SOFTWARE COMPONENTS FOR APPLICATION DEVELOPMENT

Arnaud Desitter, Antoine Le Hyaric, Geoff Morgan,
Gareth Shaw, Anne Trefethen
The Numerical Algorithms Group, Ltd
Oxford, UK

Abstract In this paper we discuss the needs of the application developer in terms of the components and tools required within an application development environment. We use, as examples, three European projects in which NAG has been a collaborator.

Keywords: Numerics, software libraries, visualisation, application development environments

1. APPLICATION DEVELOPMENT REQUIREMENTS

In this paper we present some of the components that are provided by NAG and an infrastructure that can harness those components to provide a rich application development environment. Application developers have a general set of needs which include computational components, visualisation, access to data storage resources and often the ability to interact with the simulation or application during the course of a run. In the following sections we will discuss computational components developed by NAG and a system to provide integrated visualisation in an application, together with examples of such application development building toolkits.

The examples are taken from three European funded projects that have explored these issues in different application areas.

2. COMPUTATIONAL COMPONENTS

NAG has been involved in high-performance libraries for many years. Beginning in the early seventies on vectorization of routines and the

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35407-1_22
R. F. Boisvert et al. (eds.), *The Architecture of Scientific Software*

definition of the BLAS [1], [2] [3], continuing through the eighties and nineties in projects such as LAPACK [4]and ScaLAPACK [5] and the development of their own parallel and SMP libraries.

In developing libraries we need to consider always the needs of the application developers. These vary according to the computational problems that they are developing, the software environment in which they work, and the computing architectures that are available to them. When NAG first became involved in the development of high-performance numerical libraries the languages were FORTRAN, the computing architecture was a mainframe, workstation or for a few lucky individuals, the Cray Supercomputers. Today, the scene has changed dramatically and continues to evolve at a rapid rate. In order to provide the kind of computational support that application developers have come to expect we need to create plug-and-play libraries that can adapt to the appropriate hardware/software environment. This implies algorithms which can take advantage of the architecture but remain portable, in languages that can interoperate with a number of possible application languages and programming paradigms.

Over time NAG has developed many of the components to allow this flexibility of programming environment. These include libraries in FORTRAN [6],[7]and C [8] together with high-performance libraries for both shared memory [9] and distributed memory machines [10]. The algorithms included in these libraries cover most areas of numerical analysis and statistics.

The libraries have become the backbone of many industrial and research acpplications. They have been developed with traditional programing styles in mind. In the examples below, however, we show how these libraries of routines can be thought of as collections of computational components that can be integrated into modern interactive environments.

2.1. PROVIDING AN INFRASTRUCTURE

So far we have talked only about the computational components provided by the NAG libraries. It is just as important to be able to harness these components across distributed resources. In providing application development environments it is important that not only do they meet the ease-of-use requirements, the ability to run on distributed platforms, but also provide visualisation and in some cases the ability to interact with the application.

NAG develops the IRIS Explorer system [11] which although historically a data flow visualisation system, more and more is proving invaluable as the infrastructure for computational problems. The IRIS Explorer environment provides the user with a number of attractive features, including, the ease-of-use visual programming environment - modules of the application are connected by simple pipes, the capability to interact with the application, the module builder which acts as a gateway to Fortran/C/C++ applications. The latest version of IRIS Explorer also comes with a built in collaborative environment in which application developers can cooperate on the same application and of course two- and three-dimensional visualisation is an integrated part of the environment. An application built in IRIS Explorer is simply a set of modules connected - a map (see Figure 1 and 2). An application developer can easily create applications running on distributed machines by simply attaching modules that reside on those machines. The framework is such that the developer can create maps to represent the application and then define the user interface for that application. The end-user may see only the application interface which has been defined for their needs, incorporating output of a variety of forms including graphics and visualisation and slides or dials to alter parameters during the computation. They do not need to know anything about the IRIS Explorer environment and modules in order to run the application.

IRIS Explorer provides two user interface mechanisms: the graphical user interface (GUI) and a command interface. The command interface provides a method to run IRIS Explorer using text-based commands. Commands may be issued directly from the keyboard, or can be supplied in a script that IRIS Explorer runs, using the IRIS Explorer scripting language called Skm (pronounced scheme). The scripting language is useful for three reasons:

- In the case where a standard sequence of operations on a map may be desired such as in testing, this can be provided in skm.
- skm can also be useful in the case that a user wants to run an IRIS Explorer application in batch..
- IRIS Explorer can be remotely from another system, such as a personal computer.

There are many advantages to using the IRIS Explorer as an application framework. First, the visual programming environment enables non-programmers to develop their applications in a user-friendly way. Secondly, the capabilities for defining new interfaces means that specific applications can be tailored to meet the needs of non-specialist end-users.

Thirdly, powerful visualisation facilities become integrated with the application and the user has the ability to interact with the application during the execution.

Using IRIS Explorer in this application development mode has much in common with the more specialised system SciRUN [12] and other similar problem solving environments such as GasTURBLAB [13]. The SciRUN environment has many of the same features but is aimed essentially at the large-scale computations and as such as the essential modules for those problems integrated. In the GasTURBLAB PSE, IRIS Explorer provides the interface to a complex problem solving environment.

IRIS Explorer has provided this type of infrastructure in three systems developed within European funded projects in which NAG is a partner. One of the results from the collaboration in these projects is that the development of IRIS Explorer itself is in pertinent directions to the application areas and the needs of an application development environment. In the next sections we will review the issues raised in each of these projects and the resulting developments.

3. IRIS EXPLORER AS DEVELOPMENT ENVIRONMENT

On this section we review the three projects and the ways in which the computational components and IRIS Explorer have been utilised.

3.1. THE STABLE PROJECT

The aim of the STABLE project was to design, build and demonstrate a modern Statistical Application Building Environment.

The STABLE system was an integration of IRIS Explorer, and an existing widely used statistical software system, Genstat [14]. Part of the project was to build and test the system within three different application areas. A team worked within each of these application areas to provide the specifications and build the particular applications from the prototype system. The three application areas were:

- Design of Experiments and Statistical Process Control for the process for the manufacture of aerosol cans.
- Analysis of field experiments for agricultural crops.
- Forecasts of the electricity demand on the Balearic Islands.

The end-user problems are an important aspect of the project as they will not only inform the design of computational modules, providing a test bed during the development phase, but will also demonstrate the effectiveness of the platform in tackling 'real world' problems. The resulting STABLE environment allows experimentation and prototyping while at the same time provides a framework for the development of complex applications.

In the follow sections we will describe the Stable system and the re-engineering and integration that was required.

The statistical components in STABLE are based mainly on the Genstat statistical system with additional functionality taken from the NAG libraries. Genstat covers all the standard statistical techniques such as analysis of variance, regression and generalised linear models, curve fitting, multivariate and cluster analysis, estimation of variance components, analysis of unbalanced mixed models, tabulation and time series. It also offers many advanced techniques, including generalised additive models and the analysis of repeated measurements.

Genstat is a package created in the 1970s and like many packages of that era, it was written in FORTRAN with a central pool of data and a dependence on COMMON for communication of that data.

Genstat is based on command-interpreters that execute commands issued by the user. Each interpreter is implemented as a hierarchy of functions that interface with the kernel to obtain or define data. In order to create STABLE modules from the Genstat algorithms it was necessary to restructure each interpreter so that the algorithmic code became a closed system, whose input was a data array, control parameters, and workspace, and whose output was another data object. This implies that the architecture with a central pool of data and COMMON communications had to be re-engineered so that it operated on passed arguments rather than using pointers into the main data array. The results were a set of statistical algorithmic Dynamic Link Libraries (DLL) which could then be incorporated into the system as modules.

The integration of the two separate elements, Genstat DLLs and IRIS Explorer, required the definition of a data structure, which provided the interface to the statistical modules and carried all information required from the input of information to the graphical or other interpretation of the data. The data structure, a DataTablet, allowed a flexible infrastructure and an open system. Once an application developer is familiar with the DataTablet it is easy to integrate an existing program or routines into the system. As data is passed through the application all information required ensuring the correct action on the data is immediately available.

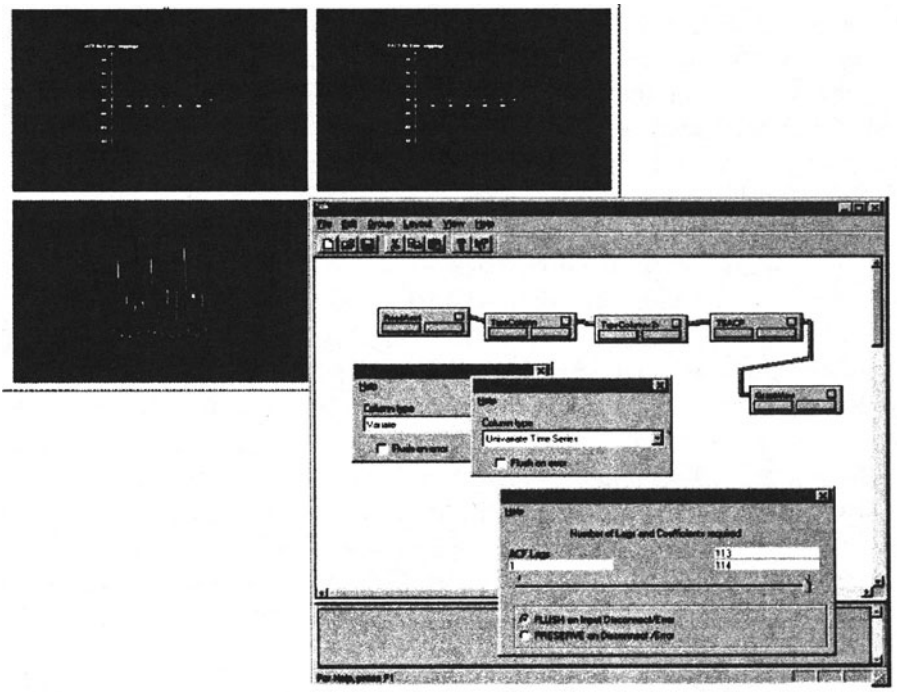

Figure 1 A Stable Example

Figure 1 shows a simple STABLE example. The module ReadAscii inputs data from a file. In this case the user wants to perform a Time Series analysis of the data using the TSACF module. The module TypeColumn is used to set the type of data to a univariate time series. The analysis is performed, the control panel of TSACF allows the users to vary the number of lags and coefficients to be used, and the results directed to GraphView. Since GraphView knows from the datatype that this is Time Series data then it knows the type of visualisations suitable and provides the figures in the background. The tools to rotate and zoom in and out of the plot are an integrated part of IRIS Explorer.

3.2. THE DECISION PROJECT

The Decision project was an HPCN-funded project in Integrated Optimisation Strategies for Increased Engineering Design Complexity. The goal of the project was to a tool for decision makers in new product

design optimisation. Just as in the STABLE project, the development of the Decision platform, DEEP (Decision Explorer Engineering Platform) was driven by industrial applications provided by Nokka Tume a Finnish manufacturer of forestry machinery, MESSET a Finnish manufacturer Electromechanical films, and VTT an independent Finnish research centre. Messet and VTT supplied test problems concerned with structural design of electromechanical film (EMF) flat loudspeakers and with their optimal placement in a room. Also within the project was the development of new optimisation techniques for some of the applications involved.

There were three major stages in the DECISION project:

1 Specification and design

2 Development of optimizers, including: traditional deterministic algorithms, gradient-based algorithms, non-smooth algorithms, multicriteria and stochastic algorithms.

3 Development of the optimisation platform within the industrial problem areas, followed through by the development of commercial products.

Thanks to the tools within IRIS Explorer which allow the creation of new data types and modules it is relatively straightforward to make any of the simulation software into a module of IRIS Explorer and connect to any of the optimisers within the DEEP framework.

The DEEP platform has at its core one module - the Controller module. This acts as a front-end to any of the optimisation algorithms and is provided for the non-specialist end-user to run a previously defined optimisation process. The optimisation modules themselves have been designed such that the more sophisticated user can access them directly, bypassing the Controller module.

The optimisation algorithms incorporated in the system have a variety of requirements in terms of input data and consequent results. In order to create a *plug-and-play* environment for the optimisers it was necessary to define a minimum set of inputs and outputs for the IRIS Explorer module environment. Interface software to meet this specification was then created for existing optimisers.

There is often a need with optimisation functions for information regarding the objective function or constraints to be provided by user-defined functions. In programming languages such as FORTRAN, C or C++ such a function may be provided as an input parameter to the optimisation routine or alternatively the optimisation routine may halt execution and return to the caller when it requires such information.

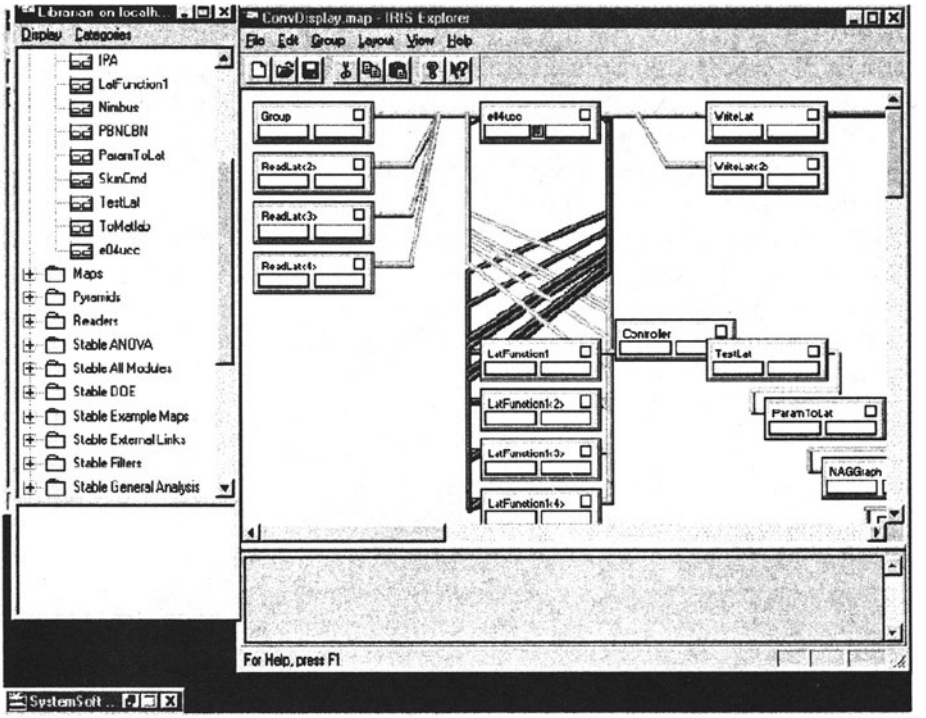

Figure 2 A Decision Map

The latter method, usually referred to as *reverse communication*, provides more flexibility to the user whilst the former, usually referred to as *call-back*, is perhaps more intuitive and efficient. Within the DEEP platform it was important to retain the efficiencies of the call-back system but in order to interface with integration systems, such as IRIS Explorer, to provide the extra functionality of reverse communication. Hence a library was created to help programmers write a function in the call-back model but easily create a reverse-communication interface to it.

As with any application development environment it was also necessary to ensure the system was able to integrate with other systems or environments. In particular in this case one of the solvers that was to be used in an application was written in Matlab. Rather than rewrite the solver in another programming language, a module, *ToMatlab* was developed which would accept any solver written in Matlab as an objective or constraint function.

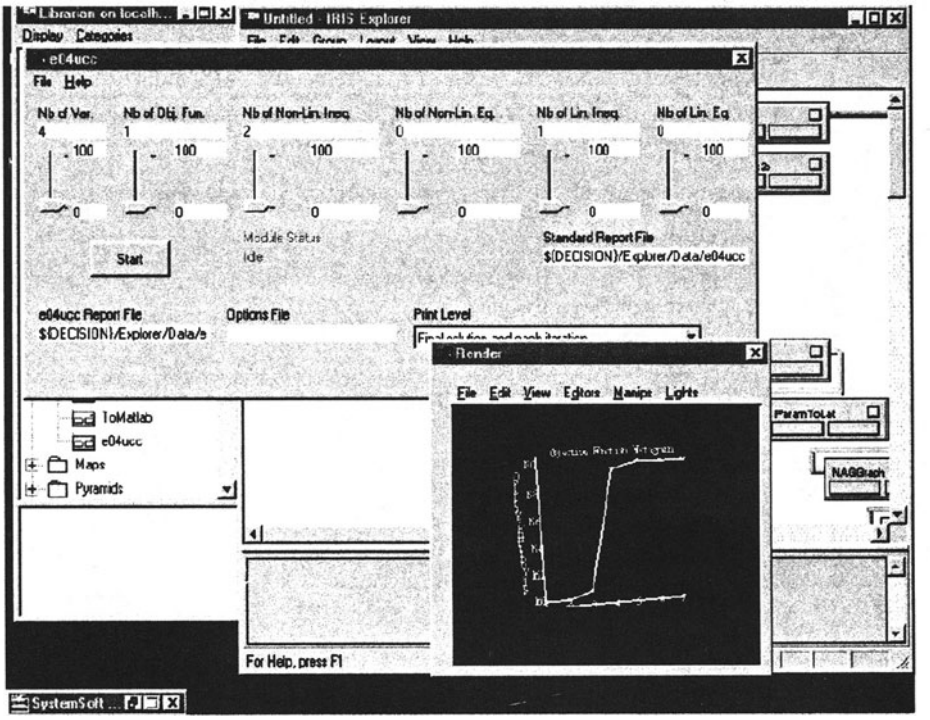

Figure 3 The Control panel and Graphics of the Decision Map

Figure 2 and 3 illustrate a typical map and the user interface to a particular problem. Once the users chooses the optimiser that should be used, the map is automatically configured for that choice.

3.3. THE JULIUS PROJECT

The focus of the Julius project [ref] was the creation of an integrated environment for multidisciplinary engineering simulation, with applications in the aerospace, automotive and manufacturing sectors. Again there were industrial partners with large applications and a variety of algorithmic and other needs. The software pieces involved in the integrated environment included CAD input and repair, a range of 3D mesh generation capabilities, facilities for integrating numerical simulation packages, data extraction and visualization, data base integration, parallel tools and resource scheduling. The simluations and computations were taking place on a variety of compute platforms from PCs to

high-performance parallel systems. See figure 4 for an illustration of the Julius framework.

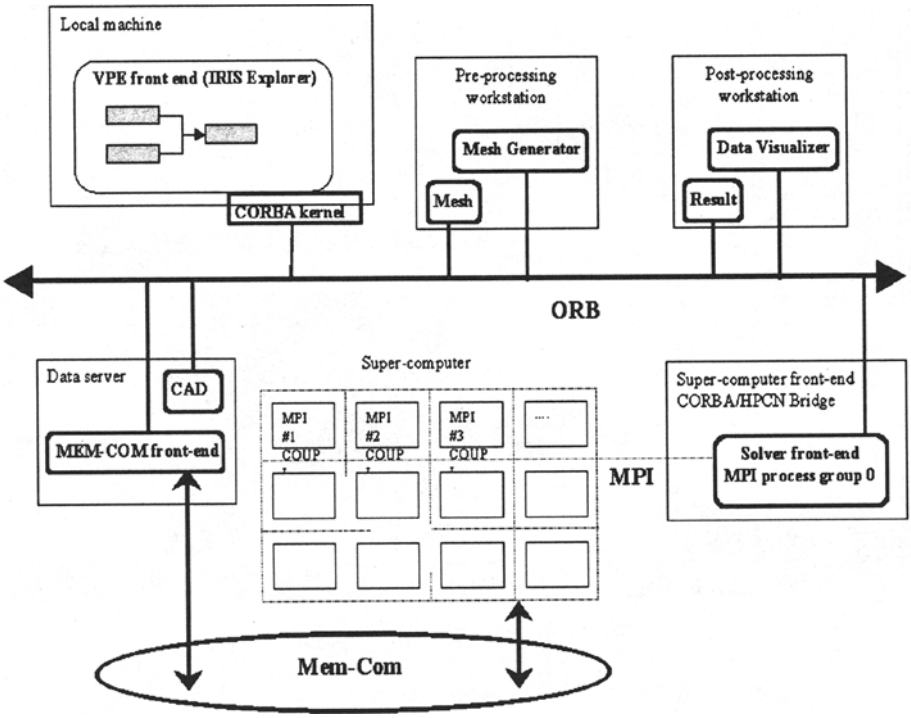

Figure 4 The Julius framework

In Figure 4 the various JULIUS components are wrapped as CORBA objects. The front-end of any parallel component and the database (MEM-COM) are also wrapped as CORBA objects. IRIS Explorer runs on the local machine and provides the visual programming front-end, maintaining the workflow or the metaprogramming sequence. For each JULIUS component there is a corresponding IRIS Explorer module on the local machine. When a module is placed in the map editor its user function binds to the relevant CORBA object. Data objects may also be wrapped as CORBA objects and their object references may be communicated through the IRIS Explorer input and output ports.

The architecture allows a parallel application to employ MPI exclusively within its computationally intensive code. Its CORBA-wrapped front end allows it to be integrated with the other JULIUS tools.

The MEM-COM front end is a CORBA object providing methods to store and retrieve CORBA data objects from within the database. These methods may be invoked directly by other CORBA-wrapped tools within the system.

The advantage of this system is that an application developer can sit at a Linux PC and build and run large-scale applications efficiently on a mixture of computing resources. The workflow sequences can be assembled, modified, stored and retrieved using IRIS Explorer visual programming. Using widgets in the IRIS Explorer modules it is possible to implement computational steering of each component. At any stage in the simulation results can be returned to the PC where the visualisation capabilities of IRIS Explorer come into their own. The data extraction components can be used to significantly reduce the amount of data to be passed to the visualization modules, thereby increasing the efficiency.

4. CONCLUSIONS

IRIS Explorer is proving to be an very effective environment for application development. The European projects discussed above have provided a good deal of feedback from which we have been able to improve IRIS Explorer with application development in mind. It has been clear from these studies that the visual programming interface and the ability to design the module interfaces to meet the needs of the end-user are extremely beneficial - raising the level of the usability of the environment from a sophisticated programmer to a less programming aware audience.

Some of the improvements to be made available in future releases of IRIS Explorer include the ability to create a single module out of a group of modules - this was very mcuh a rquirement from the Stable project where the applications developed where highly complex and required O(200) modules. This enhancement to the system will improve the efficiency of the application and also the time to load the map into memory as the application begins.

The DEEP environment of the DECISION project identified another area where IRIS Explorer could be improved - in the area of loops in maps. In the optimization algorithm maps, such as in figure 2, execution of the loops within the map need to be very efficient. Once this issue had been identified in the project, this feature in IRIS Explorer was refined, hence again making a more efficient system for applications with this requirement.

Both the Stable and Decision projects have led to software components that can be commercially exploited but beyond that they have led to tools and experience that can be applied in building such environments in a broader context.

Clearly the Julius project was very ambitious, involving the integration of software components from a wide range of diverse sources. It is not surprising that the JULIUS consortium found the task of defining an architecture suitable for the needs of all industrial partners and potential end-users particularly challenging. The result of this was that prototype versions of the software were significantly delayed. In its role as exploitation channel for the project NAG found it increasingly difficult to maintain a viable exploitation plan for the software deliverables, and eventually chose to withdraw from the project. Despite this we consider the experience gained in integrating IRIS Explorer with CORBA, and with visualization of very large data-sets, valuable. The project also showed that the IRIS Explorer visual programming model has much to offer problem solving environments in this area.

The task of integrating many varied software components while maintaining efficiency and attempting to satisfy the sometimes diverse requirements of end-users is clearly very challenging. NAG is continuing developments in this area to consider a broader provision of algorithms and tools within the IRIS Explorer framework, to provide an integrated, open environment for a wide range of application areas.

Acknowledgments

The authors would like to thank colleagues at NAG who have supported and contributed to the work presented in this paper. The Decision project (EP 25058), Julius project (EP 25050) and the Stable project (EP 22832) were supported by the European Commission. The authors also acknowledge the contributions made by other members of the project consortia.

References

[1] C L. Lawson, R. J. Hanson, D. Kincaid, and F. T. Krogh. Basic Linear Algebra Subprograms for FORTRAN usage.ACM Trans. Math. Software, 5:308-323, 1979.

[2] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson. An extended set of FORTRAN Basic Linear Algebra Subprograms. ACM Trans. Math. Software, 14:1-32, 399, 1988.

[3] J. J. Dongarra, J. Du Croz, I. S. Duff, and S. Hammarling. A set of Level 3 Basic Linear Algebra Subprograms. ACM Trans. Math. Software, 16:1-28, 1990.

[4] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenny, S Ostrouchov, and D. Sorenson, Lapack Users' Guide, Release 3.0. SIAM, Philadelphia, 1999.

[5] S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, D. Walker, R. Whaley, ScaLAPACK Users' Guide, SIAM, Philadelphia, 1997

[6] The NAG Fortran Library Manual, Mark 19, Numerical Algorithms Group Ltd, Oxford, October 1999

[7] The NAG F90 Library Manual, Release 3, Numerical Algorithms Group Ltd, Oxford, February 1998

[8] The NAG C Library Manual, Release 5, Numerical Algorithms Group Ltd, Oxford, December 1998

[9] The SMP Library User Manual, Release 2, Numerical Algorithms Group Ltd, Oxford, April 2000

[10] The NAG Parallel Library Manual, Release 3, Numerical Algorithms Group Ltd, Oxford, April 2000

[11] IRIS Explorer 4.0 User's Guide, Numerical Algorithms Group Ltd., Oxford, 1999

[12] S.G Parker, D. M. Weinstein, and C.R. Johnson, The SCIRun computational steering software system. Modern Software Tools in Scientific Computing, edited by E. Arge, A. M. Bruaset and H. P. Langtangen. Birkhauser Press, 1997.

[13] S. Fleeter, E. Houstis, J. Rice, C. Zhou, A. Catlin, GasTurbnLab: A Problem Solving Environment for Simulating Gas Turbines, Proc. 16th IMACS World Congress, Aug 2000, No 104-5.

[14] Payne, R.W., Lane, P.W., Digby, P.G.N., Harding, S.A., Leech, P.K., Morgan, G.W., Todd, A.D., Thompson. R., Tunnicliffe Wilson, G., Welham, S.J. and White, R.P. (1993). Genstat 5 Release 3 Reference Summary. Oxford: Numerical Algorithms Group.

DISCUSSION

Speaker: Anne Trefethen

Morven Gentleman : For the re-engineering of Genstat to produce STABLE, how much did you have to rely on prgram understanding tools and techniques, and how much were you able to exploit understanding and advice from the current maintenance staff or original developers (such as John Nelder) from Rothamsted.

Anne Trefethen : The re-engineering work of Genstat was largely completed by the Genstat developers at Rothamsted who were collaborators in the STABLE project. Genstat is developed by a team that evolves in the same way as the program itself. Some of the original developers are still working on Genstat, while others have handed their responsibilities on to the next generation. So, for example, when John Nelder retired in 1985, he passed on the leadership of the Genstat project to Roger Payne (who worked on the STABLE project). Nevertheless, the current team try to keep close links with their predecessors. John Nelder, in fact, remains a very keen Genstat user, has been very interested and impressed by the new framework that STABLE is providing. [*The authors would like to thank Roger Payne for his input to the details of the answer provided here.*]

Richard Fateman : Has Axiom been considered as a software component in these or other projects? What's happening with Axiom?

Anne Trefethen : Axiom was not considered in the projects discussed here but has been involved in other such projects. In particular Axiom was part of the FRISCO project (Framework for Integrated Symbolic/Numeric Computation) a project funded by the European Commission under the Esprit Reactive LTR Scheme (project No. 21.024). NAG is in the process of creating what will be the final release of the Axiom product which will be released with support for a limited period.

Morven Gentleman : You said that the basic module in the software architecture that IRIS Explorer can express is single input/single output. In the standard studies of software architecture such as Garlan and Shaw, this architectural style is very limited unless what is passed in the dataflow is complete databases, in which case the dataflow becomes meaningless. Have you found this to be a problem?

Anne Trefethen : In my presentation I mispoke about the single input/single output. Although there appears to be a single pipe of information entering a module, that pipe may represent several inputs or outputs. In the Stable project, however, we did find the need to create a new datatype for the system to provide sufficient information about the statistical nature of the inputs and outputs. In a way this formed the

interface "glue" between the IRIS Explorer system and the underlying Genstat algorithms.

Robert van de Geijn : The use of a graphical programming language creates the opportunity to arrange icons to make the "picture" (that one would use to explain an algoritm) become the implementation. Here I refer to more than going back to flow-charts.

Anne Trefethen : I agree this opportunity is a strength of a visual programming language, particularly within an educational context. Our collaborators within the STABLE project also found the two dimensional representation of their algorithm, as opposed to a one dimensional piece of code, allowed new insights into their application/algorithm.