivative values	-1	-1	-1	1	1	-1
derivative bounds	-1	-1	-1	-1	-1	0
Class : visual property						
preserve convexity	-1	-1	-1	-1	1	0
preserve monotonicity	-1	1	-1	-1	1	1
Class : polynomial degree						
quadratic	0	-1	-1	-1	1	0
cubic	0	1	1	-1	-1	0
quintic	0	-1	-1	1	-1	0
user-selected	1	-1	-1	-1	-1	0
Class : end condition						
natural	-1	-1	-1	1	-1	0
user-supplies derivatives	-1	0	-1	-1	-1	-1
software estimates derivatives	-1	0	-1	-1	-1	0
extra continuity	1	0	-1	-1	-1	0
periodic	-1	-1	1	-1	-1	-1
Class : knots						
supplied by user	1	-1	-1	-1	-1	0

For each problem class we admit a set of features partitioned into a small set of feature classes. Next we encode information about how each feature applies to each module in the problem class. In particular, for each (module, feature) pair we assign an integer as follows: 1 (always present), 0 (optionally present), -1 (not present). Finally, we maintain a vector that represents the user's opinion of each of the features. The possible values are: 1 (feature is required), 0 (no opinion), -1 (feature is forbidden). Table 1 illustrates a subset of features for the problem class E1a : One-dimensional interpolation by piecewise polynomial functions. Also included are encoded representations of several modules, and a user's opinion vector.

This information may be used to reduce the number of module alternatives presented to the user. In Table 2 I present a truth table for evaluating the user's opinion of a particular feature. A rejection of any feature is ground for rejecting the module. Note that we cannot immediately reject a module whose optional feature is forbidden by the user. In this case there may be acceptable alternatives, and we reject only when *all* optional features of a given feature class are forbidden by the user. In the example of Table 1 the user is seeking a monotonicity-preserving

Table 2: Truth Table for Evaluating User Opinions

		User 1	User 0	User -1
Module	1	Accept	Accept	Reject
Module	0	Accept	Accept	Conditional
Module	-1	Reject	Accept	Accept

interpolant and cannot provide derivatives. Modules A, C and D do not preserve monotonicity and module E requires derivatives. Only module B satisfies all user requirements.

More than one module may remain applicable, even after all user opinions have been ascertained. In this case, information implied by the user vector can be used to order the remaining modules so that those most closely matching the user profile are listed first. For each module we count the number of features required by the module (extra features) and the number of optional module features (extra options) about which the user has no opinion. Modules are then presented in increasing order of extra features, with ties listed in increasing order of extra options.

Domain advisory capabilities such as these can be easily accommodated in the GAMS architecture as new services provided by the GAMS interface. An advantage to this approach is that small domain-dependent knowledge bases may be constructed incrementally; more global schemes would be much more difficult to construct and maintain.

Designing an intuitive and robust user interface to constructing the user's opinion vector is nontrivial. I briefly describe two alternatives : dialog and dynamic checklist. The dialog simulates the question-and-answer interaction between human consultant and client. At each stage of the dialog, the consultant selects a new feature class and presents to the user the various alternatives possessed by remaining modules. The user offers an opinion on zero or more of these, after which rejected modules are pruned from the list and the process is repeated. Several difficulties remain. The user may inadvertently eliminate all modules from consideration, necessitating a backtracking procedure. Also, the outcome may be influenced by the ordering of the questions. A simple user interface of this type is illustrated in [1].

The dynamic checklist is appropriate for use in a graphical user interface. Here one initially presents all features enumerated as in the first column of Table 1. Adjacent to each feature is an interface construct (a widget) allowing the user to choose yes, no or maybe; all are initialized to maybe. Also displayed is a count of the number of applicable modules. Users interact with the various choice widgets in any order. When a choice is modified two things happen: the module count is updated and features absent from remaining modules are removed from the checklist. The latter can be done by shading the feature text and rendering the widget inoperable. Not only is this simpler than dialogs, but it implicitly supports backtracking. Another difference is that here the user initiates all interaction, while the consultant is the active agent in the dialog. The latter may have some appeal for inexperienced users. In either case facilities must also be available to provide the user with further explanation of the various features.

6 Summary

I have described a model for a virtual mathematical software repository and the architecture used in its implementation. Such a system is now operational at NIST. I then outlined a set of new data and related selection heuristics that can be used to augment the advisory capabilities of such a system. Further investigation of the feasibility of this approach still needs to be undertaken.

The GAMS user interfaces are available for anonymous use on the Internet, and external use is encouraged on an experimental basis in order to obtain additional user feedback. To try the system, telnet to `doc.cam.nist.gov` (129.6.80.116) and present username **gams** (command line interface) or **xgams** (X11 interface).

Acknowledgements

Several NIST staff have contributed to the emergence of the current GAMS architecture. The X11 interface was developed by Michael Strawbridge; Jeanne Springmann has provided ongoing programming and database management support; and Marjorie McClain has helped in integrating *netlib* data into GAMS. Bruce Miller provided helpful discussions on dynamic checklists. Thanks also go to Reed Wade of the University of Tennessee at Knoxville for the use of the *xnetlib* server, and to Barry Jacobs of NASA Goddard Space Flight Center for help in formulating the GAMS repository model.

References

[1] R. F. Boisvert. Toward and intelligent system for mathematical software selection. In P. W. Gaffney and E. N. Houstis, editors, *Programming Environments for High-Level Scientific Problem Solving*, pages 79–92. North-Holland, Amsterdam, 1992.

[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] J. J. Dongarra and E. Grosse. Distribution of mathematical software via electonic mail. *Comm. ACM*, 30:403–407, 1987.

[4] IMSL Inc., Houston, TX. *MATH/LIBRARY User's Manual, FORTRAN Subroutines for Mathematical Applications*, September 1991. Version 2.0.

[5] M. S. Kamel, K. S. Ma, and W. H. Enright. ODEXPERT: An expert system to select numerical solvers for initial value ode systems. *ACM Trans. Math. Softw.*, 19:44–62, 1993.

[6] M. Lucks and I. Gladwell. Automated selection of mathematical software. *ACM Trans. Math. Softw.*, 18:11–34, 1992.

[7] The Numerical Algorithms Group, Ltd., Oxford, U.K. *The NAG Fortran Library Manual*, June 1991. Mark 15.

[8] *OSF/Motif Programmer's Reference.* Prentice-Hall, Englewood Cliffs, NJ, 1991. Open Software Foundation, Revision 1.1.

[9] V. Quercia and T. O'Reilly. *The X Window System User's Guide*, volume 3 of *X Window System Series.* O'Reilly & Associates, Inc., Sebastopol, CA, 1990.