From Scientific Software Libraries to Problem Solving Environments

John R. Rice
Purdue University

and

Ronald F. Boisvert
National Institute of Standards and Technology

June 14, 1996

Modern manufacturing design is increasingly driven by the need for both high flexibility and speed in product development. In this environment rapid computational prototyping is of critical importance. Advances in desktop software/hardware, workstation clustering and distributed computing technologies, as well as the ease of access to supercomputing facilities, have been crucial in enabling computational prototyping to emerge as a cost effective alternative to the design new products and for the study of science and engineering phenomena. Unfortunately, effective use of such techniques still requires not only significant expertise in the particular application domain, but also in applied mathematics, numerical analysis, multiple arcane computer systems and languages, and parallel computing techniques. As a result, complex computational modeling remains inaccessible to most practitioners.

This situation can only get worse. Increasingly realistic models will combine computationally intensive problem-solving from many disciplines. One recent study projected that about 10,000 separate programs would be involved in the design of next generation aircraft. Such multidisciplinary design and optimization requires a much higher level of system and model integration than is currently feasible. To provide such capabilities, more powerful and flexible systems for modeling and simulation must be developed. However, to maintain affordability, these systems must enable this work to be done without an increase in development costs. This is a grand challenge in its own right.

Scientific Software Libraries

Mathematical software libraries were introduced in the 1960s to support the reuse of high quality software as a means of transferring numerical analysis and algorithmic expertise to practitioners. Specialized journals, conferences, software repositories (e.g., ACM CALGO, Netlib), and commercial libraries (i.e., IMSL, NAG) were established to support this concept. The increasing number, size and complexity of mathematical software libraries made necessary the development of a classification and indexing of existing and future software modules in terms of the mathematical models they support. A significant effort in this direction is the GAMS on-line advisor system which has become a standard framework of indexing and classifying mathematical software.

Scalability has emerged as a factor of critical importance in the development of software to support computationally intensive scientific computing. There are two different ways to view the scalability of a problem solver. First is that the work to solve a particular problem decreases proportional to the power of the computing resources used. Second is that the work to solve similar problems grows proportionally to the problem size as the size increases. The first view is that of speed up in parallel computing and the second is that of computational complexity using a fixed solver. Both views are important in practical applications.

A library can contain many solvers and is said to be scalable for a particular problem or problem family if it contains a set of solvers which jointly achieve scalability. (One may change solvers as the computer resources changes or as the problem size changes.) Of course, there are many ways to change both computing power and problem size, and thus libraries (or solvers) may be scalable in some ways and not in others. In changing computing resources one usually assumes the changes are balanced in some reasonable way. In changing problem size there is more variability and it is unlikely that a library will be scalable in all possible ways. For example, changing the dimension of physical space in a model, the accuracy required, or the number of physical phenomena in a model can have very different effects. Nevertheless, the goal for a scalable scientific library is that the work required decreases in a direct, maximal way as the computing power increases and that it increases in a known, minimal way as the problem size increases.

One way to make parallel computing easier and more natural is to hide many of the complexities of the underlying parallel architecture through the use of virtual parallel environments (VPEs) and languages (VPLs). These methodologies are based on machine-independent programming models that are simpler to use than the model provided by the hardware. Libraries built on such systems can be made portable to a wide range of hardware platforms. Mathematical software libraries such as LAPACK, ScaLAPACK and PETSc are some early examples of this. New VPE's will be built using emerging technologies such as HPF, MPI and Java.

Object-oriented numerical computing is another new technology which is beginning to have significant impact in this area. The abstract data types of languages such as C++ ease information hiding, providing an effective means of supporting VPEs and raising the level of abstraction for casual users. Its hierarchical class structure also provides new mechanisms for reuse. Examples of new software packages in this area are Diffpack, IML++ and PETSc.

Problem Solving Environments

Although the software library provides one form of abstraction and a facility of reusing software parts, it still requires a level of expertise beyond the background and skills of the average scientist and engineer. A variety of technologies which are now reaching maturity provide a first glimpse at what might be possible in easing the burden of advanced scientific computing. Graphical user interfaces (GUIs), for example, have repeatedly demonstrated the ability to provide both usability and flexibility for applications which just a few years ago required extensive training and technical expertise to master. Such interfaces are now commonplace in mass-market PC-based applications, and, increasingly, monolithic science and engineering applications are being outfitted with rudimentary GUIs to increase their usability. General-purpose systems like Macsyma, MATLAB, Mathematica, and Maple have succeeded by integrating graphical, numerical and symbolic subsystems with a high-level problem specification language to provide rich environments for routine mathematical problem solving.

These ideas have lead to a new concept in software reuse, the *Problem Solving Environment* (PSE). A PSE is a computer system that provides all the computational facilities necessary to solve a target class of problems efficiently. The facilities include advanced solution methods, automatic or semi-automatic selection of solution methods, and ways to easily incorporate novel solution methods. They also include facilities to check the formulation of the problem posed, to automatically (or semi-automatically) select computing devices, to view or assess the correctness of solutions, and to manage the overall computational process. Moreover, PSEs use the language of the target class of problems, so users can solve them without specialized knowledge of the underlying computer hardware, software or algorithms. In principle, PSEs provide a framework that is all things to all people; they solve simple or complex problems, support both rapid prototyping or detailed analysis, and can be used both in introductory education or at the frontiers of science.

Early PSEs have consisted of small set of modules, usually taken from existing libraries, integrated (packaged) to solve a predefined class of engineering or mathematical problems. ELLPACK, MATLAB, and several engineering software systems such as ANSYS, were early examples of this approach. Pre-processing (e.g., computer-aided design, mesh generation) and post-processing (e.g., data visualization) software have experienced a similar evolution. Such libraries, interfaces and pre- and post-processing tools have served to increase the level of abstraction of computational prototyping, allow users with a minimum computational background to design more complex artifacts.

Future PSEs will be distinguished from monolithic systems by the wide domain of problems or applications they can handle; they have built-in flexibility, extensibility, and prototyping facilities. The software architecture of these PSEs will be characterized by the integration methodology used to connect the software parts involved and the underlying execution model assumed. An ideal PSE is the one that can make many decisions for the user by consulting its associated knowledge base. This leads to an alternate definition of a PSE in terms of its autonomous components as follows:

PSE = user interface + libraries + knowledge base + integration

Some examples of such PSEs for the domain of partial differential equations (PDEs) are under development at Purdue University. An early example, //ELLPACK (pronounced "parallel ELLPACK") provides a variety of integrated high-level tools to describe PDEs, domains and boundary conditions, to partition the domain for parallel execution, to select solution methods, to monitor the solution process, and to visualize the results. A successor project, PDELab, is developing a general architectural framework for constructing systems of this type (see Figure 1).

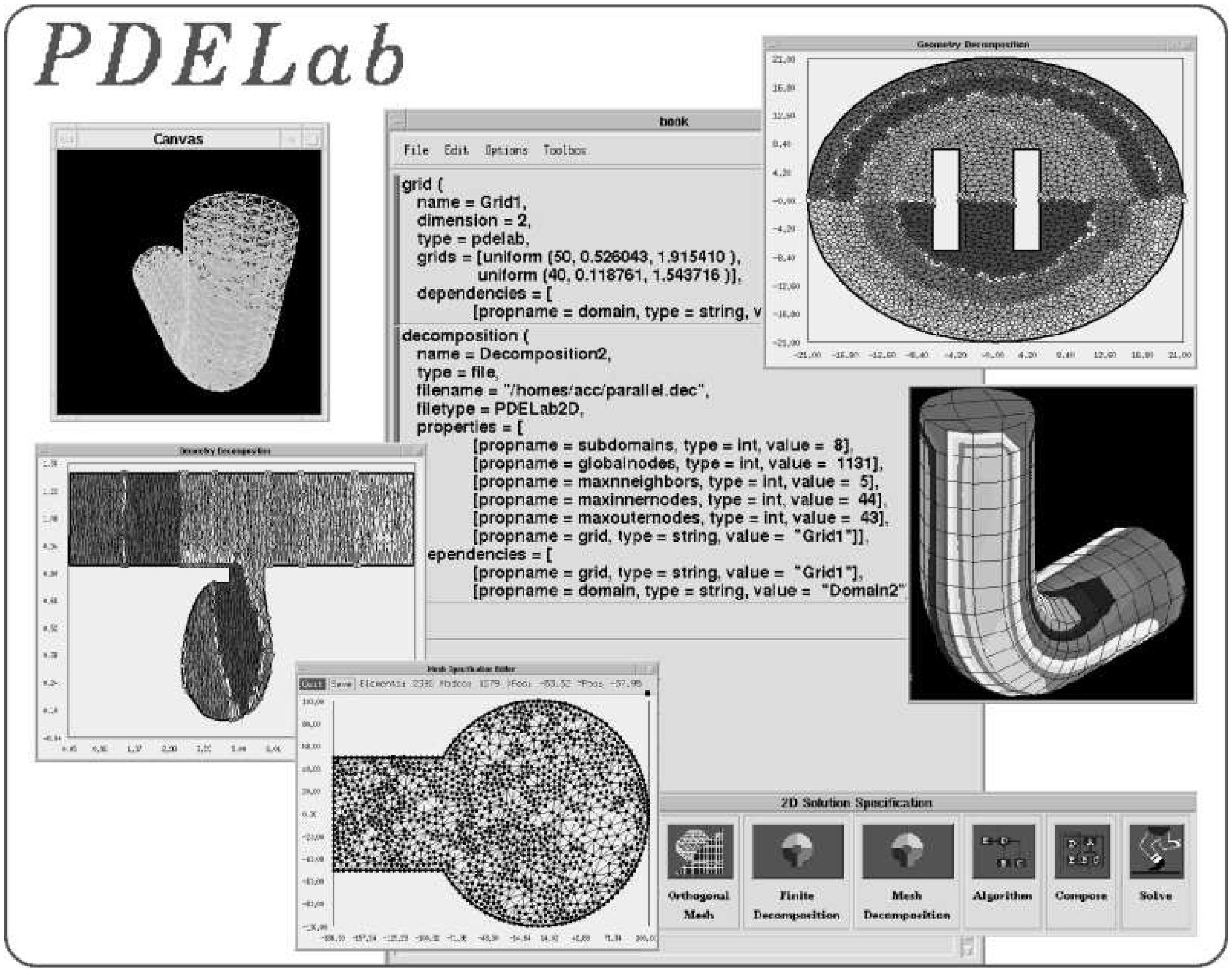
One of the purposes of the workshop was to identify key enabling components for PSEs that might lead to such a common architecture or "kernel" upon which future PSEs might be built. Unfortunately, the set of technologies that make constructing, prototyping, realizing, testing, maintaining, and evolving PSEs possible, easier, simpler, or faster, is wide and varied. Some emerging technologies which will undoubtably play a role in the architecture of future PSEs were identified by workshop participants, however. Examples are listed in Table 1.

Problems, Barriers, and Research Directions

The workshop participants identified a wide range of open problems and barriers to creating the next generation of high level scientific software systems. Research directions are naturally associated with solving and overcoming them. They range from basic scientific problems to an organizational shortcoming. The seven principal technical problems and barriers are discussed here, the complete report [1] identifies about 20 more.

- *The currently common "GUI plus monolithic solver" paradigm is too restrictive.* It works well when the scope of an application is narrow enough that all important options and specializations can be anticipated and provided for in the GUI. The more complex applications require programmability (at a very high level) plus control and customization of the PSE and its problem solving process. Domain specific languages and compliers must be integrated into dynamically configured, "smart" PSEs.

Figure 1: PDELab, a framework for PDE-based PSEs



```
grid (
  name = Grid1,
  dimension = 2,
  type = pdelab,
  grids = [uniform (50, 0.526043, 1.915410 ),
           uniform (40, 0.118761, 1.543716 )],
  dependencies = [
          [propname = domain, type = string, v
decomposition (
  name = Decomposition2,
  type = file,
  filename = "/homes/acc/parallel.dec",
  filetype = PDELab2D,
  properties = [
          [propname = subdomains, type = int, value =  8],
          [propname = globalnodes, type = int, value =  1131],
          [propname = maxnneighbors, type = int, value =  5],
          [propname = maxinnernodes, type = int, value =  44],
          [propname = maxouternodes, type = int, value =  43],
          [propname = grid, type = string, value =  "Grid1"]],
 ependencies = [
          [propname = grid, type = string, value =  "Grid1"],
          [propname = domain, type = string, value =  "Domain2"
```

Table 1: Enabling Technologies for Future PSEs

- Parallel/Distributed computing — meta-computing
- "Low-level" virtual machines such as MPI, PVM, etc.
- "Fine grain, high-level" languages (C++, HPF etc.)
- Scalable mathematical software libraries
- Tools for computational geometry and grid generation
- Tools and techniques for coarse grain software integration (such as CORBA, Opendoc and the software bus)
- "Web-ware" and scripting middle-ware (Perl, Java, VRML, Python, etc.)
- Interface specification support and information exchange protocols (i.e., meta-data for objects and services)
- Interoperability and wrapper technology for legacy systems
- Visualization including virtual reality, televirtuality, etc.
- Interactive interface development (GUI) technologies
- Configuration control and human-in-the-loop (computational steering)
- Collaborative computing technology
- Federated multi-media databases
- PSE templates and frameworks

- *The understanding of the architecture, technologies and methodologies for scientific PSEs is immature.* It is clear that the high goals for scientific PSEs require modularity at all levels, a wide range of powerful solvers, effective expert system support and flexible software control/composition. This leads to the simplified architecture for PSEs given earlier but the structure of the intermediate levels (the middleware) between this and the underlying algorithms is less clear. The *software bus* is the glue that holds everything together and there are operational PSEs which use this approach. However, none of these have the full range of capabilities needed for PSEs and the effectiveness of these designs is yet to be proven.

- *Our ability to achieve easy software evolution and to incorporate large legacy systems is weak.* Software evolution is essential so new algorithms, techniques, and requirements can be accommodated. Much study has been given to providing for modularity (the "plug and play" paradigm) for new software but the reality is that there are very large, very poorly understood, and very useful legacy software systems that must be incorporated into PSEs. It is too expensive to reimplement them. Thus the PSE architecture and middleware must support evolution well. The efforts to develop interconnecting software components (e.g., Opendoc, OLE, CORBA, ILU, Glish, PolyLith) must be expanded to connect very heterogeneous components (different sizes, different languages, different execution environments, different everything).

- *The understanding of the architecture, technologies, and methodologies for scalable problem solving is immature.* The gap between algorithm analysis (based on idealized computers) and real applications is still very large. Reasonable scalability must be delivered to a wide variety of users for a broad range of applications without tuning individual codes for every computing environment. The virtual parallel environments are clearly a step in the right direction and they are one of the key enabling technologies in this area. However, their effectiveness is still uneven and, at the same time, they must be extended to even more heterogeneous computing environments (e.g., meta-computers and network-based solvers).

- *There is persistent difficulty in creating and managing ever more complex software systems.* Many of the problems of next generation scientific software systems seem to be common to software in general. The search for a "silver bullet" for this problem has not succeeded. Perhaps we should consider the possibility that different application areas require different software design and production methodologies. After all, the engineering methodologies for automobiles, bridges, and computers have very little in common. The same might be true of the software engineering methodologies in diverse applications.

- *The lack of versatile, general systems for complex three-dimensional geometry is a substantial barrier to building software in many science and engineering area.* There exist several incompatible, approximate ways (e.g., computational solid geometry, triangulations, boundary representations, spline-like methods, pixel-like methods) to represent geometry. While all computation involves approximations, those in geometry are much less effective than in numerical, symbolic, or logical computing. An obviously simple shape may require many megabytes of data for some of these representations. To change from one representation to another might require a major computation by an amazingly complex program.

- *The understanding and experience in the methodologies and use of knowledge bases, expert systems, etc., for scientific computing is immature.* "Intelligence" at a reasonable level is essential for complex PSEs. However, the current state-of-the-art of knowledge-based frameworks for scientific computing is low-level and far from adequate for building PSEs. A better understanding is needed of appropriate knowledge formats, ontologies, exchange protocols, and databases of meta-data.

In addition, there is a serious organizational barrier. Success in developing these scientific software systems requires that a variety of community standards be adopted for interoperability, interfaces, library structures, terminology, etc. Such standards have been developed in the past by group efforts resulting in "community consensus" rather than a formal national or international standards effort. Successful past efforts include the GAMS classification of scientific software routines, the BLAS and LAPACK in linear algebra, the MPI effort for message passing, the HPF language, and CORBA for interfaces. The underlying problem is that the people who will develop the next generation of scientific software do not yet form a community. The technical areas involved are so diverse that there is no one (or few) forum where the participants meet. The result is that incompatible standards and terminology have been created simply because the sub-communities have not been aware of how their activities will interact.

The thrusts for future research must include the solution of these problems and the removal of the barriers. The workshop identified seven particularly important directions. A common theme is to provide structure and modularity to scientific software systems.

- *The plug-and-play paradigm.* Plug-and-play means that one can take a new algorithm, a new representation, a new implementation, etc., and exchange it for an existing software component to see how it performs. A common infrastructure does not automatically guarantee that plug-and-play is easy even though it is essential. The evolution of PSE solving technology seems to demand this capability, otherwise it becomes too expensive to test new ideas and approaches — and too expensive to put them into operation. Openness is essential because standards can be (and often have been) used as a mechanism to exclude certain groups or methodologies.

- *Network based PSEs and problem solving.* We need to integrate the methods and expertise of the distributed and parallel computing communities. Some of the work on metacomputing and distributed shared memory illustrates this but we need to address it more broadly. Parallel computing has taught us much about synchronization and decomposition which needs now to be implemented in a distributed heterogeneous fashion. Typical problems include: providing a shared memory model of the world wide metacomputer; developing new algorithms tolerant of the latency in geographically distributed systems; taking the promising research in scalable I/O for scientific problems and implementing it in a database-dominated distributed environment; efficient implementation of MPI with interoperable ATM networks using switches supporting ATM adaptation layers from outside the parallel computing community.

- *Integrate PSEs into science education.* Science and engineering are tree structured disciplines that start from very basic roots and grow through branching to more and more advanced technologies. This structure should be reflected in the science and engineering PSEs of the future. PSEs for a particular area should be effective both for simple, elementary problems and for complex, demanding applications. While it is implausible that there will ever be a universal science and engineering PSE, this is a vision that could be considered for utopia. Students should be able to build directly on their previous skills and move transparently to advanced PSEs as their education progresses.

- *Larger scale components for building PSEs.* The software library concept has been very successful for encapsulating algorithms — large and small — for widespread reuse as software objects. This methodology is based on a static view; the object is invoked and the desired results returned. This view is inadequate for components that perform data visualization, symbolic manipulation, information searching or geometric modeling. Such components are required for many PSEs and yet the current practice is to make such systems stand-alone, designed to interact only with a human user. In the future all PSEs should become candidates for components of even larger PSEs. A new vision for scientific computing is to combine and integrate multiple computing paradigms (e.g., symbolic manipulation, numeric computing, graphics visualization, database management, document preparation, parallel/distributed processing, Web-based retrieval). A PSE can become much larger than the sum of its parts and will raise the power, functionality, and convenience of scientific computing systems to a new level.

- *Technology for multi-disciplinary PSE combinations.* One view is that science and engineering PSEs might grow in size and power indefinitely in order to handle the tremendously complex simulations of the real world (e.g., create a PSE for the complete analysis, design and manufacturing of transportation machines — cars, planes, ships, etc.). An alternative is to have PSEs for large (but manageable) disciplines which collaborate and interact for a particular application. Thus a PSE might be used alone for one problem, used as a component in a larger PSE, and as an agent in a collaboration for another problem. All three uses are appropriate. Building a large, permanent PSE allows one to optimize many aspects of the internal structure and algorithms and to develop a specialized user interface. In either approach, a PSE in one discipline must be suitable for using together with PSEs from other disciplines.

- *Integrate real and simulated systems.* The weakness of the computational approach to science and engineering is that one must understand the phenomenon of interest very well, well enough to create software to simulate it accurately. This might not always be practical or possible or economical. Similarly, building real systems — or parts of them — might not always be practical or possible or economical, so combining real and simulated systems is attractive. This methodology is already used successfully in some virtual environments, e.g., aircraft simulators for training pilots. The methodology for combining PSEs should allow one to "plug-and-play" with real and/or simulated phenomena.

- *Dynamic software parts technology for scientific computing.* A recurring theme of the above research directions is that the next generation of scientific software will be complex and should be built with great modularity. It is also clear that it will execute in a widely distributed and heterogeneous environment with computations tailored for the problem at hand. Thus we must begin thinking in terms of a dynamic software parts technology. Existing static network-based information repositories need to evolve into highly interoperable problem-solving service providers. Repositories for high level scalable computational models for a variety of disciplines need to emerge. Standardized meta-data for collections and services to aid in search and discovery among repositories will become increasingly important.

Recommendations

The workshop presented a bold and ambitious future for scientific software development, one that invites a broad range of recommendations. Six of the most immediate and urgent recommendations are given here; they vary from research through commercialization to community building.

1. *Create and explore effective frameworks and architectures for building PSEs.* There is an urgent need to understand better the strengths and weaknesses of the approaches proposed so far. There are four sub-recommendations here:

- *Identify the relevant enabling technologies and their roles in building PSEs.* Particular attention is needed in critical component technologies such as computational geometry, grid generation, and sparse matrix methods; in each case new, more effective algorithms are needed, as well as community standards for core functions, data representation and exchange.

- *Focus on the "glue" for building PSEs as well as the bricks.* A paradigm shift in scientific software development is needed, changing its emphasis from bricks to glue. New methodologies must be developed to allow independently developed components to be easily combined to cooperate in the solution of common problems. Of crucial importance here are standards for interfaces and for data exchange.

- *Determine how to maintain openness in the architectures.* Open architectures that promote the development of reusable modules are needed. By openness we mean that the architecture must support extendability and that individual modules must be designed so that they easily adapt to a wide variety of computing environments and contexts (e.g., with input provided via a subroutine call, by a socket connection over a network, or by a user typing at a keyboard). This openness must be maintained at all levels of the software structure, from libraries of basic algorithms through the middleware and to the large scale components of PSEs. An important aspect of this for scientific computing will be the ability to provide estmates of required computational resources (through the use of performance models, for example) so that automated systems for module selection can be constructed. Also, such modules should also provide an estimate of the accuracy of their computed solutions, so that the overall fidelity of mathematical models based on composition of modules can be judged.

- *Develop systematic abstract library structures for a number of important science and engineering disciplines.* The success of such architectures can only be judged by using them in real applications. Thus, complete and precise definitions of modules and their interactions in this context must be developed for particular application domains. The structure should be based upon mutliple levels of abstraction. The definition might be bound to particular languages at the lower levels and not at the higher ones. It is envisaged that this action will be broken into a number of parts addressed by various groups.

2. *Develop the PSE infrastructure, enabling technologies and building tools.* Many of these are known or partly known but we are still lacking a complete picture. Concrete examples are needed to evaluate the ideas and to promote the development of this field. Three sub-recommendations here are:

- *Develop an infrastructure for high performance software.* New techniques, standards and tools for constructing scientific software based on the use of a hierarchy of levels of abstraction are necessary. This is one of the tenets of object-oriented approaches to software development, an approach which is only just beginning to affect scientific software. Of prime importance are techniques which enhance reusability, reliability, robustness, as well as performance portability. Also crucial are mechanisms for validating the correctness of numerical software. Without these features successful commercialization of high performance scientific computing tools cannot occur. The research community must lead the way in developing a broad-based framework in which to build scalable and maintainable component libraries for a wide range of core problems.

- *Devise a cost effective methodology for building PSEs.* New emphasis must be placed on research in technologies which enable development of the next generation of scientific PSEs. In particular, the identification of “middleware” which can provide a common substrate for PSE development should be a high priority. Examples are open distributed object management systems which promote coarse-grain software component integration. The most important barrier to creating the next generation of scientific software is cost. Experience has shown that it takes many person-years to build even a small scale PSE with the current state of infrastructure. On the other hand, the lack of experience in building PSEs is a barrier to realizing better and more appropriate PSE infrastructure so that the cost of building PSEs may be reduced. This cycle must be broken if PSEs and scalable libraries are to deliver on the promise of providing interesting answers to important problems in a reasonable time. An essential feature of any solution to this difficulty must lie in a component-oriented approach where individual groups do not build a full PSE, but rather components for a PSE that is then incorporated into larger systems which have enough infrastructure and capability to be interesting for users.

- *Create a testbed based on partial differential equation (PDE) applications.* Eventually PSEs should provide such testbeds as a by-product of their “plug-and-play” design. But it appears possible to create something much less complicated (and implemented sooner) for the specific purpose of evaluating PDE library components. The PDE area is recommended because it has most of the complexity of other applications and the software systems that are the most advanced.

3. *Increase the commercialization of PSEs.* Commercialization of PSEs is critical to their sustainability. However, since component technologies are moving so rapidly, and industry commitments are increasingly short term, it is unrealistic to expect that such systems will be developed

solely in the commercial sector. As a result, there is great need for a long term government-sponsored research agenda to advance development of these systems and their transfer to commercial software vendors.

4. *Create some complete scientific PSEs based on a common PSE architectural framework..* Efforts to build prototype PSEs must be encouraged, both to learn what middleware is needed as well as to demonstrate the utility of newly developed middleware. Such efforts should be multidisciplinary, drawing on experts in human-computer interfaces, artificial intelligence, database management, parallel processing and networking, as well as experts in the particular application domain of the PSE. Infrastructure development cannot occur in a vacuum. Extensive experimentation will be necessary to determine what is feasible and to demonstrate the merit of new approaches. Infrastructural components must be shown to be practical and cost-effective so that commercial support is viable. Without this such new technologies will not be sustainable.

5. *Evaluate some large scale knowledge bases for scientific applications.* Knowledge-based systems with particular domain expertise must be devised to aid the use of complex PSEs. For example, such systems could help select among solvers, data structures, and network resources to maximize performance. Detailed performance models must be developed for software as well as for the underlying communications and computing fabric to provide the basis for such reasoning systems. However, the current state-of-the-art of knowledge-based system frameworks for mathematical computing is very low-level and far from appropriate for building PSEs. More work is needed in developing appropriate knowledge formats, ontologies, exchange protocols and databases of meta-data.

6. *Create forums and mechanisms to promote interaction within the PSE and scientific application communities.* The PSE development community is not coherent; indeed, many researchers developing application-specific PSEs are unaware of basic PSE concepts. Thus, there is a great need for forums in which PSE developers in all domains can exchange case histories, design concepts, infrastructure and components. These forums must include industry and academia, builders and users. They must develop mechanisms to propose, evaluate and, eventually, adopt community standards for libraries, interfaces, structures, representation, middleware and PSE components.

References

[1] John R. Rice, Scalable Scientific Software Libraries and Problem Solving Environments, CSD TR-96-001, Computer Sciences, Purdue University, W. Lafayette, IN, Jan. 3, 1996.

(See also the URLs of the Glossary.)

The Workshop

Software libraries provide encapsulated problem solving power and *problem solving environments* (PSEs) give ordinary users painless access to problem solving power. Thus the structure of libraries and the design of PSEs are inextricably linked. A few application areas (linear algebra is an example) have a straightforward structure amenable to software libraries, and a widely accepted language (mathematics) to use as the basis for a PSE. These essential elements are missing in many important scientific application areas where both libraries and PSEs are embryonic. These areas often have almost unlimited complexity so that high performance computing power is essential. Thus, both the PSE design and library structure must be scalable in the complexity of the applications.

Experts in these subject areas from academia, government, and industry (see Table 2) met at Purdue University last fall[1] to explore the state-of-the-art in these two areas, their interdependence, and their combined future. The workshop, entitled *Scalable Libraries and Problem Solving Environments*, was organized by John Rice and Elias Houstis of Purdue and sponsored by DARPA and NSF[2]. The workshop focused on applications based on partial differential equations (PDEs) and closely related areas for its examples. Overview talks by John Rice (on PSEs), James Demmel (on scalable libraries), and by Robert Nelson (on the PDEase system), set the stage for the two and a half day meeting, which was centered around eight separate panel discussions (see Table 3). Afterwords, participants formed four groups, each of which developed a separate white paper. These white papers are available as a technical report [1]. This article summarizes the ideas expressed there.

Glossary

ACM CALGO The Collected Algorithms of the Association for Computing Machinery. (See `http://www.acm.org/toms/`.)

Ansys A system for engineering design (e.g., structural analysis) marketed by Ansys, Inc. (See `http://www.ansys.com/`.)

ATM Asynchronous Transfer Mode. An evolving connection-oriented networking technology based on cell relay switching which can provide a very fast common interface for a wide variety of network services.

BLAS Basic Linear Algebra Subroutines. A core linear algebra library which has enhanced performance portability for a wide range of computer architectures. (See `http://www.netlib.org/blas/`.)

CORBA Common Object Request Broker Architecture. A standard for interchangeability of objects set forth by the Object Management Group. (See `http://www.omg.org/`.)

DIFFPACK An object-oriented framework for the solution of partial differential equations. (See `http://www.oslo.sintef.no/avd/33/3340/diffpack/`.)

ELLPACK A system for solving elliptic boundary value problems developed in the late 1970s, featuring a high-level problem description language and a collection of some 50 problem-solving software modules. (See `http://www.cs.purdue.edu/ellpack/`.)

//ELLPACK Parallel ELLPACK, a problem-solving environment for partial differential equations. (See `http://www.cs.purdue.edu/research/cse/pellpack/`.)

GAMS The Guide to Available Mathematical Software. A cross-index and virtual repository of mathematical software built upon a detailed problem classification system. (See `http://math.nist.gov/gams/`.)

Glish A software bus allowing transport of large volumes of data between tasks, possibly on different machines. (See `ftp://ftp.ee.lbl.gov/glish/`.)

[1]September 25-27

[2]The workshop was supported by DARPA under ARO grant DAAH04-94-G-0010 and the NSF under grant CCR95-23243.

HPF High Performance Fortran. A Fortran extension with facilities for data-parallel programming. (See `http://www.crpc.rice.edu/HPFF/home.html`.)

ILU Inter-Language Unification system. A multi-language object interface system. (See `ftp://parcftp.parc.xerox.com/pub/ilu/ilu.html`.)

IML++ Iterative Methods Library. A C++ templated library of iterative methods for linear systems. (See `http://math.nist.gov/pozo/iml++.html`.)

Java A platform-independent network-based programming language. (See `http://java.sun.com/`.)

IMSL General-purpose mathematical subroutine libraries marketed by Visual Numerics, Inc. (See `http://www.vni.com/`.)

LAPACK A linear algebra library portable over a wide range of environments from RISC to shared-memory multiprocessor systems. (See `http://www.netlib.org/lapack/`.)

LINPACK An influential library of subroutines for solving linear algebraic systems based on the BLAS. It has been superceeded by LAPACK. (See `http://www.netlib.org/linpack/`.)

Macsyma One of the earliest symbolic computing systems, now marketed by Macsyma, Inc. (See `http://www.macsyma.com/`.)

Maple An interactive mathematical problem-solving system featuring symbolic, numeric and visualization facilities marketed by Waterloo Maple, Inc. (See `http://www.maplesoft.com/`.)

Mathematica A software system for numeric, symbolic and graphical computations marketed by Wolfram Research. (See `http://www.wri.com/`.)

MATLAB A technical computing environment for high-performance numeric computation and visualization marketed by The MathWorks, Inc. (See `http://www.mathworks.com/`.)

MPI Message Passing Interface. A standard for writing parallel programs in message-passing environments. (See `http://www.mcs.anl.gov/mpi/`.)

NAG General-purpose mathematical subroutine libraries marketed by the Numerical Algorithms Group, Ltd. (See `http://www.nag.co.uk/`.)

Netlib An extensive repository of mathematical software and related information developed and maintained by the University of Tennessee at Knoxville, Bell Laboratories, and Oak Ridge National Labs. (See `http://www.netlib.org`.)

OLE Object Linking and Embedding. A collection of system-level object-oriented technologies for linking and embedding. (See `http://198.105.232.6/oledev/`.)

Opendoc An open, multiplatform architecture for interconnecting component software. (See `http://www.cilabs.org/`.)

PDE Partial Differential Equation.

PDEase A system for finite element analysis of PDEs marketed by Macsyma, Inc. (See `http://www.macsyma.com/`.)

PDELab A framework for developing PDE-based PSEs under development at Purdue. (See `http://www.cs.purdue.edu/research/cse/pdelab/`.)

Perl Practical Extraction and Report Language. A powerful scripting language. (See `http://www.perl.org/`.)

PETSc Portable, Extensible Toolkit Scientific computing. A suite of data structures and routines for both uni- and parallel-processor numerical solution of large-scale PDE problems. (See `http://www.mcs.anl.gov/Projects/petsc/petsc.html`.)

Polylith A software bus for the interconnection of mixed-language components for execution in heterogeneous environments. (See `file://thumper.cs.umd.edu/files/docs/2469.ps.Z`.)

PSE Problem-Solving Environment. An integrated system that provides all the facilities needed to solve a target class of problems conveniently and efficiently.

PVM Parallel Virtual Machine. A message passing library for heterogeneous distributed computing. (See `http://www.netlib.org/pvm/`.)

Python An interpreted, interactive, object-oriented scripting language. (See `http://www.python.org/`.)

RISC Reduced Instruction Set Computer.

ScaLAPACK A linear algebra library for distributed-memory multiprocessors. (See `http://www.netlib.org/scalapack/`.)

Software Bus An agent which mediates interconnection between heterogeneous software components.

VPE Virtual Parallel Environment. A software development environment which supports parallel programming while hiding many of the complexities of the underlying parallel/distributed computing system.

VRML Virtual Reality Modeling Language. (See `http://www.vrml.org/`.)

Authors

John R. Rice is W. Brooks Fortune Professor of Computer Sciences at Purdue University. After receiving his Ph.D. in mathematics from Caltech in 1959 he held positions at the National Bureau of Standards and General Motors Research Labs. In 1964 he joined the Purdue faculty, and was head of the Department of Computer Sciences from 1983-1996. The author of several books on approximation theory, numerical analysis, computer science, and scientific software, Rice founded the *ACM Transactions on Mathematical Software* in 1975 and remained its Editor-in-Chief until 1993. He is a member of the National Academy of Engineering, the IEEE Computer Society, ACM, IMACS and SIAM. He serves as *IEEE CS&E*'s area editor for problem-solving environments. More information can be found on his Web page, `http://www.cs.purdue.edu/people/jrr`.

Ronald F. Boisvert is leader of the Mathematical Software Group at the National Institute of Standards and Technology (NIST). He received his Ph.D. in computer science from Purdue University in 1979 and has been at NIST (formerly the National Bureau of Standards) ever since. His research interests include numerical solution of partial differential equations, mathematical software, and information services that support computational science. He contributed to the development of the original ELLPACK system, the NBS Core Math Library, VFFTPACK, VFNLIB, and leads the Guide to Available Mathematical Software project (`http://math.nist.gov/gams/`). He has been Editor-in-Chief of the *ACM Transactions on Mathematical Software* since 1993, and is a member of the IEEE Computer Society, ACM and SIAM. More information can be found on his Web page, `http://math.nist.gov/acmd/Staff/RBoisvert/`.

Table 2: Principal Workshop Participants

Bajaj, Chandrajit	Purdue University
Boisvert, Ronald	NIST
Bramley, Randall	Indiana University
Char, Bruce	Drexel University
Demmel, James	University of California, Berkeley
Fox, Geoffrey	Syracuse University
Gallopoulos, Stratis	University of Illinois
Grimes, Roger	Boeing
Grosz, Lutz	University of Karlsruhe
Harrand, Vincent	CFD Research Corp.
Heath, Michael	University of Illinois
Houstis, Elias	Purdue University
Johnsson, Lennart	University of Houston and Harvard
Joubert, Wayne	Los Alamos National Laboratory
Lucas, Robert	ARPA
Marinescu, Dan	Purdue University
Masso, Joan	University of Illinois
Mitchell, William	NIST
Nelson, Robert	SPDE, Inc and Macsyma
Parashar, Manish	University of Texas at Austin
Ribbens, Calvin	Virginia Tech
Rice, John	Purdue University
Saied, Faisal	University of Illinois
Sameh, Ahmed	University of Minnesota
Sewell, Granville	University of Texas at El Paso
Sherman, Andrew	Scientific Computing Assoc., Inc.
Sincovec, Richard	Oak Ridge National Laboratory
Singhal, Ashok	CFD Research Corp.
Skjellum, Anthony	Mississippi State University
Smith, Barry	Argonne National Laboratory
Thompson, Joseph	Mississippi State University
Wang, Paul	Kent State University
Weerawarana, Sanjiva	Purdue University

Table 3: Panels at the Workshop

1. *Application-Specific PSEs.* L. Johnsson, moderator.
   R. Bramley, S. Gallopoulos, D. Marinescu.
2. *User Interfaces for PSEs.* E. Houstis, moderator.
   C. Bajaj, R. Nelson, G. Sewell.
3. *Enabling Technologies for PSEs.* R. Boisvert, moderator
   F. Saied, P. Wang, S. Weerawarana.
4. *Virtual Parallel Environments and Languages.* G. Fox, moderator
   M. Parashar, C. Ribbens, A. Sherman.
5. *Architecture of Scalable Libraries.* M. Heath, moderator
   R. Grimes, R. Sincovec, A. Skjellum.
6. *Characteristics and Components of PDE Libraries.* J. Rice, moderator
   J. Demmel, L. Grosz, W. Mitchell.
7. *How to Achieve Scalability in PSEs for PDEs.* A. Sameh, moderator
   W. Joubert, J. Rice, B. Smith.
8. *Future Directions and Research Thrusts.* R. Lucas, moderator
   R. Boisvert, G. Fox, E. Houstis, L. Johnsson, J. Rice, A. Sameh.