On the Future of Problem Solving Environments

Elias N. Houstis and John R. Rice*

March 30, 2000

Abstract

In this paper we review the current state of the problem solving environment (PSE) field and make projections for the future. First we describe the computing context, the definition of a PSE and the goals of a PSE. The state-of-the-art is summarized along with sources (books, bibliographics, web sites) of more detailed information. The principal components and paradigms for building PSEs are presented. The discussion of the future is given in three parts: future trends, scenarios for 2010/2025, and research issues to be addressed.

*This work was supported in part by a grant from Kozo Keikaku Engineering, Inc.

Contents

1 THE COMPUTING CONTEXT

Solving problems with computers involves three technologies. First is the *hardware* that executes basic operations (arithmetic, data transfers, logical operations, displays,...) very rapidly. Second is the *algorithmic technology* which takes a problem as input, and manipulates its data to produce the solution. Algorithms are specified rather abstractly and often involve manipulations which do not exist exactly as (or even close to) basic operations in the hardware. The final technology is *programming* which takes abstract algorithms and expresses them in complete detail so they can be executed approximately by computer hardware. While programming exists at several levels, e.g., machine language, assembly language, programming language (Fortran, C, Java,...) and higher (more abstract) levels, we define programming in terms of the currently common high level programming languages such as Ada, Fortran 90, C, C++, and Java. This class of languages has been in widespread use for 40 years with a modest rate of evolution and the bulk of programming, especially for science and engineering, is done using these languages.

	1945	1955	1970	1995	2010	2025
Speed (flops)	0.05	3K	20M	6G	10T	5P ?
Memory (words)	–	4K	200K	50M	100G	50P
Store (words)	–	10M	200M	500G	1P	2Q
¢ per Billion Adds	600M	250K	20K	70	0.02	0.005 ?

Table 1. Measures and projections of the increase in computer hardware power from 1945 to 2025. The specific hardware is a typical high end computer used by a small group of scientists and the abbreviations used are K=thousand, M=million, G=billion, T=trillion, P=quadrillion, Q=quintillion.

Everyone is familiar with the enormous increase in the speed of computer hardware and in other measures of its power. Table 1 provides a little data about these past increases and some projections for the future. People are much less familiar with the enormous increase in algorithmic power over the past 60 years. Before computers there was little incentive to develop powerful algorithms using basic mechanical operations and one can check that the algorithms known in the 1935–45 period were quite slow by todays standards. This is illustrated in Table 2 where the time to solve a particular problem using a *fixed computer* is given for the period 1945–1985 [Rice, 1976b, Rice, 1992]. The problem is to compute the temperature distribution in an automobile engine block. The progress "stops" in 1985 because the time to solve this problem has fallen below the time needed just to display the solution.

	1945	1955	1960	1965	1970	1985
Compute Time	300 Million years	200 years	6 hours	30 minutes	24 hours	2 sec.
Memory Used	800 Million	5 Million	50 Million	300,000	170,000	50,000

Table 2. Time required to compute the temperature distribution in an automobile engine block using a machine with a 1 Gflops processor. For each year it is assumed that the best algorithm known at that time is used.

The algorithmic progress seen in Table 2 for solving partial differential equations is repeated in many problem areas [Odlyzko, 1995]. The rate and amount of progress depends on the problem area and is especially dependent on the size of the problem. Thus algorithms for multiplying two 10-digit numbers have been sped up by a factor of perhaps 100, but is is implausible to hope for algorithmic speed up here by a factor of 100 million. As problem sizes increase, the effects of algorithm speed up increase and then it is plausible to hope for speeding up the simulation of an entire automobile by a factor of 10^{20} even if this is not realistic for a simple 3D temperature computation.

People are also familiar with lack of increase in programming power over the past 40 years. The evolution of programming languages and aids might have increased programming power by a factor of 5 or 10, some

people would say the increase is much less. An increase of 2.7% per year yields a factor of 3 increase over 40 years, 6% per year yields a factor of 10 increase. This slow increase has totally changed the financial nature of computing. In 1960 the cost of buying and operating hardware dominated computing costs. Today the cost of programming dominates computing costs.

2 PSEs FOR COMPUTATIONAL SCIENCE

2.1 Definitions and Goals

The potential for very high level and powerful problem solving systems for science was recognized very early [Culler and Fried, 1963, Rice and Rosen, 1966, Klerer and Reinfelds, 1968]. Inadequate computing power made such systems unfeasible until the 1980s when serious work began [Rice and Boisvert, 1985, Ford and Chatelin, 1987]. In April 1991 a research conference was held which issued a long report [Gallopoulos et al., 1992] exploring this field, see also [Gallopoulos et al., 1994a]. The definition of a problem solving environment (PSE) that emerged is

> *A PSE is a computer system that provides all the computational facilities necessary to solve a target class of problems*

These facilities include advanced solution methods, automatic or semiautomatic selection of solution methods and ways to easily incorporate novel solution methods. Furthermore, PSEs use a language natural for the problem class and allow users to solve problems without specialized knowledge of the underlying computer hardware or software.

The goal of PSE technology is to exploit the existing enormous hardware and algorithm power to deliver "cheap and fast" problem solutions. PSEs are one important component of the effort to remedy the lack of increase in programming power for science and engineering.

This definition of a PSE can be summarized by the formula

```
PSE = Natural Language + Solvers + Intelligence + Software bus
```

where software bus represents the computing infrastructure (machines, systems, networks,...) tying everything together. From this definition we see that a PSE must have a natural (for the application) language component where the computer system "understands" the user rather than the other way around. Common mathematic and scientific notations are to be used along with graphical and geometric tools of all types. This language forms the user interface of the PSE. The PSE must also have a collection of solvers as there will not be one single solver that is best (or even effective) for all problems in an interesting class of science and engineering problems. Individual solvers usually have, in turn, parameters (e.g., step size, tolerance, priority, polynomial degree, etc.) which adapt the solver to the particular computation at hand. The PSE must have considerable intelligence and knowledge about the target class of problems, the solvers used and the underlying computing services. The PSE user is not expected to be able to answer questions such as

- What is the best solver to use for this problem?
- What step size is needed to achieve the accuracy required?
- Which computer should be used?
- Where is the data needed for this computation?

The high level and complexity of the PSE performance goals leads to a large and complex software system. This topic is explored further in Sections 4 and 5. The basic conclusion is that the software bus uses a layered architecture and a software components methodology. Future PSEs will use essentially all the resources of future computing technologies, e.g., net-centric facilities, dynamic and collaborative computations, knowledge discovery and learning, software reuse, human-machine infraction, etc.

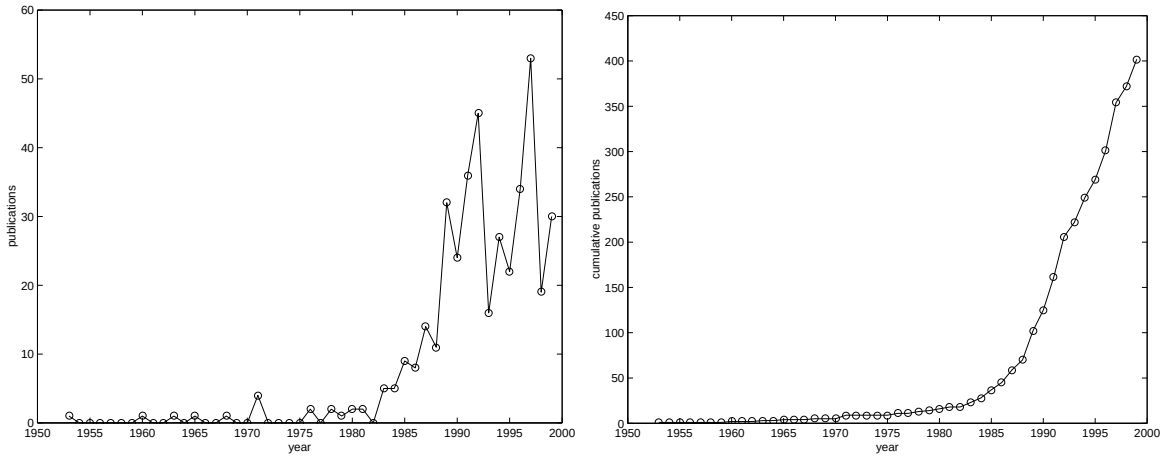

Figure 1: Publication dates from the bibliography of Gallopoulos (a) by year, (b) cumulative from 1963 to 1999.

3 STATE OF THE ART IN 2000

Problem solving environments is now a well established methodology for computational science. Rather than present a detailed review of PSEs, we first present a set information sources and briefly summarize their content. Then we present reviews of a few typical PSEs for computational science.

3.1 Information Sources for PSEs

The PSE web site

`http://www.cs.purdue.edu/research/pses`

provides a definition of PSEs, a reading list (11 items), a list of conferences on PSEs (12 items, 10 since 1991) and a list of PSE projects in computational science. The latter have a brief description, contact information and link to the PSE web site. Most of these projects are to create actual PSEs (31 projects), some are to build infrastructure (11 projects) and a few are related symbolic systems (4 projects).

A book edited by Houstis, Gallopoulos, Rice and Bramley [Houstis et al., 2000b] contains a set of 28 papers which cover a broad range of work on PSEs. Of particular interest in this book is the extensive bibliography of 415 papers related to PSEs. Figure 1 shows a plot of the paper's publication dates, both by year and cumulative. This figure shows how the PSE field blossomed once the computing power (and other infrastructure) became available in about 1990 to make PSE practical.

Another indication of the state of the PSE field is the number of government research initiatives that either focus on PSEs or have PSE research as a large component. In the United States there have been such initiatives from the National Science Foundation and the Department of Energy.

3.2 Example PSEs

Brief descriptions are given of several PSEs, all but one are listed on the PSE web site. The criterion for selection is primarily to illustrate the variety of PSEs in computational science and engineering.

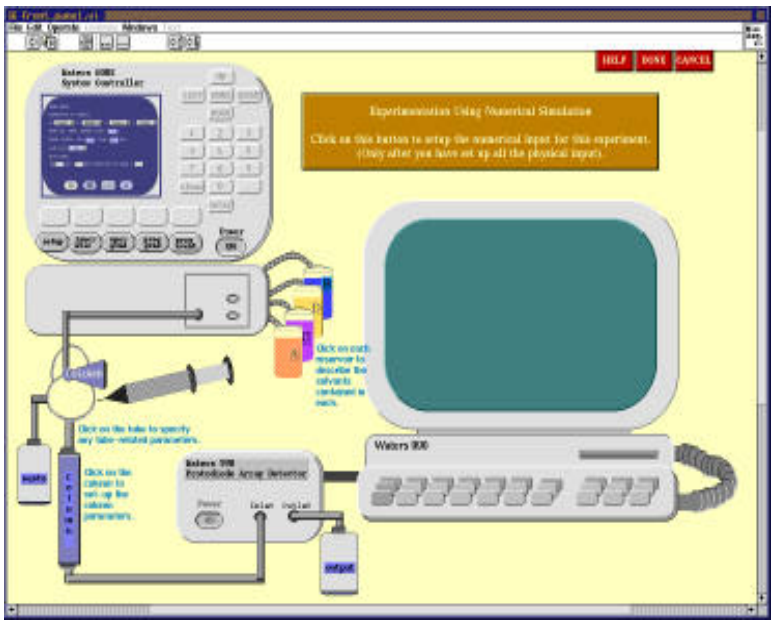

Figure 2: The user interface of the BioSoftLab PSE. The machine controlling the equipment is shown at the upper left.

3.2.1 BioSoftLab

This PSE [Catlin et al., 2000] is an example where the simulation and problem solving are completely hidden from the user. The user interface is a schematic (Figure 2) of the laboratory equipment for experiments about the separation of products (foods, drugs) to purify them. The display includes the console of the two machines that control the experiment and in one mode the user can actually operate the machines using the PSE. Another mode of use is complete simulation of the experiment so that computed rather than measured values are shown on the machine screens. The third mode is to run the equipment and the simulation simultaneously and to use the simulation to help control the equipment. The need for simulation is that the purity of the final product is satisfactory before the purity can be measured. Thus, in real applications the separation process is always run longer than necessary. The simulation allows the separation to be stopped just when the final product parity will be satisfactory. More information is available at

`http://www.cs.purdue.edu/research/cse/softlab/softlab-vlabs/softbiolab/`

3.2.2 ELLPACK

The original PSE [Rice and Boisvert, 1985] allowed a user to define an elliptic partial differential equation on a general 2D domain in a mathematical notation and to have it solved by simply naming the methods to be used. This PSE was implemented completely in portable Fortran and distributed world wide. ELLPACK has evolved in several directions: (i) to provide a graphical user interface, (ii) to include more elliptic PDE solvers and to add parabolic/hyperbolic PDE solvers, (iii) to support parallel computation (domain decomposition and parallel numerical algorithms), (iv) to have a web server interface, and (v) to have a highly modular and layered software architecture (see Figure 3), (vi) to have several sophisticated visualization options, (vii) to have an integrate performance measurement, and (viii) to provide 3D solvers [Houstis et al., 1998b]. The total lines of code in these systems has grown to almost 2 million, and easy portability has been lost as user services have been enhanced. ELLPACK and its descendents are described at

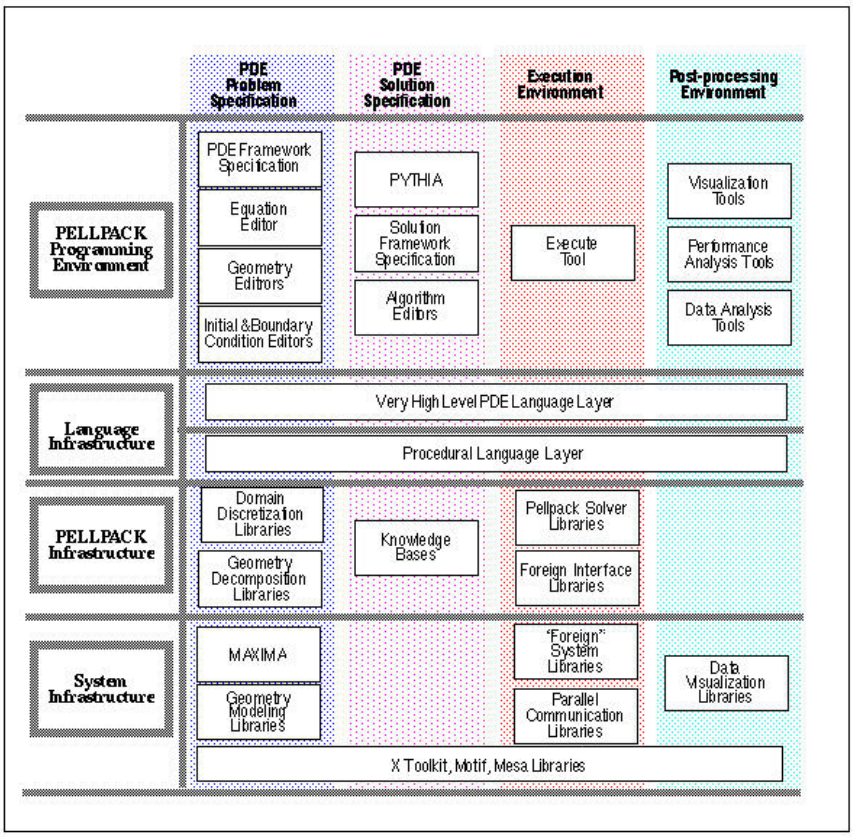

Figure 3: A block diagram of the PELLPACK software architecture.

`http://www.cs.purdue.edu/ellpack`

3.2.3 HiQ

HiQ is a PSE from National Instruments which helps analyze, visualize and document the solution of science and engineering problems. It is organized as a notebook that is created as a problem is solved, sophisticated 2D and 3D displays are created in the notebook and can be modified directly. The analysis is made using the HiQ-Script language, by direct access to MatLab (see below) or by access to user created objects which encapsulate existing software. HiQ-Script include a wide array of built-in mathematical, numerical, statistical, visualization and utility functions. HiQ is integrated into the Windows operating system so generic facilities like Word, Excel and Power Point are available to the notebook environment. For more information visit the web site

`http:/www.ni.com/hiq/`

3.2.4 MATLAB

.

MATLAB is a PSE from Mathworks based on a programming language for matrix computations. Its origin traces back to the early 1970s when it was purely an interactive matrix/vector language. It has

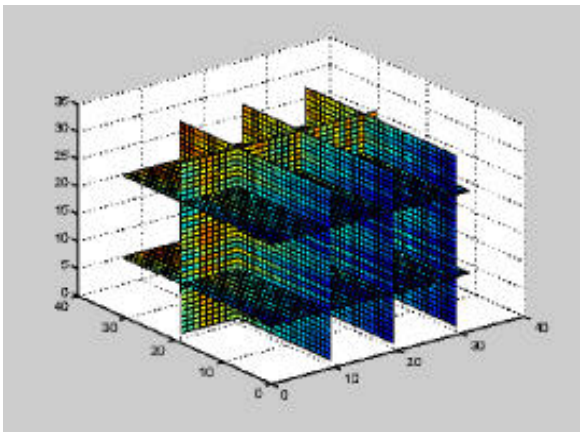

Figure 4: Matlab has specialized interpolation and data gridding functions that operate specifically on multidimentional data.

evolved by adding good graphics and toolboxes to broaden the problem domains it can address. It is available on most computing platforms and it includes toolboxes for control systems, databases, finance, fuzzy logic, image processing, optimization, partial differential equations, statistics, symbolic mathematics, etc. MATLAB is probably the most widely used PSE for computational science, many university programs assume its availability to students. It provides interfaces to a number of generic facilities (web browsers, and Fortran, Active X) and in the Windows environment uses the word processor to provide a notebook facility. Figure 4 shows the result of a short MATLAB program to create a plot of a data gridding operation:

```
>> x1=-2*pi:0.2:0; x2=2*pi:0.2:4*pi; x3=0:0.2:2*pi;
>> [x1,x2,x3] = ngrid(x1,x2,x3);
>> z = x1 + exp(cos(2*x2.^2)) + sin(x3.^3);
>> slice(z, [10,20,30], 20, [10,25] )
```

For more information visit the web site

`http://www.mathworks.com/products/matlab`

3.2.5 PDEase2D

PDEease is a PSE from Macsyma for solving a wide range of 2D partial differential equation problems. It uses finite element PDE solvers for elliptic, parabolic and hyperbolic equations which may be linear or nonlinear. The user interface is mathematically oriented, one provides equations, formulas for boundaries, boundary conditions and parameter values. The grid and time step are selected automatically based on an internal error estimator, which can also be accessed by the user. The user interface is through the Macsyma symbolic mathematics system and visualization also uses the Macsyma facilities. The Windows version of PDEase provides a notebook interface and access to some generic Windows facilities. PDEase2D provides over 150 sample problems from many application areas. A sample result is shown in Figure 5. For more information see the web site

`http://www.macsyma.com/pdease2D.html`

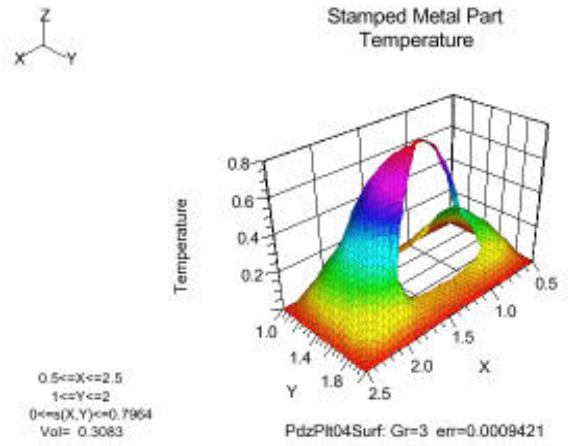

Figure 5: Graph of the temperature computed by PDEase2D for a heat flow problem.

4 PSE PARADIGMS

4.1 Software Reuse

One of the design objective of the future class of PSEs will be the use of scalable libraries as building blocks for creating seamless scientific applications that enable users to define the application I/O via a multi-media user interface that uses a geographically distributed infrastructure consisting of heterogeneous resources where some of them could be HPC servers. There are different definitions of software reuse. Some definitions refer to interoperability between software modules or components to describe reuse. Others address the portability of components from one application or platform to another. Interoperability and cross-platform portability certainly are among the incentives for code reuse. To realize this objective we must develop tools that enhance reuse and enable layered approaches to application development. In the case of sequential applications CORBA and OLE help integrate software components and give interoperability between packages and languages. Java delivers them with its Write Once, Run Anywhere capability. Specifically, the JavaBeans component model and the standard Java APIs developed collaboratively within the software industry, together form a powerful spring-board for software reuse across three dimensions: reuse across applications, reuse across tools, reuse across architectural layers.

In parallel programming there are no commercial tools that support software reuse other the various BLAS. The Purdue group has developed some methodologies and tools to support software reuse in the context of PDE stationary computations. Specifically, they have formulated two parallel frameworks to reuse the discretization components of sequential elliptic PDE solvers. These parallel reuse methodologies follow the "divide–and–conquer" computational paradigm and are based on a non–overlapping decomposition of the problem domain. They primarily enable rapid prototyping of parallel applications by reusing legacy scientific code. They are aimed at significantly speeding up the existing computational PDE models while preserving the integrity of the legacy code and thereby maintaining confidence in the computed solutions. More general tools are needed in this area.

4.2 Natural Languages

Several attempts have be made to increase the level a user specifies the problem and the associated computations. Some results of these efforts include specialized language interfaces associated with PSE like ALPAL, ELLPACK, MatLab, Mathematica, Maple, CAPSE, etc. The dream here is to develop a language that

allows the user to specify an "outline" of the problem and the associated computations. The difficulty associated with these languages is the specification of the geometric artifacts involved in the problem definition. There are several environments that handle this part. Unfortunately, they tend to be closed systems and the communication between a customized language processor and them is difficult to impossible. Purdue has made an effort in this direction to define a PDE language [Weerawarana, 1994]. We append its specification for completeness. It is based on a well known system called Maxima. The integration of natural language interfaces to application environments will be one of the biggest breakthroughs. However, we feel that it is several decades away.

4.3 Collaboratory Problem Solving

The predicted growth of computational power and network bandwidth suggests that computational modeling and experimentation will be one of the main tools in big and small science. In this scenario, computational modeling will shift from the current single physical component design to the design of a whole physical system with a large number of components that have different shapes, obey different physical laws and manufacturing constraints, and interact with each other through geometric and physical interfaces. For example, the analysis of an engine involves the domains of thermodynamics (gives the behavior of the gases in the piston-cylinder assemblies), mechanics (gives the kinematics and dynamic behaviors of pistons, links, cranks, etc.), structures (gives the stresses and strains on the parts) and geometry (gives the shape of the components and the structural constraints). The design of the engine requires that these different domain-specific analyses interact in order to find the final solution. The different domains share common parameters and interfaces but each has its own parameters and constraints. We refer to these multi-component based physical systems as multi-physics applications (MPAs). The realization of the above scenario, which is expected to have significant impact in industry, education, and training, will require the development of new algorithmic strategies and software for managing the complexity and harvesting the power of the expected HPCC (High Performance Computing and Communication) resources. It will require PSE technology to support programming-in-the-large and reduce the overhead of HPCC computing. In the near future we will see the development of generic frameworks for the numerical simulation of multi-physics applications and the development of enabling theories and technologies needed to support and realize this framework in specific applications. The MPSE (Multiphysics PSE) is the software implementation of this framework. It is assumed that its elements are discipline-specific problem solving environments. The MPSE design objective is to allow the natural specification of multi-physics applications and their simulation with interacting PSEs through mathematical and software interfaces across networks of computational resources.

4.4 Netcentric Computing

Rapid advances in modern networking technologies and commodity high performance computing systems are leading the field of computing in a new paradigm referred to as network-based computing (NC). This paradigm views a large number of geographically distributed computer resources such as PCs, workstations, Symmetric Multi-processors (SMPs) and Massively Parallel Processing (MPP) systems connected through a high speed network as a single meta-computer or computational grid.

The scientific computing community established in the 1970s the concept of the software library and introduced procedures for testing and disseminating such artifacts that have evolved in international standards. Current information technologies allow the easy development and integration of GUI interfaces, domain specific textual and visual languages, visualization libraries, portable computational libraries, knowledge bases and other related technologies. These technologies already allow the user to exploit the power of the hardware resources while reducing the overhead of specifying and visualizing the results of a simulation. These developments have led to the concept of a Problem Solving Environment (PSE) that promises to provide industrial scientists and engineers with environments and seamless integration mechanisms allowing them to spend more time doing science and engineering rather than doing "computing". Now, the NC paradigm

promises to put the PSE technology at the figure tips of any scientist and engineer anytime and anywhere. The first NC paradigm is the so-called remote computing. In this scenario, the software usage is not viewed as a commodity but as service. The computational service provider will offer to the user all the resources that s/he needs to solve his/her problem in some "natural" form. In this context, we will see the development and implementation of the virtual scientific library and computational server concepts. The realization of these paradigms require the study of the related issues of accounting and Quality of Services (QoS). In the context of internet computing, locating/selecting the computational resources/services out of high-level specifications of the user problem/application becomes an important issue. We predict that the concept of portal for NC will emerge and evolve into a popular scientific computing e-business.

The process of prototyping is part of every scientific inquiry, product design, and learning activity. The new economic realities require the rapid prototyping of manufactured artifacts and rapid solutions to problems with numerous interrelated elements. This, in turn, requires the fast, accurate simulation of physical processes and design optimization using knowledge and computational models from multiple disciplines (multi-physics and multi-scale models) in science and engineering. Thus, the realization of rapid multidisciplinary prototyping is the new grand challenge. In these applications the software is often distributed geographically and the various groups involved have limited knowledge of all software components used to simulate the atomic elements involved in the prototyping of a composite artifact. In this application scenario the natural computational resource is a "computational grid" that connects the needed distributed hardware and software resources used to simulate the elements of the artifact. Some variations of this application scenario have been addressed in the context of cluster computing (LAN based computational grids). However, the mapping and execution of such computations on an internet computational grid involving hundreds of heterogeneous nodes distributed across enterprises is still at the proof of concept stage; it involves many unresolved research issues. Moreover, it is unclear whether the current distributed computing technologies can support scalable, robust, and secure internet computing. We envision the design and implementation of PSE frameworks specialized for network computing based primarily on the proxy computing paradigm, utilizing primarily existing middleware technologies.

For addressing the requirements of adaptive computations, we will see the utilization of mobile computing or code shipping paradigms. Research issues involved in the realization of the future PSE for NC that include methodologies for the incorporation of legacy code, fault tolerance, load balancing, user-interface design, and security. The internet has long been used for communication. Until recently, there has been little use of the network for actual computations. This situation is changing rapidly and will have enormous impact on the future. The PSE technology described in this report will have a significant role to play in these developments. It is clear that the old slogan that "The Network is the Computer" is becoming a reality. It will not only change the way we live, work, learn, and communicate with each other, but it will change the way we do computational science and electronic prototyping.

4.5 Intelligence in Computational Science

Complex problems, whether scientific, engineering or societal, are most often solved today by utilizing public domain or commercial libraries or some form of problem solving environments. The existing software is characterized by a significant number of parameters, affecting its efficiency and applicability, that must be specified by the user. This complexity is significantly increased by the number of parameters associated with the execution environment. Furthermore, one can create many alternative solutions of the same problem by selecting different software that implements the various phases of the computation. Thus, the task of selecting the best software and the associated algorithmic/hardware parameters for a particular problem or computation is often difficult and sometimes even impossible. In [Houstis et al., 1995a] the Purdue group had proposed an approach for dealing with this task by "processing" performance data obtained from "testing" software. The validation of the approach is described in [Houstis et al., 1997b] and realized by an implementation (referred as KBAS) restricted to a specific performance evaluation study. This experience made us realize the high level of complexity involved in the algorithmic discovery of knowledge from per-

formance data and the management of these data together with the discovered knowledge. To address the complexity issue together with scalability and portability of this approach, we are presenting *a knowledge discovery in data bases* methodology for testing and recommending scientific software which uses existing database and recommendation technologies. Its implementation is referred to with the acronym PYTHIA-II [Houstis et al., 2000a].

Given a problem description from a known class of problems, along with some performance criteria, PYTHIA II provides a knowledge based technology for the selection of the most efficient software/machine pair and estimation of associated parameters involved. Due to its ability to make recommendations by combining attribute-based elicitation of specified problem and matching them against those of predefined "dense" population of similar type of problems, we classify PYTHIA II as a recommender system. We believe that the technology of recommender systems or portals as it is known in the e-business word will be embedded in many scientific PSE in the near future.

5 PSE SOFTWARE ARCHITECTURE

The PSE software architecture is very much influenced by the structure of the so called "Problem Solving Process". In this section we consider the problem solving process used when computation is the primary technique for solving some problem. We consider activities from both the user's viewpoint and the "system's" viewpoint. What we refer to as the system is the sum total of all software/hardware which is involved in computationally solving the problem.

Initially, the user must define the problem to the system. In a PSE, this specification is declarative (i.e., only indicates the required information and not what to do with it or how to do something with it), symbolic (i.e., in some abstract form) and in terms that are natural to the problem domain. At this level only the essential features of the problem are specified; there is no indication of how it is to be solved or any other solution scheme related information provided. If a PSE already exists for solving this type of problems, then the user must interact directly with the PSE and solve the problem. Since in this paper we are concerned with the situation where a PSE is not already available, we will ignore this case from now on.

Suppose that while there is no existing PSE for solving the specific problem, there does exist a (large) collection of problem solving components (i.e., a workbench), including those that are needed to solve the current problem. Then, the user must first combine some of these components to form a custom PSE and then apply it to solve the problem at hand. In this case, the user must be able to "browse" the available components, "select" the appropriate ones, and "connect" them to form a custom PSE. Then, to solve the problem, the user transfers the declarative problem specification to the PSE and interacts with the PSE appropriately. The final scenario is when only some of the components needed to solve the problem are already available from a component database. The other needed components must be custom developed. Thus, the user must "build" the components by writing program code in some form and then connect the components together to form the PSE, as in the previous scenario.

How does the problem solving environment thus assembled finally solve the problem? The process involved can typically be decomposed to the following five stages:

- *Declarative problem specification*: As described above.
- *Computational script*: The problem specification must be transformed to some solution algorithm which, when run, will result in solving the problem. The computational script can be viewed as a high level, pseudo-code specification of this algorithm. In some instances, this script may not be explicit, but it is usually present nevertheless.
- *High-level programming language program*: The computational script must be executed by either interpreting it directly or by translating it to a program in some traditional high level programming language.

- *Problem solvers* (libraries, servers): The problem solvers are the components that do the real work for solving the problem. These are invoked from the high level language or by the script interpretation process.

- *OSs/networks/utilities*: The lowest level is the traditional computing platform on which the problem solvers execute.

While PSEs are a special type of software, they do share many properties with other large scale integrated software environments such as Microsoft Office. Clearly, there already exist many software frameworks which support the development of such systems, including Microsoft OLE, OpenDoc, CORBA, and JavaBEANS. There is also a large body of existing work on distributed communication environments such as RPC, MPI, and Glish. However, none of these systems completely addresses the somewhat unique infrastructure needs of problem solving environments. The Purdue group has proposed a kernel for building PSEs referred as PPK [Weerawarana et al., 1994]. A recent project called PSEWare by a consortium lead by Indiana University is researching kernels for building PSEs as well and appears to use an architecture very similar to PPK, at least at the design specification level.

The goal of the Purdue PSE Kernel is to develop a software kernel that can be used to build PSEs that support the problem solving process described above. This goal is realized by the following components:

- *PSE architecture*: PPK defines and supports a powerful, extendable PSE architecture. All the components of PPK and the resulting PSEs assume and support this architecture.

- *PSE component database and browser*: The component database and browser allow users to view existing PSE components as well as to install new components into the component database.

- *PSE composer*: The composer is essentially a very high-level programming facility where the user programs in a data flow manner. Composing components selected from the component database is expected to be the likely approach to building custom PSEs.

- *Electronic notebook*: The problem solving process typically involves multiple steps and a solution path determined by trial-and-error. The electronic notebook serves as the central recording and access environment for monitoring, controlling and steering this process. An embedded programming environment allows users to program (or script) a sequence of operations to be performed during a problem solving process.

- *Object manager*: The problem solving process involves many objects including the problem input objects, the solution objects and the output objects. The object manager is the database which manages these components for the user and for the PSE components.

- *PSE component builder*: The components (tools) of a PSE are what provide the real computing muscle to it. The component building process involves using the appropriate data object standards for input and output and implementing the component's internal functionality using whatever toolkits are provided by the environment.

- *Language kernel*: The language kernel is a toolkit with which one can build the application specific language with which the user may interact with the PSE.

- *Software bus*: The software bus is the underlying "glue" that supports the integration and operation of the PPK framework outlined above.

The architecture of PSEs supported by PPK is also based on the levels of computation as Figure 6. That is, a PSE build with PPK will have roughly the layers present illustrated in Figure 6. Each layer in this model contains a collection of tools. The tools interact within a level via well-defined object interfaces.

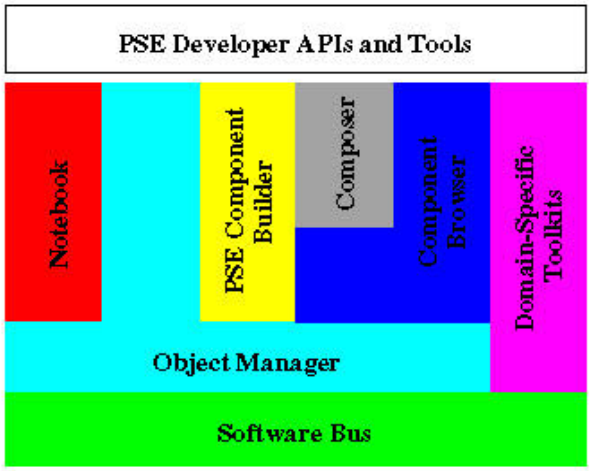

Figure 6: Layered Architecture of PPK.

For example, at the high-level programming language level, the word "tool" may refer to a function while "object interface" may refer to some "standard" data structures and function signatures. Interaction across levels occurs in some abstract specification language or by some automated process that translates a set of tools and objects from one layer to the representations used at a different layer. Clearly the key then is to allow the integration of the various pieces to form the comprehensive, integrated system that provides problem solving facilities to the user. Software that provides such integration frameworks is typically called "middleware" and PPK can be viewed at as a middleware system for PSEs.

Building a PSE using PPK requires one to customize it by configuring the core components (software bus, notebook and object manager) of PPK appropriately and by developing any necessary tools. The result is a customized framework into which application-specific components can be integrated conveniently. The description of the implementation of the PPK is given in [Weerawarana et al., 1994].

6 FUTURE TRENDS

The PITAC (Presidential Information Technology Advisory Committee) report [White House, 1999] published recently in the United States, devotes a section to high–end computing. Following we present the findings and the recommendations published in PITAC report for this area. As a result of this report the Information Technology funding initiative has been launched by NSF this year. One can view these findings and recommendations as defining the future trends in computational science in general. They call for an extension of the HPCC initiative driven by petaflop performance machines. The experience with HPCC suggests that a new generation of software and algorithms must be developed to take advantage of the computational resources that will be provided by the new generation hardware. A critical observer will conclude that the only lasting breakthrough produced by the HPC initiative at the software level was the MPI communication standard and several algorithmic technologies.

The realization of the PITAC recommendations will require the development of new algorithmic strategies and software for managing the complexity and harvesting the power of the expected high performance resources; it will require PSE technology to support programming-in-the-large and reduce the overhead of HPC computing.

6.1 PITAC Findings for High–End Computing

The successes of the HPCC program have led to widespread use of computational simulation and modeling as a means to understand natural phenomena and to explore and optimize engineering designs. Over the next few decades, computation will become even more essential to the Nation's fundamental and applied science and engineering endeavors, both civilian and military. Currently, the extreme high-end of computing is done under the auspices of focused mission agencies such as the Department of Energy (DOE), while widespread academic research is primarily sponsored by the National Science Foundation (NSF). Both for the sake of fundamental scientific research and to enable applications to benefit from the research, the research community needs access to systems at the leading edge of capability. The power of these systems should be comparable to that of systems–like the DOE Accelerated Strategic Computing Initiative (ASCI) systems available to the mission agencies. Furthermore, if the scientific community is to continue to benefit from high–end computing, it is important to ensure that there are enough computer and computational scientists to collaborate in making leading-edge computation an effective research tool.

Finding: *High–end computing is essential to science and engineering research.*

Finding: *High–end computing is an enabling element of the United States national security program.*

Finding: *New applications of high-end computing are ripe for exploration.*

Finding: *Innovations are required in high-end systems and application–development software, algorithms, programming methods, component technologies, and computer architecture.*

Finding: *The high–end computing capability available to the civilian science and engineering community is falling dangerously behind the state of the art.*

6.2 PITAC Recommendations for High–End Computing

In high-end computing as in other areas of information technology, we need more fundamental research – the kind of ground breaking, high–risk research that will provide the ideas and methods for new disciplinary paradigms a decade or more in the future. Our greatest needs are improving systems software and algorithm–level software support at the high end, exploring innovative architectures and devices, and making it possible for the academic research community and the Federal Government to conduct essential research and development on computers of the highest possible performance.

Recommendation: *Fund Research into Innovative Computing Technologies and Architectures.*

Recommendation: *Fund R&D on software to improve the performance of high-end computing.*

Recommendation: *Drive high–end computing research by trying to attain a sustained petaflops & petaflops on real applications by 2010 through a balance of software and hardware strategies.*

Recommendation: *Fund the acquisition of the most powerful high–end computing systems to support science and engineering research.*

7 FUTURE PROJECTIONS FOR 2010 & 2025

7.1 The Next Decade

We project that

- Future computational paradigm for "small" and "large scale" numerical computations will be based on a cost–effective pooling of local– or wide-area-networked resources including small and large scale machines.
- Remote computing services and servers will support primarily the future numerical computing for research, education, and electronic prototyping

- The next computational challenge is the class of multidisciplinary applications and the concurrent design of composite artifacts.
- The human-computer interfaces and interaction will resemble more the interfaces of today's simulators for commercial aircraft, military systems, and games. Full-immersion environments are beginning to explore these areas.
- The future PSEs will look like sophisticated CAD systems today where the "elements" will instantiate both geometric and physical properties.
- Natural languages will be part of every interface of a library.
- Portals will manage most of the software and compute servers.

The above projections coincide with many of the objectives of the 5th European framework for research, the vision articulated in the report PITAC (see above) and the US IT2 information technology funding initiative for the twenty-first century. These visions are consistent with the current impact of internet and web in many human activities. The outcome of this vision has the potential to empower the average engineer and scientist, affect the prototyping of manufacturing artifacts, and revolutionize engineering and science education. It will not only impact 'big" science but the R&D of many SMEs (small manufacturing enterprises) in the manufacturing sector. The realization of this vision in the area of computational science and engineering can not be achieved with the existing software and algorithmic infrastructure only. The current remote computing infrastructure does not scale, it is not secure, it can not handle highly interactive applications, the re–use of legacy software for remote computing needs rethinking, it is not robust, since it involves heterogeneous and non-fault tolerance computing resources, it does not guarantee quality of services. All these issues are magnified in the context of the proxy computing paradigm.

7.2 PSEs in 2025

Projections for 25 years in the future are not expected to be accurate, they only present reasonable possibilities. The combined effect of the better hardware and algorithms will be to deliver at least a million fold increase in computing power. For complex applications (e.g., simulating an airplane or living organism), the increase in computing power will be much more, from a billion fold to a trillion fold.

PSEs will be one of six new enabling technologies that will emerge to exploit this computing power. The role of PSEs will be to deliver this power to people in a form they can easily use. The other five enabling technologies are:

- *Computational intelligence and discovery.*
- *Hybrid environments of real and simulated devices.*
- *Total high speed information access and recall.*
- *Micro/nano machines and computers.*
- *The natural languages of science and engineering used for "programming" applications.*

These six technologies support and interact with one another. No doubt there will be several more that we do not foresee at this time.

The impact of these advances on science and engineering will be very large. There are several obvious directions of evolution that will occur as a result of increasing hardware and algorithm power.

- *Numerical models will become more accurate and more complete.*

  More grid points, more particles, longer times, etc., will be used. More accurate (higher order) numerical models will be used. More physical, chemical, etc., phenomena will be incorporated into the models. Three and four dimensional simulations will become commonplace.

- *Optimization of designs through simulation will become commonplace.*

- *Devices will become controlled by computers.*

  These will range from the commonplace (cars, stoves, lawn mowers, ...) to large industrial operations (blast furnaces, pharmaceutical production, farming) to very sophisticated (medical procedures, telescopes, airplanes, human organs, ...)

- *The computing environment will become highly parallel and highly distributed.*

  Most people will not be aware of which server is supplying the computer cycles, algorithm power or information they are using.

To explore the impact of these advances on science and engineering, we pose and answer five questions.

- *Will people solve all science/engineering problems just by asking a PSE*?

  No. Recall that a PSE encapsulates problem solvers that are *well understood and routine*. Thus a PSE might appear to be magic for a high school student or someone from the 1960s, but the frontiers of science is full of problems that are not well understood and whose solution is not routine.

- *Will there be enough power to solve all those problems we do understand*?

  No There are problems whose computational requirements are so huge that they will remain open for many decades.

- *What will a university student be doing with a PSE*?

  For example, a civil engineering student could be given the specifications of a river and surrounding terrain and asked to design a bridge across the river. The specifications could be difficult, e.g., cliffs, deep water, fast current, wide river, etc. The student is asked to evaluate not only design and location, but also competing technologies, e.g., reinforced concrete, steel structures, lightweight composite materials, ... The student will specify the design, location, and materials, then the PSE would complete the details and finally compute the actual designs. The student (or professor) would then evaluate the student's skill on the basis of cost, strength, esthetics, durability, etc.

- *What will be a routine use of a PSE for a working engineer or scientist*?

  Two examples are: (1) Specification of a building: requirements and a rough design will result in a complete set of plans (including details of electrical, heat, access, etc.). These will be reviewed by the architect (and the customer) using virtual reality, and several rounds of review will result in the final plans. (2) Determination of chemical reactions: The initial chemicals and environment are specified along with the final chemical compound. The PSE will explore reactions that produce the desired compound. Such a computation has very high complexity, so many "long" chains of reactions are found only with great effort (even for 2025 computers). The chemist can look at initial results, determine those that seem promising, ask to obtain 'intermediate" compounds that seem relevant, and gradually uncover a long chain.

- *What will be a difficult challenge for a PSE in 2025*?

  As complexity grows slowly, the computing power required grows rapidly. Further, as complexity grows the probability of involving a poorly understood physical phenomenon increases. These are common

physical phenomena that, so far, cannot be simulated from the first principles because these principles are unknown. Examples include ordinary friction between materials, the breakup of a stream of liquid into a spray of drops, the propagation of a crack in a material, the chemical reactions that occur when gasoline burns. Consider, for example, the design and simulation of a gas turbine engine. It has perhaps 30,000 parts and there are many extreme physical conditions present. The latter requires very fine numerical models for accurate simulation. Even if every physical and chemical process in the engine were understood perfectly, it is likely that a PSE (or any other computing environment) could produce an accurate (say, one part in a thousand for all interesting variables) simulation in the year 2025 after days or weeks of computation. But there are several physical phenomena in an engine that are not understood well enough, e.g., how cracks form and propagate in the blades, how the fuel spray develops, how the jet fuel burns.

8 RESEARCH ISSUES FOR 2000–2010

Research issues in eleven general areas are listed and briefly discussed; these range from studying the nature of problem solving to finding better PSE architectures and construction methodologies. Specific research issues are italicized and usually phrased as questions.

8.1 Models of Problem Solving

Communication between humans and PSEs is critical in problem solving. Currently PSEs follow the traditional scientific practice of requesting that problems be stated in a somewhat standard form. Indeed, achieving this in a natural way is still a challenge for many PSEs. A basic research issue is: *Which functions should be performed by humans and which by the PSE*? People normally has a large context in mind when they start on problem solving; they use "fuzzy" sketches and back-of-the-envelope analyses to get started. *How can context and ad hoc, fuzzy information be incorporated into the PSE*? *Are there easier and more powerful ways to communicate about scientific problem solving*?

Problem solving is often an iterative process involving changing specifications, strategies, and goals. *How can computer power in information processing be used to track the problem solving process, to organize it for review by the human, and for analysis by the PSE*? *How can relevant information from PSE operations be gathered for algorithm developers and PSE architects*? *How can data mining be applied to this information to improve the effectiveness of the PSEs*? Addressing these research issues probably requires a systematic framework for representing the problem formulation and solving process.

8.2 Recommender Systems for Knowledge Discovery

Recommender systems are to provide advice and options in problem formulation, algorithm selection, computer resource allocation and similar areas. To deliver the advice is much easier than to gather or create the knowledge which is the basis of the advice. Systems are needed for many particular domains ranging from well understood ones like the execution time performance of direct linear solvers to poorly understood ones like the effectiveness of general, large-scale optimization systems. Detailed performance and behavior models are needed for all aspects of problem solving, from algorithms to computing resources to network behavior to problem formulations. *Which domains have the best prospects for effective recommendations*? *Which data mining* (*knowledge discovery*) *techniques are most effective*? *How is the underlying data for knowledge discovery collected and represented*? *Can ordinary PSE use provide useful data for the knowledge bases*? *Can generic recommender systems be constructed analogous to systems for symbolic computations or visualization*? *Can the models of problem solving be used to anticipate a person's needs for advice*? *Can recommender systems be used to improve a person's skills as well as assisting with particular issues*? It is

clear that providing some degree of intelligence to PSEs is a long term research program that is now in its very early stages.

8.3 Collaborative Problem Solving

Collaboration is very common in problem solving involving multiple people, software and hardware. PSEs should support tracking and recording the actions, activities and information of problem solving. They should also facilitate the collaboration process and assist in coordinating disparate algorithms, scientific disciplines and computer systems. *How can the record of collaborations be represented and effectively presented to the people*? *What levels of control over the collaboration should be given to multiple users*? *to computational grids*? *to the PSE*? *How can PSEs for different problem domains and from different disciplines be brought into collaborative problem solving*? Collaboration support must be an integral part of PSE design. Views of the state of a collaboration must be available for diverse people to study, annotate and communicate. *How can master control be managed in such computations*? *How is authority delegated*? *How are dynamic people, software and hardware resources identified and managed*?

8.4 Interoperability and Openness

Future computational science applications will be very complex, operating in dynamic and heterogeneous environments. PSEs must be able to use highly interoperable problem-solving service providers. *How can generic desktop tools be integrated into PSEs*? There will be a paradigm shift in scientific software as the emphasis changes from "bricks" to "glue", from atomic problem solving components to the methodology for combining components dynamically and in unanticipated ways. PSEs that completely define their interfaces in advance cannot operate and evolve in future computational environments. *How can this plug-and-play paradigm be implemented*? *How can interface standards be both efficient and completely open*? *What form should these standards take*? It is estimated that making code reusable increases the cost of development by 30% to 50%. *How can the computational science community create reward mechanism* (*financial or otherwise*) *that encourage both commercial and non-profit developers to create reusable software components.* Truly enormous computational science software systems will be constructed in the 2000–2010 period. These programs cannot be written from scratch, not even major governments can afford this expense. *How are PSEs to be designed so they can be integrated smoothly as components of such systems*? *Can we avoid dictating that common data structures be used throughout such systems*? *What are the costs of optimizing the interface connections used in object-oriented components*? *Can compiler-like methods be used to leverage the plug-and-play paradigm and yet provide the efficiency needed in enormous computations*?

8.5 Sharing Large, Standard Components

The library concept has been very successful for encapsulating algorithms for widespread reuse. We have just seen that reuse must be extended to larger and more complex components such as those for data visualization, symbolic mathematics, geometric design, data mining, and document preparation. A new vision for computational science is to combine multiple computing paradigms dynamically to raise the power in problem solving. Perhaps the most important single part of this vision is to make the large components completely and easily reusable. Once we can do this then we will know better how to reuse more specialized components. *How can component developers collaborate and share software creations*? *What economic incentives can be used to overcome the "not invented here" syndrome that prevents leveraging existing software investments*?

8.6 Netcentric Computing

Implicit in the previous area is that problem solving will be netcentric, the computing environment will be completely distributed with its details invisible to ordinary users or programs. The components of PSEs will themselves be distributed to use remote instruments and software servers interchangeably and automatically. *How will components of a PSE migrate over the network*? *What are the costs and constraints in migration*? *How are suitable servers* (*hardware or software*) *identified*? *How do PSEs compensate for components that fail remotely, perhaps with no warning or diagnostic information*? *How can practical user interfaces be provided for such PSEs*? It is plausible that within the 2000–2010 decade that all the large standard computational science PSE components (as well as many PSEs) will be available as network servers. *How can existing "single machine" PSEs exploit netcentric computing*? When code goes to a machine of unknown characteristics (e.g., shared memory multiprocessor, distributed memory multiprocessor, single vector processor, etc.) it is unlikely that the code can benefit from the advantages that various characteristics provide. *How can algorithms and/or PSEs adapt to changing characteristics of the hardware*? *the software*? One advantage of service providers is that they have their own code, but there can still be constraints or assumptions about the impact/output for the user. *How can network servers for computational science interface with PSEs which use a variety of data structures and representations*?

8.7 Validation of Computations

The validation of computational science PSE results is critical, yet minimal effort is spent by users or PSEs to check that the answers are correct. We still primarily rely on checking against experimental results. Yet the costs of experiments is increasing while the cost of computation is decreasing. As a result, more and more results are being accepted as correct without independent validation. Errors can occur in problem formulation, computational methods, low level software systems and in hardware. Some mechanisms already exist for error checking in robust PSEs and sometimes automatic compensation is possible. However, some errors cannot be handled automatically by currently known mechanisms and users are less and less able to understand errors that occur deep in ever larger software systems. *What mechanisms can provide high reliability for PSE results*? *How can PSEs best assist in validating results*? *Are there one or a few general approaches to validation that are applicable to a wide variety of computational science problems*? *Or must each type of problem and subproblem be studied separately*? It is often difficult in a single-platform PSEs even to determine what failed. *Can future PSE information gathering subsystems be used to help identify the location and nature of errors*? *How best can PSEs implement check-pointing and restart approaches for error correction*? *Can the check-point information be a high level representation of the computation instead of just a "memory dump"*? *Is it possible to create a generic validation, error handling, exception handling and check-point/restart component for use in PSEs*?

8.8 Performance Management

Performance management includes the usual issues of selecting appropriate computing resources for a PSE. In computational science there is a deeper component of performance management, namely how to tailor the computation at hand for high performance on the available resources. Small changes in the formulation of a problem or the representation of some aspect of it can have huge effects on performance. Computational science involves many applications where high performance is critical and yet PSEs tend to hide information about the sources of poor performance from users. For example, in a symbolic mathematics computation, the expansion of an algebraic expression can cause an invisible explosion in the size of the underlying data structures. The user only learns that the system has ran out of memory or that the wait for results is unacceptable. Usually no clue is given about how to improve the performance. *What mechanism can be developed so uses can get answers to questions like*:

- *Is there a problem with the performance*?

- *What are the resource limits that cause performance problems*?
- *When should the computation be finished*?
- *How much better will the PSE perform if more resources are provided.*

The tracking of the problem solving process provides a natural place for performance information to be collected and made available to the user. *How can low level performance information be collected and aggregated for high level display to a user*? As PSE components migrate over the network, one expects changes in performance. But one also expects some type of "portability of performance", that is, performance should not change dramatically just by switching to another computing resource of similar capability. Thus, if one goes from using 20 processors and 50 gigabytes of memory to using 10 processors and 15 gigabytes of memory, one expects the performance to decrease some, perhaps by a factor of 1.5 to 4, but one does not expect performance to decrease by a factor of 100 without some obvious explanation. *How can one measure performance portability for PSEs*? *Can software and its underlying algorithms be modified to provide better performance portability*?

8.9 The Languages for Basic Science

Science has developed a standard language (with many sub-dialects for different sub-fields) for itself. A PSE should use this language, this is part of the definition of a PSE. Some parts of this language are widely implemented (symbolic mathematics, numerical algorithms), but some are not. In particular, computer languages for geometry are quite primitive and, no surprise, complex geometry is the single most difficult aspect of computational science. There are no language facilities that reflect the common geometry operations that people use in problem solving. Consider, for example, the simple phrases "use this shape", "join these curves", "cut this curve into 5 smooth pieces", "make these 2 points corners so as to fit the data", and "let x be a point near the center of this domain". Mesh and grid generators are used to discretize geometry but this software is very complex, less than completely robust, and sometimes provides unnatural results. *Is is possible to create a generic, natural geometry language and system*? *If so, how can it be done*? *If not, what are the best alternatives*? *Can we even make mesh/grid generators in 3D which are generic, robust and reflect natural expectations*? *Can such geometry languages be integrated into language systems for symbolic and numerical mathematics*?

A second shortcoming of PSE language is the lack of flexibility and provisions for "fuzziness" that people use in natural communication. Part of this is that PSE languages for science have not tried to adopt the capabilities that are available in other areas. Part of this is that no one has tried to systematize the languages that scientists use in describing science problems. *Can these natural languages attributes be incorporated into a language for basic science*? *If so, how difficult is it*? A language for basic science should include "everything" up to about the second year of university mathematics. This material forms a common foundation for communication among scientists. In addition, the various subfields of science, e.g., physics, geology, and mechanical engineering, each has a language (call it jargon) of their own of a similar level. A PSE involving mechanical engineering should use the jargon natural for this field. *Can the jargons of the science subfields be processed by PSEs*? The long term objective of these basic science languages is to participate linguistically in a discussion with scientists using a blackboard. *Can we incorporate into a PSE language the capability to understand a blackboard discussion*?

If such a basic science language were available in PSEs, then we might be able to carry out a long term dream in computational sciences: reprogram the legacy codes that people depend on now. This reprogramming is very unlikely to ever be done if traditional languages (i.e., Fortran, C, Java,...) are to be used.

8.10 Education and Levels of Abstraction

Science and engineering education starts from very basic roots and grows, through branching, to higher and higher levels. This occurs both on a very broad scale (all of science, say) and on much smaller discipline or technical specialty scales (e.g., solving partial differential operations or designing internal combustion engines). This tree structure should be reflected in the PSEs that support science and engineering. Users should be able to build on their skills and move easily to more advanced and specialized PSEs. *How can the transition from "basic" PSEs to "advanced" PSEs be made easy for users*? *Would connections between PSEs at different levels hinder or help the use of PSEs*? *Would basic PSEs need to be extremely stable in content and operation in order to fill an education role*? *If so, who would decide when and how they would change*? Closely related to education levels are levels of abstraction in a PSE. One might, at a high level, just want to see the basic equations involved. Another person might want to examine the lower levels such as (1) how the equations are discretized, (2) what are the parameters of an iteration, (3) how are the unknowns distributed among the machines and their memories. *How can one represent the different levels so one can easily move between them*? *Are the representations tied to specific languages or terminologies*? *Are there execution performance penalties associated with multiple levels of abstraction*?

8.11 PSE Architecture and Construction

Once all the above considerations are evaluated and decisions are made about what a PSE should be, then one has to decide on how to build it. The PSE architecture clearly involves both "bricks" and "glue" but there must also be a larger scale, overall plan. *Are there PSE frameworks suitable for a variety of applications*? *Are some architectures more suited for software component reuse than others*? *How can one build a problem specific PSE dynamically*? *Who will create the infrastructure and generic middleware needed for PSE construction*? *Are domain specific standards, data structures, and interfaces likely to be in conflict with their generic counter parts*? *Can one create PSE factories*? *Are there ways to put components together to make a PSE work properly and then later systematically optimize the efficiency of the interfaces between components*? Using a component based, multi-level approach to PSE architecture is natural. *Are there significant differences if some components are "real" instead of "simulated"*? *Can PSEs adjust the scale of physical devices so that people can obscure and control activities that are very fast, very slow, very big, very small, etc.*? *What are the economics of using real versus simulated components*?

Acknowledgement.
We thank Ann C. Catlin for her extraordinary help in preparing this report.

9 REPORT BIBLIOGRAPHY

References

[Adve et al.,] Adve, V. S., Bagrodia, R., Brown, J. C., Deelman, E., Dube, A., Houstis, E. N., Rice, J. R., Sakellariou, R., Surdaram-Stukel, D., Teller, P. J., and Vernon, M. K. POEMS: End-to-end performance of large parallel adaptive computational systems. *IEEE Trans. Soft. Eng., to appear.*

[Agrawal and Shim, 1996] Agrawal, R. and Shim, K. (1996). Developing tightly-coupled data mining applications on a relational database system. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, Portland, Oregon.

[Akarsu et al., 1998] Akarsu, E., Fox, G., Furmanski, W., and Haupt, T. (1998). Webflow — high-level programming environment and visual authoring toolkit for high performance distributed computing. In *Proceedings of Supercomputing'98*. (http://www.npac.syr.edu/users/haupt/WebFlow/papers/SC98/INDEX.HTM).

[Akarsu et al., 1999] Akarsu, E., Fox, G., Haupt, T., Kalinichenko, A., Kim, K.-S., Sheethaalnath, P., and Youn, C.-H. (August 1999). Using gateway system to provide a desktop access to high performance computational resources. HPDC8 Conference.

[Bhatia et al., 1997] Bhatia, D., Burzewski, V., Camuseva, M., Fox, G., Furmanski, W., and Premchandran, G. (1997). Webflow – a visual programming paradigm for web/java based coarse grain distributed computing. *Concurrency: Practice and Experience*, 9:555–578. (http://tapetus.npac.syr.edu/iwt98/pm/documents/).

[Boisvert et al., 1991] Boisvert, R., Howe, S., and Kahaner, D. (1991). The guide to available mathematical software problem classification system. *Communications in Statistics – Simulation and Computation*, Vol.20(4):pages 811–842.

[Boisvert and Rice, 1996] Boisvert, R. and Rice, J. (1996). From scientific software libraries to problem solving environments. *IEEE Comp. Sci. & Engr.*, 3:44–53.

[Boisvert et al., 1979] Boisvert, R. F., Rice, J. R., and Houstis, E. N. (1979). A system for performance evaluation of partial differential equations software. *IEEE Transactions on Software Engineering*, SE–5(4):418–425.

[Bratko and Muggleton, 1995] Bratko, I. and Muggleton, S. (1995). Applications of inductive logic programming. *Comm. ACM*, 38(11):65–70.

[Casaletto et al., 1969] Casaletto, J., Pickett, M., and Rice, J. (1969). A comparison of some numerical integration programs. *SIGNUM Newsletter*, 4(3).

[Catlin et al., 2000] Catlin, A., Gaitatzes, M., Zidu, M., Rice, E. H. J., Wang, N.-H., Markus, S., and Weerawarana, S. (2000). Softlab: A virtual laboratory framework for computational science. In E.N. Houstis, e. a., editor, *Enabling Technologies for Computational Science*. Kluwer Academic Press.

[Catlin et al.,] Catlin, A., Weerawarana, S., E.N., H., and Gaitatzes, M. The pellpack user guide. Technical report, Dept. of Computer Sciences, Purdue University. to appear, 2000.

[Cornea-Hasegan et al., 1994] Cornea-Hasegan, M., Costian, C., Marinescu, D., Martin, I., and Rice, J. (1994). Towards problem solving environments for high performance computing. *High performance computing '94*, pages 354–366.

[Culler and Fried, 1963] Culler, G. and Fried, B. (1963). An on-line computing center for scientific problems. *IEEE Pacific Computer Conference*, page 221.

[Dodson et al., 1968] Dodson, D., Miller, P., Nylin, W., and Rice, J. (1968). An evaluation of five polynomial zero finders. Technical Report CSD-TR-24, Dept. Comp. Sci., Purdue University.

[Dongarra et al., 2000] Dongarra, J., Foster, I., Fox, G., Kennedy, K., Torczon, L., and White, A. (2000). *The CRPC Handbook of Parallel Computing*. Morgan Kaufmann Publishers.

[Dongarra and Grosse, 1987] Dongarra, J. and Grosse, E. (1987). Distribution of mathematical software by electronic mail. *Communications of the ACM*, Vol. 30:pages 403–407. URL: `http://www.netlib.org/`.

[Dyksen et al., 1984] Dyksen, W., Houstis, E., Lynch, R., and Rice, J. (1984). The performance of the collocation and galerkin methods with hermite bicubics. *SIAM Journal of Numerical Analysis*, 21:695–715.

[Dyksen et al., 1988] Dyksen, W., Ribbens, C., and Rice, J. (1988). The performance of numerical software methods for elliptic problems with mixed boundary conditions. *Numer. Meth. Partial Differential Eqs.*, 4:347–361.

[Dzeroski, 1996] Dzeroski, S. (1996). Inductive logic programming and knowledge discovery in databases. In Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R., editors, *Advances in Knowledge Discovery and Data Mining*, pages 117–152. AAAI Press/MIT Press.

[Fayyad et al., 1996a] Fayyad, U., Haussler, D., and Stolorz, P. (1996a). Mining scientific data. *Comm. ACM*, 39(11):51–57.

[Fayyad et al., 1996b] Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996b). From data mining to knowledge discovery: an overview. In Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R., editors, *Advances in Knowledge Discovery and Data Mining*, pages 1–34. AAAI Press/MIT Press.

[Ford and Chatelin, 1987] Ford, B. and Chatelin, F. (1987). *Problem Solving Environments for Scientific Computing*. North Holland, Amsterdam.

[Fox and Furmanski, 1998] Fox, G. and Furmanski, W. (1998). High performance commodity computing in the grid blueprint for a new computing infrastructure. Morgan-Kaufmann Publishers, Inc., San Francisco.

[Fox et al.,] Fox, G., Furmanski, W., Ozdemir, H., and Pallickara, S. High performance commodity computing on the pragmatic object web. (http://tapetus.npac.syr.edu/iwt98/pm/documents/).

[Gaffney and Houstis, 1992] Gaffney, P. and Houstis, E. (1992). *Programming Environments for High-Level Scientific Problem Solving*. North Holland, Amsterdam.

[Gallopoulos et al., 1994b] Gallopoulos, E., Houstis, E., and Rice, J. (1994b). Computer as thinker/doer: problem-solving environments for computational science. *IEEE Computational Science and Enginerring*, Vol. 1(2):pages 11–23.

[Gallopoulos et al., 1992] Gallopoulos, E., Houstis, E., and Rice, J. (May 1992). Future research directions in problem solving environments. Technical report, Dept. of Computer Sciences, Purdue University, 92-032.

[Gallopoulos et al., 1994a] Gallopoulos, E., Houstis, E., and Rice, J. (Summer 1994a). Computer as thinker/doer: Problem-solving environments for computational science. *IEEE Computational Science & Engineering*, pages 11–21.

[Giarratano, 1991] Giarratano, J. C. (1991). *CLIPS user's guide, version 5.1.* NASA Lyndon B. Johnson Space Center.

[Haupt et al.,] Haupt, T., Akarsu, E., and Fox, G. Web based metacomputing. *Special Issue on MetaComputing for the FGCS International Journal on Future Generation Computing Systems.*

[Haupt et al., 1999a] Haupt, T., Akarsu, E., Fox, G., Kalinichenko, A., Kim, K.-K., Sheethalnath, P., and Youn, C. (April 1999a). The gateway system: Uniform web based access to remote resources. High Performance Computing and Networking'99, Amsterdam.

[Haupt et al., 1999b] Haupt, T., Akarsu, E., Fox, G., Kalinichenko, A., Kim, K.-S., Sheethalnath, P., and Youn, C.-H. (June 1999b). The gateway system: Uniform web based access to remote resources. In *Proceedings of ACM Java Grande Conference.*

[Hollander and Wolfe, 1973] Hollander, M. and Wolfe, D. (1973). *Non-parametric Statistical Methods.* John Wiley and Sons.

[Houstis et al., 1995a] Houstis, C., Houstis, E., Rice, J., Varadaglou, P., and Papatheodorou, T. (1995a). Athena: a knowledge based system for //ELLPACK. *Symbolic–Numeric Data Analysis and Learning*, pages 459–467.

[Houstis et al., 1997a] Houstis, E., Joshi, A., Rice, J., Weerawarana, S., and Ramakrishnan, N. (1997a). Intelligent networked scientific computing. volume Vol. 4, pages pages 785–790. Wissenschaft and Technik Verlag.

[Houstis et al., 1978] Houstis, E., Lynch, R., and Rice, J. (1978). Evaluation of numerical methods for elliptic partial differential equations. *Journal of Comp. Physics*, 27:323–350.

[Houstis and Rice, 1980] Houstis, E. and Rice, J. (1980). An experimental design for the computational evaluation of elliptic partial differential equation solvers. In Delves, M. and Hennell, M., editors, *The Production and Assessment of Numerical Software*, pages 57–66. Academic Press.

[Houstis et al., 1998a] Houstis, E., Rice, J., Weerawarana, S., Catlin, A., Gaitatzes, M., Papachiou, P., and Wang, K. (1998a). Parallel ELLPACK: a problem solving environment for PDE based applications on multicomputer platforms. *ACM Trans. Math. Soft.*, 24(1):30–73.

[Houstis et al., 1998b] Houstis, E., Rice, J., Weerawarana, S., Catlin, A., Gaitatzes, M., Wang, K., and Papachiou, P. (1998b). PELLPACK: A problem solving environment for PDE-based applications on multicomputer platforms. *ACM Trans. on Math. Soft.*, 24(1):30–73.

[Houstis and Rice, 1982] Houstis, E. and Rice, J. R. (1982). High order methods for elliptic partial differential equations with singularities. *Inter. J. Numer. Meth. Engin.*, 18:737–754.

[Houstis et al., 1995b] Houstis, E., Weerawarana, S., Joshi, A., and Rice, J. (1995b). The pythia project. In *Neural, Parallel and Scientific Computations*, pages pages 215–218. Dynamic Pub.

[Houstis et al., 1997b] Houstis, E. N., , Weerawarana, S., Rice, J. R., Joshi, A., and Houstis, C. (1997b). PYTHIA: a knowledge based system to select scientific algorithms. *ACM Trans. Math. Soft.*, 23:447–468.

[Houstis et al., 2000a] Houstis, E. N., Catlin, A. C., Rice, J. R., Verykios, V., Ramakrishnan, N., and Houstis, C. (2000a). Pythia-ii: A knowledge/database system for managing performance data and recommending scientific software. *ACM Trans. Math. Soft., to appear.*

[Houstis et al., 2000b] Houstis, E. N., Rice, J. R., Gallopoulos, E., and Bramley, R. (2000b). *Enabling Technologies for Computational Science.* Kluwer Academic Publishers.

[Houstis et al., 1990] Houstis, E. N., Rice, J. R., and Vichnevetsky, R., editors (1990). *Intelligent mathematical software systems.* North–Holland.

[Houstis et al., 1992] Houstis, E. N., Rice, J. R., and Vichnevetsky, R., editors (1992). *Expert systems for scientific computing.* North–Holland.

[Houstis et al., 1994] Houstis, E. N., Rice, J. R., and Vichnevetsky, R., editors (267-503, 1994). *Third international conference on expert systems*, volume 36 of Special triple issue of. *Math. Comp. Simulation.*

[James and Rice, 1967] James, R. and Rice, J. (1967). Experiments on matrix attributes and SOR success. Technical Report CSD-TR-9, Dept. Comp. Sci., Purdue University.

[John and Lent, 1997] John, G. H. and Lent, B. (1997). Siping from the data firehose. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, pages 199–202, Newport Beach, California.

[Joshi et al., 1996a] Joshi, A., Weerawarana, S., Ramakrishnan, N., Houstis, E., and Rice, J. (1996a). Neuro–fuzzy support for PSEs: a step toward the automated solution of PDEs. *IEEE Computational Science and Engineering*, vol.3(1):44–56.

[Joshi et al., 1996b] Joshi, A., Weerawarana, S., Ramakrishnan, N., Houstis, E., and Rice, J. (1996b). Neuro–fuzzy support for PSEs: a step toward the automated solution of PDEs. *Special Joint Issue of IEEE Computer & IEEE Computational Science and Engineering*, Vol. 3(1):pages 44–56.

[Kitano and Shimazu, 1996] Kitano, H. and Shimazu, H. (1996). H. kitano and h. shimazu. In Leake, D., editor, *Case-based reasoning: experiences, lessons, and future directions*. AAAI Press/MIT Press, Menlo Park, CA.

[Klerer and Reinfelds, 1968] Klerer, M. and Reinfelds, J. (1968). *Interactive Systems for Experimental Applied Mathematics*. Academic Press.

[Kohavi, 1996] Kohavi, R. (1996). MLC++ developments: data mining using MLC++. In Kasif, S. e. a., editor, *Working Notes of the AAAI-96 Fall Symposia on 'Learning Complex Behaviors in Adaptive Intelligent Systems'*, pages 112–123. AAAI Press.

[Kolodner, 1993] Kolodner, J. (1993). *Case-based reasoning*. Morgan Kaufmann, San Francisco.

[Konig and Ullrich, 1990] Konig, S. and Ullrich, C. (1990). An expert system for the economical application of self-validating methods for linear equations. In *Intelligent Mathematical Software Systems*, pages 195–220, North-Holland.

[Moore et al., 1990] Moore, P., Ozturan, C., and Flaherty, J. (1990). Towards the automatic numerical solution of partial differential equations. In *Intelligent Mathematical Software Systems*, pages 15–22, North-Holland.

[Muggleton, 1995] Muggleton, S. (1995). Inverse entailment and PROGOL. *New Generation Computing*, Vol. 13:pages 245–286.

[Muggleton and Feng, 1990] Muggleton, S. and Feng, C. (1990). Efficient induction of logic programs. In Arikawa, S., Goto, S., Ohsuga, S., and Yokomori, T., editors, *Proceedings of the First International Conference on Algorithmic Learning Theory*, pages 368–381. Japanese Society for Artificial Intelligence, Tokyo.

[Muggleton and Raedt, 1994] Muggleton, S. and Raedt, L. D. (1994). Inductive logic programming: theory and methods. *Journal of Logic Programming*, 19(20):629–679.

[Odlyzko, 1995] Odlyzko, A. (1995). The future of integer factorization. *Cryptobyte*.

[Olston et al., 1998] Olston, C., Woodruff, A., Aiken, A., Chu, M., Ercegovac, V., Lin, M., Spalding, M., and Stonebraker, M. (1998). Datasplash. In *Proceedings of the ACM-SIGMOD conference on management of data*, pages 550–552, Seattle, Washington.

[Quinlan, 1986] Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.

[Ramakrishnan, 1997] Ramakrishnan, N. (1997). *Recommender systems for problem solving environments*. PhD thesis, Dept. of Computer Sciences, Purdue University.

[Ramakrishnan et al., 1998] Ramakrishnan, N., Houstis, E., and Rice, J. (1998). Recommender Systems for Problem Solving Environments. In Kautz, H., editor, *Working notes of the AAAI-98 workshop on recommender systems*. AAAI/MIT Press.

[Ramakrishnan and Rice, 2000] Ramakrishnan, N. and Rice, J. (2000). Gauss: An on-line recommender system for one-dimensional numerical quadrature. *ACM Trans. Math. Soft., to appear*.

[Resnik and Varian, 1997] Resnik, P. and Varian, H. (1997). Recommender systems. *Communications of the ACM*, Vol. 40(3):pages 56–58.

[Rice, 1976a] Rice, J. (1976a). Algorithmic progress in solving partial differential equations. *SIGNUM Newsletter*, page 21.

[Rice, 1983] Rice, J. (1983). Performance analysis of 13 methods to solve the Galerkin method equations. *Lin. Alg. Appl.*, 53:533–546.

[Rice, 1990] Rice, J. (1990). Software performance evaluation papers in TOMS. Technical Report CSD-TR-1026, Dept. Comp. Sci., Purdue University.

[Rice, 1992] Rice, J. (1992). *Numerical Methods, Software and Analysis, Section 10.2C.* Academic Press, 2nd Edition.

[Rice and Boisvert, 1985] Rice, J. and Boisvert, R. (1985). *Solving Elliptic Problems using ELLPACK.* Springer-Verlag, New York.

[Rice and Rosen, 1966] Rice, J. and Rosen, S. (1966). Napss - a numerical analysis problem solving system. *Proc. ACM National Conference*, pages 51–56.

[Rice, 1969] Rice, J. R. (1969). A set of 74 test functions for nonlinear equation solvers. Technical Report CSD-TR-34, Dept. Comp. Sci., Purdue University.

[Rice et al., 1981] Rice, J. R., Houstis, E., and Dyksen, W. (1981). A population of linear, second order, elliptic partial differential equations on rectangular domains. *Mathematics of Computation*, 36:475–484.

[Rice, 1976b] Rice, R. (1976b). The algorithm selection problem. *Advances in Computers*, 15:65–118.

[Richardson et al., 1998] Richardson, T., Stafford-Fraser, Q., Wood, K., and Hopper, A. (1998). Virtual network computing. *IEEE Internet Computing*, 2(1):33–38.

[Riesbeck, 1996] Riesbeck, C. (1996). What next? the future of CBR in postmodern AI. In Leake, D., editor, *Case-Based Reasoning: Experiences, Lessons, and Future Directions.* AAAI Press/MIT Press, Menlo Park, CA.

[Riesbeck and Schank, 1989] Riesbeck, C. and Schank, R. (1989). *Inside case-based reasoning.* Lawrence Erlbaum, Hillsdale, NJ.

[Sarawagi et al., 1998] Sarawagi, S., Thomas, S., and Agrawal, R. (1998). Integrating association rule mining with databases: alternatives and implications. In *Proceedings of the ACM-SIGMOD Conference on Management of Data*, pages 343–354, Seattle, Washington.

[Stonebraker and Rowe, 1986] Stonebraker, M. and Rowe, L. A. (1986). The design of POSTGRES. In *Proceedings of the ACM-SIGMOD Conference on Management of Data*, pages 340–355.

[Verykios et al., 1998] Verykios, V. S., Houstis, E. N., and Rice, J. R. (1998). A knowledge discovery methodoloy for the performance evaluation of scientific software. Technical Report TR-98-031, Dept. Comp. Sci., Purdue University.

[Watson, 1977] Watson, I. (1977). *Applying case-based reasoning: techniques for enterprise systems.* Morgan Kaufmann.

[Weerawarana, 1994] Weerawarana, S. (1994). *Problem solving environments for partial differential equation based applications.* PhD thesis, Dept. of Computer Sciences, Purdue University.

[Weerawarana et al., 1994] Weerawarana, S., E.N., H., Catlin, A., Rice, J. R., Gaitatzes, M., Crabill, C., and Drashansky, T. (1994). Ppk: Towards a kernel for building pses. Technical report, Dept. of Computer Sciences, Purdue University.

[White House, 1999] White House (1999). *Presidential Information Technology Advisory Committee*. Washinton, DC.

10 APPENDIX

In this report we have made the claim that PSEs will play a significant role in supporting netcentric computing in the near future. To illustrate this vision we append the description of three such PSEs: NetSolve, WebFlow and WebPDELab. This material is taken from the book [Dongarra et al., 2000].

A NetSolve: Network-enabled Solvers

A.1 Motivation and History

The current software usage model entails three basic phases: i) obtaining the software (locating and/or purchasing, investigating licensing, import and export restrictions, etc.) ii) installing the software, and finally, iii)using the software. In addition to these mundane tasks, maintenance of the software system is also necessary to ensure that the latest versions are being used and to attain patches and bug fixes. The NetSolve project, underway at the University of Tennessee at Knoxville and the Oak Ridge National Laboratory had very humble beginnings. Its original goal was to alleviate domain scientists of this tedium when trying to use numerical software, particularly on multiple platforms. And so it began, a sole interface, Matlab, with access to solver routines from the LAPACK library. The first major release of NetSolve was in 1995.

Today, NetSolve has evolved into one of the leading research projects in the area of Grid computing. Its various interfaces provide uniform access to an assortment of software toolkits and libraries. These libraries come from a diverse sphere of influence, ranging from mathematical solvers to more eclectic domains like microbiology and image visualization. Currently at version 1.2 released in the fall of 1998, the NetSolve system receives continual enhancements and feature upgrades. A beta version of 1.3 was available in the fall of 1999 with a non-beta distribution slated for release in the early spring of 2000. The NetSolve software is freely available and can be downloaded, along with additional documentation and related papers, at:

`http://www.cs.utk.edu/netsolve`

A.2 The NetSolve Philosophy

As research scientists continue efforts to harness as much computational resources and power as possible, NetSolve continues to position itself in the midst of it all. As the system becomes more enhanced and, unfortunately, more complicated, there are certain fundamentals that we try to maintain in our NetSolve system. The first and foremost is that the system is very easy to deploy and use. The thing that should be furthest from the thoughts of the middleware user is how to incorporate software that in and of itself, he is not interested in. What he really wants is the software and hardware resources that the middleware makes available to him. Although simplicity is of the essence, the interface must still be intricate enough to meet the full needs of its users. NetSolve exerts much effort to do the impossible; be simplistic yet complicated at the same time. We have managed to achieve what we believe to be an adequate, if not perfect, balance between the two.

Another perspective of NetSolve concerns the integration and usage of other grid computing infrastructure. As opposed to re-inventing the wheel, we try to leverage on the accomplishments of the grid computing community at large. But all our designs ensure that we are dependent on none of these systems. Should their resources be available in any of our users' domains, we gladly take advantage of them, but the NetSolve system can and most often does stand alone without infrastructure native to the NetSolve system.

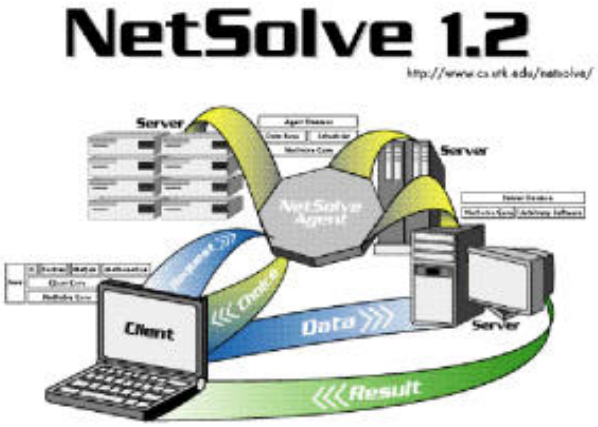

Figure 7: The NetSolve System.

A.3 NetSolve Infrastructure

A.3.1 The Big Picture

As depicted by Figure 7, NetSolve is of a client/agent/server design in which the client issues requests to agents who allocate servers to service those requests; the server(s) then receives inputs for the problem, does the computation and returns the output parameters to the client. The NetSolve client-user gains access to limitless software resources without the tedium of installation and maintenance. Furthermore, NetSolve facilitates remote access to computer hardware, possibly high-performance supercomputers with complete opacity. That is to say that the user does not have to possess knowledge of computer networking and the like to use NetSolve. In fact, he/she does not even have to know remote resources are involved. Features like fault-tolerance and load balancing further enhance the NetSolve system. In the sections below, we offer a brief discussion of the three aforementioned components.

A.3.2 The Client Interfaces

A major concern in designing NetSolve was to provide several interfaces for a wide range of users and flexibility. NetSolve can be invoked via C, Fortran, MATLAB and Mathematica interfaces (the Mathematica interface is available only on Win32 client platforms). In the past, we supported a Java Application Programming Interface (API) and web-based Graphical User Interface (GUI) and are undergoing efforts to upgrade these interfaces to the current version of NetSolve. Another concern was to implement interfaces that were as simple as possible yet while being meticulous enough to allow the user to control execution of the remote procedure as much as possible. For all the interfaces involved, we provided two basic functions. The first allows for synchronous or blocking requests that do not return until remote execution is complete (or failure is detected). The second is a more non-traditional asynchronous or non-blocking request that returns immediately giving the user a "handle" which he can use to query the readiness of and/or obtain the solution set. In addition to these, we provide in the API functions to do error reporting. We also provide the capabilities to dynamically query a NetSolve system to receive information about either the hardware or software resources. These are used primarily to determine which problems are available and the number, type and description of the input and output parameters each problem requires. In the case of command line interpreters like Matlab and Mathematica, these are in the forms of functions. For the compiled interfaces, C and Fortran, these are compiled executables.

A.4 The NetSolve Agent

A.4.1 The Agent as a Database

Keeping track of what software resources are available and on which servers they are located is perhaps the most fundamental task of the NetSolve agent. The agent keeps a database that maps software resources to hardware components in the NetSolve system thus having an complete picture of the capabilities of both the individual servers and the NetSolve system on a whole. The agent can report this information to the client via the interfaces (see above) which will then aid the user in setting up his problem on top of the NetSolve middleware. The protocol which NetSolve uses to maintain this database is fairly straightforward: Upon initialization, a new server sends a "problem description" for each problem it can solve to the agent it was configured to register with. This description, among other things, contains the location of the server and the particulars of the function(s) being contributed. Eventually the server is integrated into the system and can be used to service users' requests.

A.4.2 The Agent as a Resource Broker

In order to expeditiously service user requests, it is necessary that the agent use certain criteria to choose the best-suited computational server for each incoming request. There are two basic choices: i) static scheduling where at compile time the agent is programmed to use some a priori scheme like round-robin scheduling and ii) dynamic scheduling where the agent uses run-time information to decide which server component should be used to service a request. NetSolve uses the latter. In actuality, it combines both static and dynamic information. Static information includes speed and number of processors and complexity of the solution algorithm. Dynamic information includes server loads, network delays and transmission rates, and input data sizes. The agent then uses this information to rank the servers from best to worse. This list is passed to the client and the client makes its request to each server in turn until either the problem has been successfully solved or the list has been exhausted.

A.4.3 Fault Tolerance and Load Balancing

The protocol described in the resource brokerage section above has its primary goal rooted more in high throughput than balancing load amongst the servers. Consider a scenario where there is a high performance supercomputer acting as a NetSolve server along with other stand alone mediocre workstations, with several simultaneous requests. Then most of the requests will be sent to the supercomputer (as long as the supercomputer is determined to be the component that will finish the service quickest). In a scenario where all server resources are of essentially the same rating, however, this same paradigm will balance the load evenly amongst the servers since this scenario will then yield highest throughput. For fault tolerance, NetSolve ensures that a user request will be completed unless every single resource capable of servicing the request has failed. As explained above, when a client sends a request to a NetSolve agent, it receives a sorted list of computational servers to try. When one of these servers has been successfully contacted, the computation starts. If the contacted server fails during the computation, then another server is contacted and the computation restarts. This entire procedure is transacted independently of, and possibly unbeknownst to, the client user. Though effective, this primitive fault-tolerant mechanism needs to be enhanced. In the next section where we discuss current developments, we describe our research to employ more advanced fault-tolerance.

A.4.4 The Computational Server

One of the challenges when building the NetSolve system was to design a suitable model for the computational servers. For the user to be able to invoke numerical software directly through our servers, three major features seemed to emerge as mandatory for the servers:

Uniform access to the software: The NetSolve servers should present the interfaces with an illusion of uniformity amongst the various integrated packages. The critical point is to try as much as possible to maintain high levels of consistency amongst and within the different sets of subroutines/functions provided to the user. This allows the user to focus on the particular problem he is trying to solve rather than the peculiarities of the software package he is using to aid his investigations. This also eliminates long learning phases when using new functionalities.

Configurability: The server should not be confined to any particular software. It was, therefore, essential that we provide a framework that permitted the addition of functionality to a computational server. This framework should be as intuitive as possible and general so that one can integrate any software toolkit with NetSolve servers and make them accessible to the client interfaces.

Pre-installation: As stated in the last section, we wished to ease the user of the burden of software installation. Therefore, in the NetSolve paradigm, the client user is not responsible for installing any software directly. The software is made available via the NetSolve servers in a read-to-use fashion. It is also possible in some scenarios for the NetSolve system to dynamically install and compile routines without any intervention at the user-level. The NetSolve server addresses and successfully resolves all these issues. Mainly through the use of what we call a problem description file (PDF), the server can be configured with a set of pre-installed software libraries to provide uniform access to the sub-routines provided. The PDF in essence describes the particulars of a function to be added. Some of the information that is described in this file are the name to be given to the problem, the calling sequence to the NetSolve client interface, the libraries or archives containing the underlying functions being integrated and other things. The PDF really describes a wrapper that is used to receive or send input and output parameters from and later back to the client interface. In the midst of these networking transactions is a call to the routine from the underlying library to actually do the service that was requested. Although network interactions are involved, neither the client nor the writer of the PDF needs be concerned with this. The NetSolve system carefully encapsulates and hides these interactions from the user. These wrappers are parsed and compiled into source codes which are compiled with the library archives into NetSolve specific executables. The appropriate executable is initiated by the server daemon whenever it needs to service a client request.

A.5 Some Applications of NetSolve

In this section, we give a brief description of the integration of NetSolve into grid computing systems. We describe some of the applications that have taken advantage of what NetSolve has to offer. We later discuss in Section C, some of the other metacomputing resources that NetSolve has used to leverage itself.

A.5.1 MCell

MCell is a general Monte Carlo simulator of cellular micro physiology. MCell uses Monte Carlo diffusion and chemical reaction algorithms in 3D to simulate the complex biochemical interactions of molecules inside and outside of living cells. MCell is a collaborative effort between the Terry Sejnowski lab at the Salk Institute, and the Miriam Salpeter lab at Cornell University. NetSolve is very well suited to MCell's need and this project aims at writing a NetSolve-based framework to support large MCell runs. One of the central pieces of that framework is a scheduler that takes advantage of MCell input data requirements to minimize turn-around time. This scheduler is part of the larger AppLeS at the University of California, San Diego. The use of NetSolve isolates the scheduler from the resource-management details and allows researchers to focus only on the design of the scheduler.

A.5.2 IPARS

IPARS is a framework for developing parallel models of subsurface flow and transport through porous media. It currently can simulate single phase (water only), two phase (water and oil) or three phase (water, oil and gas) flow through a multi- block 3D porous medium. IPARS can be applied to model water table decline due to overproduction near urban areas, or enhanced oil and gas recovery in industrial applications. IPARS is being made into a fully functional NetSolve server. The goal of this project is to allow this server to be accessible via a web browser using the Common Gateway Interface on top of NetSolve's C interface. The server will also render animated graphics via a destination web-page. This total web-accessibility will allow those wanting to see IPARS simulations to do so with nothing but simple input parameters defining the simulation.

A.5.3 SCIRun

SCIRun is a scientific programming environment that allows the interactive construction, debugging and steering of large-scale scientific computations. SCIRun can be used for interactively: i)Changing 2D and 3D geometry models (meshes), ii) Controlling and changing numerical simulation methods and parameters and iii)Performing scalar and vector field visualization. Currently, NetSolve is being integrated into SCIRun as the broker for computational resources. This integration will allow for increased parallelism and performance in the SCIRun paradigm.

A.5.4 LUCAS

LUCAS is a system that uses computer modeling to integrate biological and socioeconomic data of land areas to help natural resource specialist evaluate the consequences of alternative land management scenarios. It uses the Geographic Information System, GRASS, to represent and manipulate spatial data on workstations. There is an on-going effort to integrate NetSolve to harness the computational cycles for LUCAS. This will prove especially useful when LUCAS is used to spawn several "replicates" which normally would compute in serial on the local machine. Using NetSolve, the computations would be done in parallel, possibly on machines specialized for High Performance Computing.

A.5.5 DIPS

DIPS is a software tool, developed at the Computer Graphics and Vision unit of the Graz University of Technology in Austria, which allows remote computing for image processing. DIPS extends the Image/J Java image processing application to provide remote access to the high-performance ImageVision library by Silicon Graphics. At its core, DIPS uses NetSolve as its metacomputing resource to provide unprecedented computing power by aggregating distributed resources on the Internet to a single system.

A.6 Current Developments and Future Research

A.6.1 Dynamic Server-software Enhancements

In the current NetSolve design and implementation, there is a tight coupling between the server's hardware and software components. The server is statically configured (at compile time) to solve a particular problem set. Although we have provided the tools to allow this problem set to be easily expanded, this can only be done initial configuration, so to increase a running servers capability entails a shutdown, reconfigure and restart loop. This will not be the case in the next major release of NetSolve. We are providing the capability of storing NetSolve specific software binaries in a software repository whose location is known to the NetSolve agent. At request time, should a particular server not possess the appropriate binaries, it will be directed to the repository for a download. This paradigm will not replace, but will enhance, the current protocol where the server is statically binded with software.

A.6.2 Fault Tolerance

As explained before, the fault tolerance possessed by the NetSolve system incorporates only a retry and restart mechanism. We are presently developing servers enhanced with checkpointing capabilities. As they run, the servers will take frequent checkpoints (via a core dump mechanism); should one of these servers fail, they will be restarted not from the beginning, but from the state represented by the core image of the most recent checkpoint. Homogeneous migration will also be possible, meaning that it will be possible to restart the process on a different machine of similar architecture and operating system. As this feature becomes more advanced, we will investigate heterogeneous migration and possibly checkpointing parallel programs.

A.6.3 Request Sequencing

We recently finished research that would allow us to minimize the network traffic between client and servers in a single client program that made numerous requests to NetSolve. We noticed that, in many cases, there exist data dependencies between these requests. We have implemented a feature that allows the client user to bracket together multiple requests to NetSolve. The NetSolve system then analyzes data dependencies and only sends to the servers the minimal data necessary. Inputs to later requests that were outputs of a previous request(s) need not be obtained from the client again. The server makes this data persistent and uses it across all requests as necessary. In our current model, all requests must execute on a single server. Future research will yield a model that will investigate using systems like the Internet Backplane Protocol (IBP) and other distributed storage facilities to stage data as requests are serviced on multiple servers.

A.6.4 Win32 Servers

The Distributed Component Object Model (DCOM) is a protocol that enables software components to communicate directly over a network in a reliable, secure, and efficient manner. DCOM is based on the Open Software Foundation's DCE-RPC and is a standard similar to that of the Common Object Request Broker Architecture (CORBA). We will be developing a version of the NetSolve server that acts as a gateway to problem solving libraries and systems optimized for the NT platform. The server will be built using the DCOM protocol to manage its networking interactions.

B WebFlow Object Web Computing

B.1 Overview of WebFlow System

In Figure 8, we show how one or more middle tier servers acts as a broker between any client and a collection of interesting services. Note that we view the services as being provided by a collection of (distributed) objects. We adopt what we call the pragmatic Object Web philosophy where realistic systems are likely to involve aspects of the four leading distributed object technologies: CORBA, COM, Java and XML. Appropriate middleware allows these different approaches to interoperate. In particular, WebFlow now uses XML to specify all object interfaces and these are termed the WebFlow and Grid Interfaces for the user and system view respectively. This two-interface model was adopted at meetings of the DATORR group in 1998-1999. As an example, the WebFlow Interface defines an abstract task such as "run a chemistry problem using an HPF simulation code with given data" and the middle tier server matches this with the backend objects. The latter are defined by the Grid Interface, which can use the Globus resource language RSL. This matching then instantiates a real job to solve the chemistry problem on one or more of the backend resources. WebFlow originally used Java Servers but now uses CORBA object servers. One simply takes the XML object specifications and uses this to generate the appropriate RMI or CORBA Interfaces necessary for the chosen middle tier. This use of XML object specification linked to different object runtimes is very

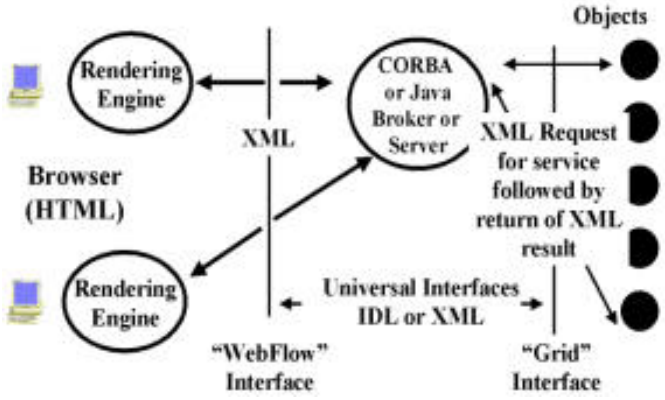

Figure 8: More Detailed 3 Tier Architecture.

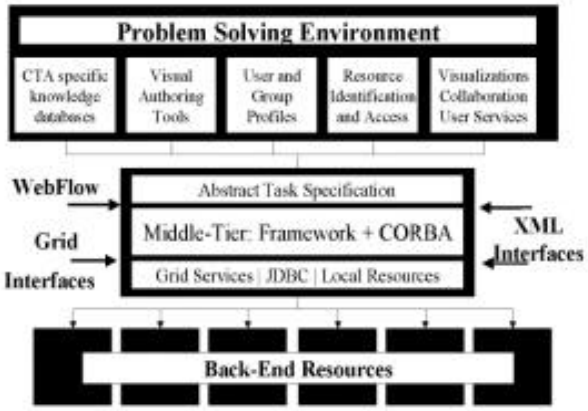

Figure 9: WebFlow system architecture.

common in modern commercial systems. Figure 9 takes the general architecture of the previous diagram and highlights the capabilities of WebFlow in each of the three tiers.

In the three tier diagram, WebFlow contributes to the client and middle tier as these are the PSE layers where one integrates the components composed of the basic HPCC tools, algorithms and applications. The WebFlow client tier can and has been constructed in several ways but one distinctive capability (which gives the system its name) is the WebFlow composition tool. Here a WebFlow front-end editor applet offers an intuitive click-and-drag metaphor for instantiating middleware or back-end modules, representing them as visual icons in the active editor area, and interconnecting them visually in the form of computational graphs, familiar to AVS and Khoros users. WebFlow middleware was originally given by a mesh of Java web servers, custom extended with servlet-based support for the WebFlow session implementing module and connection managers. These then implemented the middleware logic to support both this general distributed dataflow computing model as well as a more general linked object model. Both models are represented as abstract tasks in XML allowing scripted as well as visual invocation of programs. This computational paradigm is very popular in some fields (such as signal processing with Khoros) and is seen in research systems like Arcade from ICASE designed to support multidisciplinary applications such as you get with structures and fluid flow programs controlled by an optimization module. The WebFlow toolkit also includes the general

capability to link to backend resources as illustrated by its support of Globus.

Note that WebFlow only uses CORBA (or more generally commodity distributed object technology) to manipulate proxies for backend entities and thus is not impacted by performance limitations of commodity technology. WebFlow's front end supports visual proxies to specify the problem while the middle tier functional proxies support needed control logic. WebFlow relies on classic HPCC back-end capabilities for high performance computing and communication.

This WebFlow toolkit has been applied to build several problem solving environments. In the previous section, we describe two focused examples. One, LMS, did not use the composition tool but rather a custom Java applet front end to control particular linked applications for environmental modeling. A second application of the WebFlow is Quantum Monte Carlo Simulations developed in collaboration with the NCSA Condensed Matter Physics Laboratory. Here simulations are linked together and the results stored on many different computers. The output file of one application in the chain is the input of the next one, after a suitable format conversion. This was a natural place to use the WebFlow composition tool.

Recently we have used WebFlow technology in the Gateway project for the DoD High Performance Computing program. Gateway is designed to build a seamless access to the suite of different machines in a computer center. In this case, we needed to address security and fault tolerance more carefully and so re-implemented the WebFlow middle-tier using the industry standards distributed-object technologies, JavaBeans and CORBA and industry standard secure communication protocols based on SSL.

B.2 WebFlow Architecture

The WebFlow system is implemented as an Object Web three-tier system, as shown in Figuer 8. Tier 1 is a high-level *front* end for visual programming, steering, run-time data analysis, and visualization, that is built on top of the web and OO commodity standards. A distributed object-based, scalable, and reusable Web server and Object broker Middleware forms tier 2. *Back end* services comprise tier 3. In particular, high-performance services are implemented using the metacomputing toolkit of Globus.

B.2.1 Front End

Different classes of applications require different functionality of the front end. We have therefore designed the WebFlow system to support many different front-ends: from very flexible authoring tools and problem solving environments (PSE) that allows for dynamical creation of meta-applications from pre-existing modules, to highly specialized and customized front-ends to meet the needs of specific applications. Also, we support many different computational paradigms, from general object-oriented to data-flow to a simple "command line" approach. This flexibility is achieved by allowing as a WebFlow front end any program implementing the WebFlow API described below.

B.2.2 WebFlow and Grid Interfaces (API's)

The WebFlow API allows the user's task to be specified in the form of an Abstract Task Descriptor (ATD), following the current DATORR recommendations. The ATD is constructed recursively and may comprise an arbitrary number of subtasks. The lowest level, or atomic, task corresponds to the atomic operation in the middle tier, such as instantiation of an object, or establishing interactions between two objects through event binding. In many cases such details should be hidden from the end-user and even the front-end developer, thus the WebFlow API provides interfaces to higher-level functionality, such as submitting a single job or making a file transfer.

When specifying a task, the user does not have to specify the resources to be used to complete the task, but instead may specify requirements that the target resource must satisfy in order to be capable of executing the job. The identification and allocation of the resources is left to the discretion of the system. Typically,

the middle tier delegates it to the metacomputing services (such as Globus) or and external scheduler (such as PBS). Once the resources are identified, the abstract task descriptor becomes a job specification.

B.2.3 Middle Tier

A mesh of CORBA-based WebFlow servers (WS) currently gives the WebFlow middle tier. One of these: a dedicated gatekeeper server facilitates a secure access to the system. A general WebFlow server maintains the sessions within which the users create and control their applications. The middle-tier services provide the means to control the life cycles of modules and to establish communication channels between them. The modules can be created locally or on remote hosts. In the latter case the task of module instantiation and initialization is transparently delegated to a peer WebFlow server on the selected host, and the communication channels are adjusted accordingly. The services provided by the middle tier include methods for submitting and controlling jobs, methods for file manipulating, methods for providing access to databases and mass storage, as well as methods to query the status of the system, status of the users' applications, and their components.

Gatekeeper Server

The gatekeeper comprises three logical components: a (secure) Web Server, the AKENTI server, and a CORBA-based WebFlow server. The user accesses the WebFlow system through a portal web page from the gatekeeper web server. The portal implements the first component of WebFlow security: user authentication and generation of the user credentials that eventually will be used to grant access to resources. The AKENTI server controls the authorization process. For each authorized user, the web server creates a session (that is, it instantiates the user context in the WebFlow server, as described below) and gives permission to download the front-end applet. The applet is used to create or restore, run, and control user applications. The applet, using IIOP protocol, communicates directly with the CORBA-based WebFlow server.
To implement the WebFlow server we use the ORBacus (formerly known as OmniBroker) secure ORB, for which we have obtained a free research license. The security services are implemented on top of the IAIK SSL library, which is already used by the Jigsaw Web server.

B.2.4 WebFlow Server

The WebFlow server initializes the ORB and several generic CORBA and specific WebFlow services. The main functionality of the WebFlow server is managing WebFlow sessions. A session is established automatically after the authorized user is connected to the gatekeeper by creating a user context. The user context is a container object that stores the user applications. The application is another container object that stores components of the user application. The application component is either a single WebFlow module or another, finer-grain application context. This way, the WebFlow server can simultaneously manage many sessions, and within each session, the user can define many applications hierarchically composed of many modules.

B.2.5 WebFlow Modules

The WebFlow modules are CORBA objects conforming to the JavaBeans model whose implementation is described in detail in the Syracuse Ph.D. thesis of E. Akarsu. The functionality of a module is implemented either directly in the body of the module or the module serves as a proxy of specific back-end services, such as database or HPCC services. We expect to support the standards for HPPC back-end services under development by the Grid Forum. For databases we support the industry standard JDBC (Java Database Connectivity).

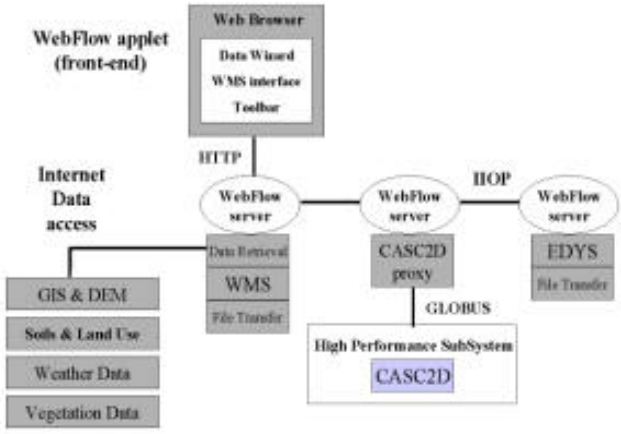

Figure 10: The LMS PSE organization.

B.2.6 Interactions Between WebFlow Modules

The WebFlow modules follow the JavaBeans model, and they interact with each other by using JavaBeans methods through event binding, property binding, and vetoable property binding. With JavaBeans, events are used to communicate information about the changing state of a bean. Events form a core component of the JavaBeans architecture in that they are largely responsible for enabling beans to be plugged together as building blocks in an application builder. Event notification in Java works using method invocation. The object that is a source of an event calls a method on the destination object for one event when the event is triggered. The destination of the message must implement the method (or methods) to be notified when the event occurs. The event object encapsulates all the information about an event.

Event targets are connected to event sources through a registration mechanism. WebFlow applications are created dynamically from independently developed WebFlow modules. Therefore, we provide support for a dynamical event binding based on the standard CORBA dynamic interface invocation (DII) and dynamic stub invocation (DSI) mechanisms. This is implemented by introducing an event adapter associated with the application context. The adapter maintains a binding table to associate the event sources with the actual event destinations. Note that we choose not to use the important commodity Enterprise JavaBean middle tier containers as currently they appear difficult to implement consistently with our security requirements.

B.3 WebFlow Applications

B.3.1 WebFlow Application: Land Management System (LMS)

The LMS project was sponsored by the U.S. Army Corps of Engineers Waterways Experiment Station (ERDC) Major Shared Resource Center (MSRC) at Vicksburg, MS, under the DoD HPC Modernization Program, Programming Environment and Training (PET).

The application can be idealized as follows. A decision maker (the end user of the system) wants to evaluate changes in vegetation in some geographical region over a long time period caused by some short term disturbances such as a fire or human's activities. One of the critical parameters of the vegetation model is soil condition at the time of the disturbance. This in turn is dominated by rainfall that possibly occurs at that time. Consequently as shown in Figure 10, the implementation of this project requires:

- Data retrieval from remote sources including DEM (data elevation models) data, land use maps, soil textures, dominating flora species, and their growing characteristics, to name a few. The data are

available from many different sources, for example from public services such as USGS web servers, or from proprietary databases. The data come in different formats, and with different spatial resolutions. Without WebFlow, the data must be manually prefetched.

- Data preprocessing to prune and convert the raw data to a format expected by the simulation software. This preprocessing is performed interactively using WMS (Watershed Modeling System) package.
- Execution of two simulation programs: EDYS for vegetation simulation including the disturbances and CASC2D for watershed simulations during rainfalls. The latter results in generating maps of the soil condition after the rainfall. The initial conditions for CASC2D are set by EDYS just before the rainfall event, and the output of CASC2D after the event is used to update parameters of EDYS and the data transfer between the two codes had to be performed several times during one simulation. EDYS is not CPU demanding, and it is implemented only for Windows95/98/NT systems. On the other hand, CASC2D is very computationally intensive and typically is run on powerful backend supercomputer systems.
- Visualization of the results of the simulation. Again, WMS is used for this purpose.

One requirement of this project was to demonstrate the feasibility of implementing a system that would allow launching and controlling the complete simulation from a networked laptop. We successfully implemented it using WebFlow with WMS and EDYS encapsulated as WebFlow modules running locally on the laptop and CASC2D executed by WebFlow on remote hosts. Note that the existing codes were not modified but rather the WebFlow PSE used object wrappers to construct a powerful integrated application specific environment. Further the applications involved showed a typical mix of HPCC and computationally less demanding PC codes.

For this project we developed a custom front-end that allows the user to interactively select the region of interest by drawing a rectangle on a map. Then one could select the data type to be retrieved, launch WMS to preprocess the data and make visualizations, and finally launch the simulation with CASC2D running on a host of choice.

B.3.2 WebFlow Application: Quantum Simulations (QS)

A major goal of the QS activity was to demonstrate the feasibility of layering WebFlow on top the Globus metacomputing toolkit. This way WebFlow serves as a job broker for Globus, while Globus (or more precisely, GRAM-keeper) takes responsibility of actual resource allocation, which includes authentication and authorization of the WebFlow user to use computational resources under Globus control.

This application can be characterized as follows. A chain of high performance applications (both commercial packages such as GAUSSIAN or GAMESS or custom developed) is run repeatedly for different data sets. Each application can be run on several different (multiprocessor) platforms, and consequently, input and output files must be moved between machines. Output files are visually inspected by the researcher; if necessary applications are rerun with modified input parameters. The output file of one application in the chain is the input of the next one, after a suitable format conversion. The logical structure of the application is shown in Figure 11. GAUSSIAN and GAMESS are run as Globus jobs on Origin2000 or Convex Exemplar at NCSA, while all file editing and format conversion a performed on the user's desktop.

Unlike LMS, for QS we are using the WebFlow program composition editor as the front-end. This WebFlow editor provides an intuitive environment to visually compose (click-drag-and- drop) a chain of data-flow computations from preexisting modules (as shown in Figure 12). In the edit mode, modules can be added to or removed from the existing network, as well as connections between the modules can be updated. Once created the network can be saved (on the server side) to be restored at a later time. The workload can be distributed among several WebFlow nodes (WebFlow servers) with the interprocessor communications taken care of by the middle-tier services. Moreover, thanks to the interface to the Globus system in the

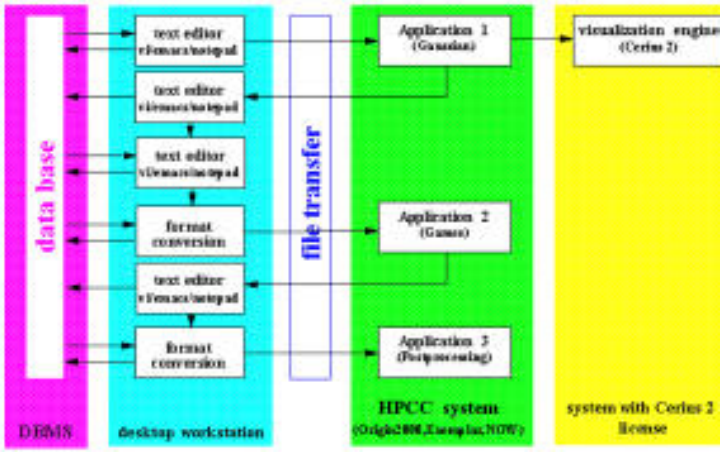

Figure 11: Functional architecture of the quantum simulation application.

backend, execution of particular modules can be delegated to powerful HPCC systems. In the run mode, the meta-application represented by the visually constructed graph is passed to the middle-tier by sending a series of requests (module instantiation, intermodule communications) to the middle tier services.

The control of the module execution is exercised not only by sending relevant data through the input ports of the module. Rather the majority of modules we developed so far requires some additional parameters that can be entered via *module controls*, which are Java applets displayed in a card panel of the main WebFlow applet. The communication channels between the backend implementation of a module and its front-end module controls are generated automatically during the instantiation of the module.

B.3.3 WebFlow Application: Gateway Seamless Access

Exploiting our experience developing the WebFlow PSEs described above, we designed a new system, Gateway, to provide seamless and secure access to computational resources at DoD modernization sites in particular first at the ASC Major Shared resource Center at Wright Patterson airforce base in Dayton. While preserving the original three-tier architecture, we re- engineered the implementation of each tier in order to conform to XML based standards indicated in Figures 3. In particular, we developed for this application the CORBA and the JavaBeans model to build a new middle tier, which facilitates seamless integration of commodity software components. The security system supports the Kerberos and SecurID system adopted by DoD for their modernization program. This new technology is being retrofitted to the initial applications described above and used in other applications being developed now.

Gateway's architecture includes provision for visualization where we are working with NCSA (VisBench) and ARL (DICE) to design visualization subsystems supporting the WebFlow distributed object model. In the first two PSE's discussed above we have integrated existing visualization systems such as WMS (for LMS) and Cerius (for QS case) with WebFlow. We have also prototyped XML specifications of collaboration which when combined with the WebFlow API can generate collaborative portals to computing. Initially Gateway is designed with a custom chemistry front-end developed by OSC. This uses job submission (to the scheduler PBS via Globus), choice of multiple applications and basic WebFlow file services. The front end is arranged in layers: Entry, Problem Description, Code and Results with well defined (XML) interfaces. This approach appears to generalize to other applications.

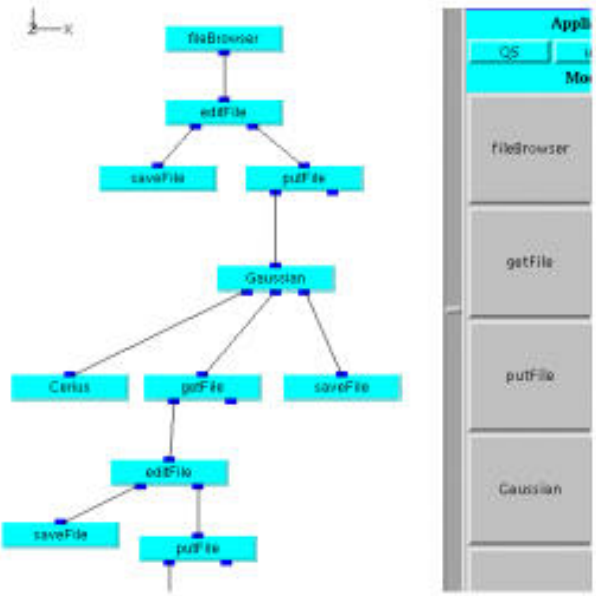

Figure 12: Fragment of quantum simulation WebFlow composition tool.

C Other Metacomputing Resources

C.1 Globus

Globus is a software system that provides infrastructure for computations that leverage distributed computational and informational resources. It is being developed at the Argonne National Laboratory and the University of Southern California's Information Sciences Institute. Currently, the NetSolve system uses a component of Globus referred to as the Heart Beat Monitor (HBM.) The HBM allows NetSolve to easily detect failed server hosts and update the agent's database. We are also testing a new NetSolve proxy-client that allows Globus-enabled NetSolve client users to access and use Globus computational resources through the NetSolve interface. We have discussed extensively the integration of WebFlow with Globus.

C.2 CONDOR

The CONDOR system, of the University of Wisconsin, takes advantage of the fact that many CPU cycles go wasted on idle workstations at times when the primary user is not using his or her machine. The system assigns tasks submitted to the CONDOR system to "registered" host machines as long as these machines are idle. Should an owner return to his machine, the task is immediately halted and assigned to another host. CONDOR pools can be used as NetSolve servers. In essence, the request for service is forwarded to the CONDOR system, which then assigns the task to an idle workstation for completion. CONDOR has not yet been used with WebFlow but it should be straightforward to use CONDOR at the back-end tier in the manner Globus is used in previous examples.

C.3 Ninf

Ninf is a system very similar to NetSolve. Developed at the Electrotechnical Laboratory in Tsukuba, Japan, it too provides an interface that allows for remote execution of functional components. In a collaborative effort, a NetSolve-Ninf bridge has been built that allow both systems to utilize servers provided to the other.

Administrators of NetSolve and Ninf systems can then join forces to create an even bigger computational grid.

C.4 Legion

Legion is an object-based metasystems software project at the University of Virginia. Its goal is to tie together host systems with high-speed links and present the illusion of a single computer with access to varied physical resources. The NetSolve client-user can use the NetSolve interface while leveraging metacomputing resources of Legion. The NetSolve client side uses Legion data-flow graphs to keep track of data dependencies. We hope to study the linkage of Legion and WebFlow but as both have object models, the integration is not as straightforward as for Globus and WebFlow.

C.5 Gateway and Related Approaches to Seamless Access and Application Integration

There are several other projects addressed to solving the problem of seamless access to remote resources. A comprehensive list of these is available from the JavaGrande and DATORR web sites. Here we mention the three that are most closely related to the Gateway project.

The UNICORE project introduces an excellent model for the Abstract Task Descriptor that most likely will strongly influence the DATORR standard and, consequently, we are taking a similar approach. The UNICORE middle tier is given by a network of Java web servers (Jigsaw). The WebSubmit project implements web access to remote high-performance resources through CGI scripts. Both projects use https protocol for user authentication (as we do), and implement custom solutions for access control. The ARCADE project is aimed at multidisciplinary applications, and its designers intend to use CORBA to implement the middleware. As of now, there is no available description of the ARCADE security model.

D The WebPDELab Server: A PSE for Partial Differential Equations Applications

WebPDELab is a World Wide Web server that allows users to define, solve and analyze partial differential equation (PDE) problems using a comprehensive graphical user interface from any Java-enabled browser on a wide variety of platforms. The WebPDELab server is currently supported by a 16 CPU Intel cluster which allows users to solve PDE problems sequentially or in parallel on the supporting host cluster. WebPDELab is the PELLPACK problem solving environment implemented as an Internet-based client-server application. It provides access to a library of PDE solvers and an interactive graphical interface that support the pre-processing and post-processing phases of sequential and parallel PDE computing. The PELLPACK software is implemented as a system of X windows programs and libraries, compiled on an i86pc SunOS 5.6 machine. WebPDELab displays the interface of the PELLPACK software within a Java-capable browser using the Virtual Network Computing remote display system.

D.1 The WebPDELab Server

WebPDELab is a World Wide Web server that provides access to PELLPACK, a sophisticated problem solving environment for Partial Differential Equation (PDE) problems. Users can connect to the WebPDELab site at

`http://webpellpack.cs.purdue.edu`

with any Java-enabled brower for information, demonstrations, cases studies and PDE problem solving service. A new PELLPACK session is initiated for each user that connects to the WebPDELab server, where

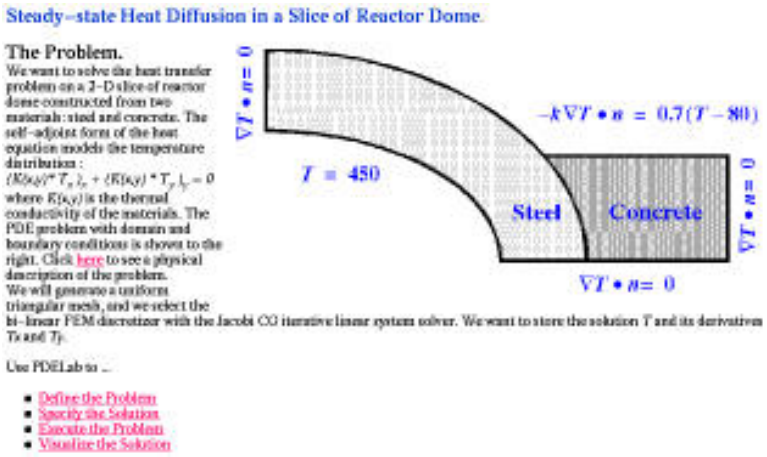

Figure 13: Sample case study from Getting Started at the web site.

a unique identification and private file space for the session are created. The file space is available until the user disconnects from the service, at which time the session is terminated and the user's files are deleted. Users may download files generated by PELLPACK to their own machines before terminating the session, and they may upload files to WebPDELab at the start of subsequent server sessions. When the server invokes the PELLPACK system software, the entire PDE problem solving environment described in is presented to the user. A detailed decription of the functionality and operation of the PELLPACK software is given in the User Guide. PELLPACK is a comprehensive system for modeling physical objects based on PDEs, and has been used by hundreds of students and faculty both inside and outside of Purdue University for solving problems in physics (liquid crystal droplets, proton flux propogation), thermal field analysis, fluid dynamics, semiconductors, geophysical research, electomagnetic field analysis, thermo-elasticity, structural analysis, and other scientific and engineering applications. PELLPACK has a user friendly interface, and even first time users can solve interesting problems by following the fully documented, step-by-step descriptions of the problem-solving process presented in `Getting Started` at the WebPDELab site.

D.2 The PELLPACK Problem Solving Environment

WebPDELab is a Internet-based client-server implementation of the PELLPACK software. PELLPACK is a system for specifying and solving PDE problems and visualizing their solutions. It provides a graphical user interface for defining the PDE model and selecting solution methods, and is supported by the Maxima symbolic system and well-known numerical libraries. The graphical interface is implemented on top of a very high level PDE language. Users can specify their PDE problem and its solution visually using the graphical interface or textually using the "natural" language. PELLPACK has incorporated over 100 solvers of various types which cover all the common PDE applications in 2 and 3 dimensions.

In the PELLPACK system, a problem is represented by the PDE objects involved: PDE model, domain, conditions on the domain boundary, solution methods, and output requirements. The PELLPACK interface consists of many graphical tools and supporting software to assist users in building a problem definition. A textual specification of these objects comprise PELLPACK's natural PDE language, and the language representation of each object is generated by the object editors/tools. The language definition of a user's problem (the .e file) is automatically passed to PELLPACK's language processor, which translates the problem into a Fortran driver program, and then compiles and links it with numerical libraries containing the user-specified solver methods. Sequential or parallel program execution is a one–step process; the program is executed on one or more machines in the supporting i86pc host cluster. Problem solutions are passed to

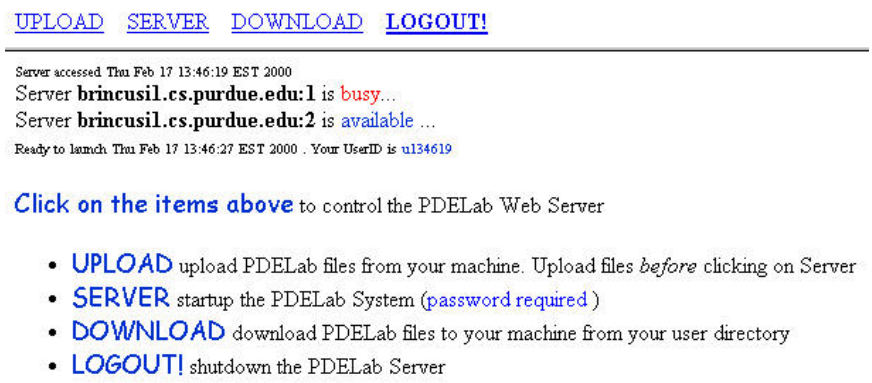

Figure 14: WebPDELab server with control panel in the top frame and panel instructions and connection information in the bottom frame.

the PELLPACK visualization system for solution display and analysis.

D.3 The WebPDELab Interface

The WebPDELab server is accessed from the WebPDELab web site. This web site is an instructional source for anyone interested in solving PDE applications. It provides information about PDE problem solving in general, and about the process of solving PDE problems with PELLPACK in particular. A collection of fully documented case studies is available at the site (Figure 13), presenting step-by-step solutions of common PDE applications (flow, heat transfer, electo-magnetism, conduction).

Potential WebPDELab users are required to register. Following a validated registration, users are connected to a host machine and presented with a control panel: `Upload`, `Download`, `Server` and `Logout` (Figure 14). At this point, the user's directory space has already been created, so `Upload` can be used to load PELLPACK files saved from previous sessions (.e files, mesh files, etc) to the directory. `Download` produces a current listing of the user's directory where files can be viewed or dowloaded. Users should look here frequently during the session to check on PELLPACK generated problem, solution and trace files. `Server` invokes the password protected PELLPACK software. After the password is entered and verified, the top level window of the PELLPACK system appears in the bottom frame of the browser window (see Figure 15.) A collection of sample problems has been placed in the user's directory, so users can load an example into the PELLACK session or begin their own problem definition. The PELLPACK session in Figure 16 is in the bottom frame of the WebPDELab server. The items of the control panel are still available in the top frame, but only `Download` and `Logout` are enabled. `Upload` and `Server` remain disabled while the PELLPACK software is running in the bottom frame.

During the PELLPACK session, WebPDELab passes the display of the remotely executing PELLPACK environment to the users browser window. The graphical interface displayed on the user's screen belongs to PELLPACK and is not described in this paper. When users click on `Logout`, the PELLPACK session is terminated and the user's directory is removed. WebPDELab traces all user activities from the start of the server session until its termination. Users files are secure from other users, but WebPDELab 'looks at' the contents of every file uploaded to WebPDELab or created by the user from within the PELLPACK system. WebPDELab protective mechanisms implemented for the security of the WebPDELab server and host cluster are discussed in Section D.5

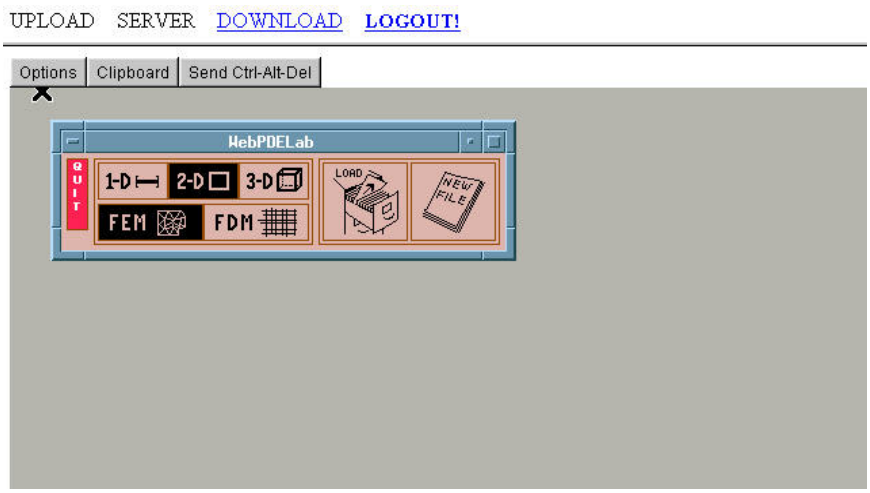

Figure 15: The PELLPACK top level window appears in the bottom frame of the WebPDELab browser window. It is ready for user interaction.

D.4 WebPDELab Implementation

WebPDELab is the PELLPACK problem solving environment implemented as a web server using Virtual Network Computing (VNC). VNC is a remote display system which allows users to view a computing "desktop" environment from anywhere on the Internet using a wide variety of machine architectures. VNC consists of a server which runs the applications and generates the display, a viewer which draws the display on the client screen, and a TCP/IP connection between them. The server is started on the machine where the desktop resides, after which any number of viewers can then be started and connected to the server. This allows the client user to access the applications, data, and entire desktop environment provided by the server. The viewer is a small, sharable, platform-independent, stateless system which runs on the client machine.

In the WebPDELab implementation, a new VNC Unix server is started for each user who accesses the WebPELab web server from a Java-enabled browser (see Figure 17). The VNC Java viewer is started from the user's browser, allowing the user to display and interact with the PELLPACK environment, which consists of X windows programs and libraries compiled and running on the i86pc SunOS 5.6 host machines. Within this framework, any user world-wide who is connected to the Internet and has access to a Java-capable browser can run WebPDELab. The WebPDELab *manager* is the collection of CGI scripts (Common Gateway Interface protocol for browser to server communication) which control all user activity once the PDELab Server button at the WebPDELab web site is pressed. When a user accesses the server, the manager collects information on all currently running VNC servers from the host machines. The manager then asks the potential user to enter registration information, including a valid e-mail address. After validation, a unique user id is generated for the new user, and a log file is set up to track registration information, user access/exit times, and PELLPACK-related activities. The host machine is selected by the manager for running the VNC server and subsequently the PELLPACK software. A protective client-server application is used to launch the VNC server, so that users are never logged in to any machine in the host cluster. The VNC server startup invokes the PELLPACK system, and the manager creates the user directory, sends the control panel to the user, and monitors the user's interaction with the control panel items (Upload, Download, Server and Logout).

`Upload` is implemented using copyrighted public domain code at `http://stein.cshl.org/WWW/software/CGI`. The code has been modified to operate with the WebPDE-

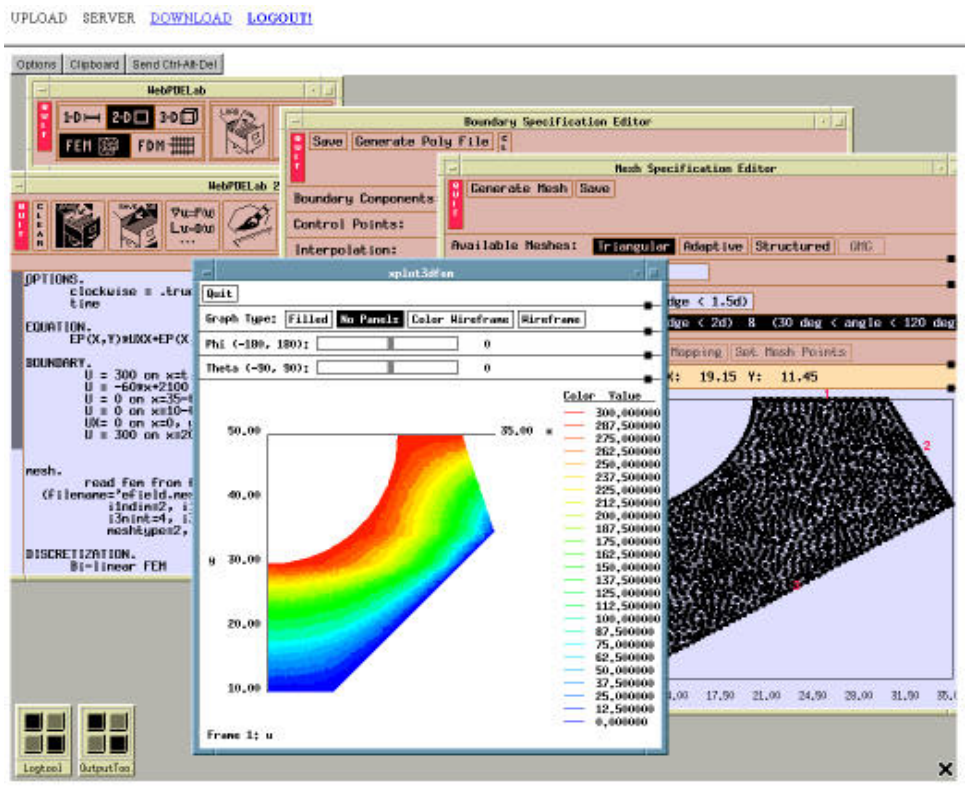

Figure 16: PELLPACK session running inside the WebPDELab browser window.

Lab/VNC user directory privacy restrictions. `Download` is implemented as a standard link to the user's file space, but additional password security protects a user's assigned directory from all other users on the Internet. `Server` connects the VNC client user to the VNC server which has been instantiated for the caller on the selected host for a specific VNC server. After control has passed to the VNC client, the manager waits for a VNC disconnect or a `Logout` button click. When signalled to start exit processing, the manager saves the trace of user activities to the log data base, kills the VNC server, and removes the user's directory. The manager also checks all executing VNC servers periodically for sessions running longer than 10 hours, and these sessions are terminated. When the manager has finished exit processing, control is returned to the WebPDELab home page.

D.5 WebPDELab Security Issues

All Internet based services must be concerned with security issues to protect their network and host environment from unauthorized access. WebPDELab implements measures to provide such a secure environment by enforcing common rules of best practices which are used to secure Unix machines, taking advantage of the strength and flexibility of the Unix operating system. WebPDELab maintains several levels of security provided by the operating system, the WebPDELab and VNC servers, and protective language processing software built on top of the PELLPACK system. These security measures are described in this section.

When a user logs into the WebPDELab server, a CGI script is executed which generates a unique UID (user identification) for that user and requests one of the cluster host machines to invoke a VNC X-server. The WebPDELab CGI scripts reside on an isolated machine dedicated to serving CGI requests. This machine

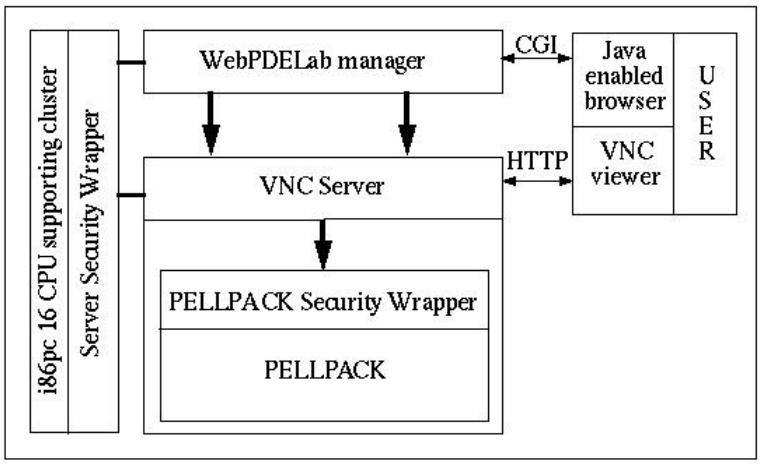

Figure 17: Implementation of the WebPDELab server.

has no NFS-mounted disks, therefore an attacker attempting to take advantage of vulnerable CGI scripts is locked into the cgi-bin directory and cannot gain access to any other machines or disks. All parameters passed to WebPDELab CGI scripts are scanned to ensure they contain precisely the expected values (argument number, length and contents), else the request is terminated, The cluster machines listen on a fixed port for startup requests from the CGI machine. If an attempt is made to connect to this port which does not originate from the CGI host, the connection is immediately terminated. All cluster machines run a daemon which listens for socket connections on a specified port and spawn a child process to serve the request, while the parent continues to listen for other connections so that requests can be served simultaneously. A client program is invoked by the CGI script to contact the cluster machine and request that a new VNC X-server be launched. The client may only specify the VNC X-server startup parameters, since the launching of the VNC X-server binary is hard-coded in the configuration file of the daemon serving requests originating from the CGI host. The VNC server itself is protected by a challenge-response password scheme.

In order to protect the machine from unauthorized Fortran code inserted by a user into the .e file, specialized filters have been built into the PELLPACK system. The original language processor already restricted the location of Fortran code to specialized segments within the PELLPACK problem definition file; these segments are now re-parsed by filters that identify inserted Fortran statements for unauthorized code.

Every user is provided with a unique directory for uploading and downloading files, thus facilitating the option of saving and retrieving material. Users' directories are password protected, securing each user from all other users. Every user file, however, is opened and checked by WebPDELab for legal content as it is uploaded or saved by the user from inside PELLPACK.

D.6 WebPDELab Features and Issues

There are significant benefits resulting from the implementation of the WebPDELab;

- *Generality.* Any machine connected to the Internet can use the PELLPACK environment without concerns about language or machine compatibility.
- *Interaction.* Users can specify the PDE with normal interaction speeds for the client machine, since data entry is done locally. The amount of code exported to support the user interface is substantial (several megabytes), but it is only a fraction of the PELLPACK system. If the user has no graphics

capability, then the text based interface tools must be used; these are less convenient but still practical to use. As the PDE problem is being specified, information is sent to the server. The server might request additional information but once the problem is completely specified, it is solved on the server's host machines. After the PDE is solved, the user can either view output generated by the server or request that the solution (normally a large data set) be sent for local use.

- *Access to High Performance Computers.* Any user can access machines with sufficient power to solve the PDE problem. Even if the solution is too large to be sent to the user (or if there are no local visualization tools), the solution can be explored over the Internet.
- *No Code Portability Problems.* User do not need to have the code in the local machine language, since the software infrastructure operates only on the server's host machines.

Three technical issues must be considered in the deployment of WebPDELab as a successful server. First, the user interface must be clearly separated from the rest of the system. Our system is very modular in nature and we have already essentially completed this task. Second, we must create an efficient, exportable user interface. We have already made a prototype exportable user interface which is neither efficient nor general. It assumes the user has an X-windows server and it requires excessive network communication. We have studied Java implementation, and believe we can use it to obtain both efficiency and generality on the network.

Third, it is the problem of dealing with the visualization of very large data sets over the network. Using WebPDELab, a person with a simple PC can generate a PDE solution consisting of millions of data points in 3-D. In our own group we have 155 Mbit/sec ATM networks and expensive graphics workstations to visualize such solutions. We see two ways to provide visualization service to the user neither of which is always satisfactory. (1) We have visualization tools to slice, rotate, color, etc., data for viewing. We can send these images back over the Net. But the user might have a slow network connection or a black and white display, and in this case the viewing process would be painfully slow. (2) We can send the data set to the user. A two million point solution is not rare and its data set would be at least 25–50 Mbytes. The transmission time could be prohibitive if the user has slow network connections. In addition, the user might not have space to store the solution, or might not have any visualization tools that can handle the data. We believe that visualization over the Net will be a serious problem for some users, and it is one we currently have no solution for. We believe that this is a common problem and that the Net infrastructure will provide solutions in a few years.

In summary, we have an operational prototype of WebPDELab and a plan providing a very useful and innovative network service using it. The implementation of the plan does not require new science or technology and it can be accomplished with reasonable cost and time.

E Appendix References

The following references may be used to learn more about the topics presented the Appendix. They are not used in the text of this appendix.

1. G. C. Fox, W. Furmanski, "High Performance Commodity Computing" in "The Grid. Blueprint for a New Computing Infrastructure", Eds, C. Kesselman and I. Foster, Morgan-Kaufmann Publishers, Inc., San Francisco, 1998.
2. Geoffrey C. Fox, Wojtek Furmanski, Hasan T. Ozdemir and Shrideep Pallickara, "High Performance Commodity Computing on the Pragmatic Object Web" , report published by RCI, `http://tapetus.npac.syr.edu/iwt98/pm/documents`.

3. D. Bhatia, V. Burzewski, M. Camuseva, G. C. Fox, W. Furmanski, G. Premchandran, "WebFlow: A Visual Programming Paradigm for Web/Java based coarse grain distributed computing", Concurrency: Practice and Experience, 9, 555-578 (1997), http://tapetus.npac.syr.edu/iwt98/pm/documents.

4. E. Akarsu, G. C. Fox, W. Furmanski, T. Haupt, "WebFlow - High-Level Programming Environment and Visual Authoring Toolkit for High Performance Distributed Computing", in proceedings of Supercomputing '98, http://www.npac.syr.edu/users/haupt/WebFlow/papers/SC98/index.html.

5. Erol Akarsu, Geoffrey Fox, Tomasz Haupt, Alexey Kalinichenko, Kang-Seok Kim, Praveen Sheethaalnath, and Choon-Han Youn, "Using Gateway System to Provide a Desktop Access to High Performance Computational Resources", HPDC8 Conference August 1999.

6. WebFlow project, http://www.npac.syr.edu/users/haupt/WebFlow/demo.html.

7. Tomasz Haupt, Erol Akarsu and Geoffrey Fox, Web Based Metacomputing, Special Issue on Meta-Computing for the FGCS International Journal on Future Generation Computing Systems.

8. T. Haupt, E. Akarsu, G. Fox, A. Kalinichenko, K-K .Kim, P. Sheethalnath, C-H. Youn, "The Gateway System: Uniform Web Based Access to Remote Resources", High Performance Computing and Networking' 99, Amsterdam, April 1999.

9. Gateway Project, ĥttp://www.osc.edu/ kenf/theGateway.

10. Globus Metacomputing Toolkit, http://www.globus.org.

11. Java Grande Forum, http://www.javagrande.org.

12. Grid Forum, http://www.gridforum.org/.

13. S. S. Mudumbai, W. Johnston, M. R. Thompson, A. Essiari, G. Hoo, K. Jackson, "Akenti: A Distributed Access Control System",
http://www-itg.lbl.gov/Akenti.

14. Object Oriented Concepts, Inc., ORBacus SSL, http://www.ooc.com/ssl/.

15. Computing Portals WebSite maintained by Gregor von Laszewski,
http://www.computingportals.org.

16. UNICORE: Uniform Access to Computing Resources, http://www.fz-juelich.de/unicore.

17. WebSubmit: A Web-based Interface to High-Performance Computing Resources,
http://www.itl.nist.gov/div895/sasg/websubmit/websubmit.html.

18. ARCADE, http://www.icase.edu:8080.

19. WMS, EDYS and CASC2D codes have been made available to us by ERDC Vicksburg. EDYS is written by Michael Childress and CASC2D is written by Fred Ogden,
http://www.wes.hpc.mil.

20. Quantum Simulations, http://www.ncsa.uiuc.edu/Apps/CMP/cmp-homepage.html.

21. Distributed Interactive Computing Environment DICE at ARL Aberdeen, Md.,
http://www.arl.hpc.mil/SciVis/dice/index.html.

22. VisBench and NCSA Visualization Activities,
http://www.ncsa.uiuc.edu/SCD/Vis/.

23. Advanced Visualization System, `http://www.avs.com/`.

24. Khoros, `http://www.khoral.com/khoros/`.

25. Catlin, A. C., Weerawarana, S., Houstis, E., and Gaitatzes, M., "The PELLPACK User Guide", Technical Report, Dept. of Computer Sciences, Purdue University, 2000.

26. Houstis, E., Rice, J., Weerawarana, S., Catlin, A., Gaitatzes, M., Wang, K., and Papachiou, P., "PELLPACK: A Problem Solving Environment for PDE-based Applications on Multicomputer Platforms", ACM Trans. on Math. Soft., 24, No. 1:30-73, 1998.

27. Richardson, T., Stafford-Fraser, Q., Wood, K. R., and Hopper, A., "Virtual Network Computing",IEEE Internet Computing, 2, No. 1:33-38, 1998.

28. DoE Common Component Architecture for Scientific Objects, `http://www.acl.lanl.gov/cca-forum/`.

29. Commodity Grid Kits based on Globus (CoG Kit), `http://www.globus.org/cog/`.

30. Indiana University Research Group in Common Component Architecture, `http://www.extreme.indiana.edu/ccat/index.html`.

F Extended Bibliography

As the reader would have discovered by now, the field of PSEs is sufficiently *wide* and *open* to make it impractical to compile and print a comprehensive bibliography that would be restricted to any reasonable page limits. Paraphrasing Jonathon Green[1], the nature of bibliography compilation, whether in form or content, brings with it decisions, and with decisions, however disinterested, comes choice. In this choice, it is hard to avoid bringing in our own biases but we hope that they had only minimal effect on this compilation.

Some background information on the way that this new bibliography was compiled might prove useful. Our starting point was the original bibliography that we compiled on the occasion of the original PSE Workshop *Future Research Directions in Problem Solving Environments for Computational Science. Report of a Worksop on Research Directions in Integrating Numerical Analysis, Symbolic Computing, Computational Geometry, and Artificial Intelligence for Computational Science*, organized by E. Gallopoulos, E.N. Houstis and J.R. Rice and hosted by K. Abdali at NSF. It contained 200 items and was included in the followup technical reports that were published simultaneously in 1992 by the Department of Computer Sciences at Purdue University and the Center of Supercomputing Research and Development at the University of Illinois. At the time of the workshop, the term "Problem Solving Environment" was not in common use, so in addition to attempting a comprehensive listing of papers on PSE research, the bibliography contained many items on topics that lie at the foundations of the area, namely Numerical Analysis, Symbolic Computing, Computational Geometry, and Artificial Intelligence. Our choices in the compilation of the new bibliography reflects the evolution of the field of PSEs to maturity. The building of this bibliography took into account the following developments:

- The natural growth of the field.
- Maturity and standardization in some of the foundation areas. As important highlights in these developments we cite the observed convergence in parallel architectures[2] the standardization of software

[1] Jonathon Green, *Chasing the Sun. Dictionary Makers and the Dictionaries they made* (New York: Henry Holt and Co.) 1996.

[2] See for example [Kuck, 1996] and D.E. Culler, J.P. Singh and A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach* (San Francisco: Morgan Kaufmann) 1998.

support systems for parallel processing (MPI), and the increasing acceptance of quality numerical libraries (LAPACK).

- The extraordinary expansion and impact of Internet technologies.
- The growing influence of object-oriented technologies in scientific computing.

These developments affected the bibliographical update as follows:

- We added citations to new efforts on PSEs. In some circumstances we opted to eliminate citations to older versions of the same work in favor of the most recent document.
- We eliminated multiple citations to some foundation topics that are now adequately standardized.
- We added several citations to relevant research on Internet technologies.

The figure on the following page depicts the timeline of the citations in this bibliography and shows the significant increase in PSE related publications after the 1991 workshop.

Not surprisingly, developments in the area affected not only the product but also the compilation process. The 1991 bibliography was compiled from the workshop organizers' personal records and after extensive "in-person" visits to the collections of the libraries of the University of Illinois at Urbana-Champaign and Purdue. This time, we not only visited additional library collections (in particular the Pennsylvania State University and Indiana University) but also used online resources from professional societies, bookstores and publishers that were unavailable only a few years ago. We are particularly grateful to those individuals that have created and are maintaining the following sites:

- URL `http://www.math.utah.edu/ beebe/` which hosts the collection of bibliographical information maintained by Nelson Beebe at the University of Utah.
- URL `http://liinwww.ira.uka.de/bibliography/index.html` for the "Collection of Computer Science Bibliographies" maintained by Alf-Christian Achilles, at the University of Karlsruhe.

References

[IMS, 1987] (1987). *Math/Library, Stat/Library,* and *SFUN/Library.* IMSL, Houston, TX.

[nag, 1988] (1988). *NAG Library Manual.* Numerical Algorithms Group, Oxford, England.

[der, 1989] (1989). *DERIVE User Manual.* Soft Waterhouse, Inc., Honolulu, 3rd edition.

[pro, 1989] (1989). *User's Manual: PDE/PROTRAN.* IMSL, Houston, TX.

[poo, 1994] (1994). *POOMA '94 Parallel Object-Oriented Methods and Applications*, Santa Fe, New Mexico. Los Alamos National Laboratory. Proceedings are located at http://www.acl.lanl.gov/Pooma94.

[poo, 1996] (1996). *POOMA '96 The Parallel Object-Oriented Methods and Applications Conference*, Santa Fe, New Mexico. Los Alamos National Laboratory. Proceedings are located at http://www.acl.lanl.gov/Pooma96.

[mat, 1999] (1999). MATCOM: A MATLAB to C++ compiler. Published by MathTools, Ltd. See URL `http://www.mathtools.com/matcom4.html`.

[mid, 1999] (1999). MIDEVA: A faster replacement for MATLAB. Published by MathTools, Ltd. See URL `http://www.mathtools.com/mideva.html`.

[Abelson et al., 1989] Abelson, H., Eisenberg, M., Halfant, M., Katzenelson, J., Sacks, E., Sussman, G. J., Wisdom, J., and Yip, K. (May 1989). Intelligence in scientific computing. *Comm. Assoc. Comput. Machin.*, 32(5):546–562.

[Abramowitz and Stegun, 1965] Abramowitz, M. and Stegun, I. A. (1965). *Handbook of Mathematical Functions*. Dover.

[Aird and Rice, 1983] Aird, T. J. and Rice, J. (1983). PROTRAN: Problem solving software. *Adv. Engin. Software*, 5:202–206.

[Akers et al., 1998] Akers, R., Baffes, P., Kant, E., Randall, C., Steinberg, S., and Young, R. (1998). Automatic synthesis of numerical codes for solving partial differential equations. *Math. Comput. Simul.*, 45:3–22.

[Akers et al., 1997] Akers, R., Kent, E., Randall, C., Steinberg, S., and Young, R. (1997). SciNapse: A problem-solving environment for partial differential equations. *IEEE Computational Science & Engineering Mag.*, 4(3):32–42.

[Alty et al., 1991] Alty, J. L., McCartney, C. D., and Zalloco, M. (November 1991). A multimedia interface support tool for process control interface design. Technical Report ESPRIT P2397, University of Loughborough.

[Alvarado, 1989] Alvarado, F. L. (1989). The Sparse Matrix Manipulation System. Technical Report ECE-89-1, Dept. of Elec. and Comp. Eng. , Univ. of Wisc., Madison, WI.

[Amarel, 1985] Amarel, S. (1985). Problems of representation in heuristic problem solving. In [Jernigan et al., 1985], pages 11–32.

[Anderson et al., 1995] Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Croz, J. D., Greenbaum, A., Hammerling, S., McKenney, A., Ostrouchov, S., and Sorensen, D. (1995). *LAPACK Users' Guide*. SIAM, Philadelphia, second edition.

[Arge et al., 1997a] Arge, E., Bruaset, A., and Langtangen, H., editors (1997a). *Modern Software Tools in Scientific Computing*. Birkhäuser.

[Arge et al., 1997b] Arge, E., Bruaset, A., and Langtangen, H. (1997b). Object-oriented numerics. In [Dæhlen and Tveito, 1997], pages 7–26.

[Arge et al., 1997c] Arge, E., Calvin, A. B. P., Kanney, J., Langtangen, H., and Miller, C. (1997c). On the numerical efficiency of C++ in scientific computing. In [Dæhlen and Tveito, 1997], pages 91–118.

[Arge and Hjelle, 1997] Arge, E. and Hjelle, O. (1997). Software tools for modelling scattered data. In [Dæhlen and Tveito, 1997], pages 45–60.

[Ashcraft et al., 1987] Ashcraft, C. C., Grimes, R. G., Lewis, J. G., Peyton, B. W., and Simon, H. D. (Dec. 1987). Progress in sparse matrix methods for large linear systems on vector supercomputers. *Int'l. J. Supercomput. Appl.*, 1(4):10–30.

[Atanassova and Herzberger, 1992] Atanassova, L. and Herzberger, J., editors (1992). *Computer Arithmetic and Enclosure Methods: Proc. Third IMACS-GAMM Symp. Computer Arithmetic and Scientific Computing (SCAN-91)*. North-Holland, Amsterdam.

[Avitzur et al., 1995] Avitzur, R., Bachmann, O., and Kajler, N. (1995). From honest to intelligent plotting. In [Levelt, 1995], pages 32–41.

[Balaban et al., 1989] Balaban, D., Garbarini, J., Greiman, W., and Durst, M. (1989). Knowledge representation for the automatic generation of numerical simulators for PDEs. *Math. Comput. Simul.*, 31:383–393.

[Balay et al.,] Balay, R., Vouk, M., and Perros, H. Performance of network based problem-solving environments. In this book.

[Ballance et al., 1993] Ballance, R., Giancola, A., Luger, G., and Ross, T. (1993). A framework-based environment for object-oriented scientific codes. *Scientific Programming*, 2(4):111–121.

[Baras et al., 1992] Baras, P., Blum, J., Paumier, J. C., Witomski, P., and Rechenmann, F. (1992). EVE: An object-centered knowledge based PDE solver. In [Houstis et al., 1992], pages 1–18.

[Barnes et al., 1992] Barnes, T., Harrison, D., Newton, A., and Spickelmier, R. (1992). *Electronic CAD Frameworks*. Kluwer Academic Pub., Boston.

[Barrett et al., 1993] Barrett, R., Berry, M., Chan, T., Demmell, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., and van der Vorst, H. (1993). *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, Philadelphia.

[Bass and Coutaz, 1991] Bass, L. and Coutaz, J. (1991). *Developing Software for the User Interface*. Addison-Wesley.

[Bau et al., 1988] Bau, H. H., Herbert, T., and Yovanovich, M. M., editors (1988). *Symbolic Computation in Fluid Mechanics and Heat Transfer*, New York. ASME, The American Society of Mechanical Engineers.

[Beamont et al.,] Beamont, O., Erhel, J., and Philippe, B. Aquarels: A problem-solving environment for validating scientific software. In this book.

[Bellen, 1991] Bellen, A. (Oct. 1991). ODE test problems. NA Digest 91(42), C. Moler ed.

[Benantar et al., 1990] Benantar, M., Biswas, R., Flaherty, J. E., and Shephard, M. S. (1990). Parallel computation with adaptive methods for elliptic and hyperbolic systems. *Comput. Meth. Appl. Mech. Engin.*, 82:73–93.

[Berman and Fateman, 1994] Berman, B. and Fateman, R. (1994). Optical character recognition for typeset mathematics. In [ISSAC, 1994], pages 348–353.

[Berners-Lee et al., 1994] Berners-Lee, T., Caillau, R., Luotonen, A., Nielsen, H., and Secret, A. (1994). The World-Wide Web. *Comm. ACM*, 37(8):76–82.

[Bertot and Théry, 1998] Bertot, Y. and Théry, L. (1998). A generic approach to building user interfaces for theorem provers. *J. Symbolic Computation*, 25:161–194.

[Berzins, 1997] Berzins, M. (1997). SPRINT2D software for convection dominated PDEs. In [Arge et al., 1997a], pages 63–80.

[Bijl, 1986] Bijl, A. (1986). *AI in architectural CAD*. Kogan Page, London.

[Bik, 1996] Bik, A. (1996). *Compiler support for sparse matrix computations*. PhD thesis, Rijksuniversiteit Leiden.

[Bischof et al., 1996] Bischof, C., Carle, A., and Mauer, A. (1996). Adifor 2.0: Automatic differentiation of Fortran 77 programs. *IEEE Computational Science & Engineering Mag.*, 3(3):18–32.

[Blackford et al., 1997] Blackford, L., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., and Whaley, R. (1997). *ScaLAPACK User's Guide*. SIAM, Philadelphia.

[Bliss et al., 1992] Bliss, B., Brunet, M.-C., and Gallopoulos, E. (1992). Automatic program instrumentation with applications in performance and error analysis. In Houstis, E., Rice, J., and Vichnevetsky, R., editors, *Expert Systems for Scientific Computing*, pages 235–260. Elsevier Science Pub. B. V. (North-Holland).

[Boisvert, 1994a] Boisvert, R. (1994a). A Web gateway to a virtual mathematical software repository. In *Electr. Proc. of Second Int'l. World Wide Web Conf.*. At URL http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings.

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

[Boisvert, 1997] Boisvert, R., editor (1997). *The Quality of Numerical Software, Assessment and Enhancement*. Chapman&Hall, London.

[Boisvert et al., 1996] Boisvert, R., Browne, S., Dongarra, J., and Grosse, E. (1996). Digital software and data repositories for support of scientific computing. In Adam, N., Bhargava, B., and Halem, M., editors, *Advances in Digital Libraries*, volume 1082 of *Lecture Notes in Computer Science*, pages 61–72. Springer-Verlag, New York.

[Boisvert et al., 1998] Boisvert, R., Dongarra, J., Pozo, R., Remington, K., and Stewart, G. W. (1998). Developing numerical libraries in Java. *Concurrency: Practice and Experience*, 10(11-13):1117–1129.

[Boisvert et al., 1979] Boisvert, R., Houstis, E., and Rice, J. (1979). A system for performance evaluation of partial differential equations software. *IEEE Transactions on Software Engineering*, SE-5(4):418–425.

[Boisvert et al., 1985] Boisvert, R., Howe, S., and Kahaner, D. (1985). GAMS – A framework for the management of scientific software. *ACM Transactions on Mathematical Software*, 11:313–355.

[Boisvert et al., 1991] Boisvert, R., Howe, S., and Kahaner, D. (1991). The guide to available mathematical software problem classification system. *Communications in Statistics, Part B: Simulation and Computation*, 20(4):811–842.

[Boisvert et al., 1997] Boisvert, R., Pozo, R., Remington, K., Barrett, R., and Dongarra, J. (1997). The Matrix Market: A Web repository for test matrix data. In [Boisvert, 1997], pages 125–137.

[Boisvert et al., 1992] Boisvert, R., Springmann, J., and Strawbridge, M. (Fall 1992). A virtual software repository system. In *Proc. Thirtieth Semi-Annual Cray User Group Meeting*, pages 68–72.

[Boisvert, 1989] Boisvert, R. F. (1989). The guide to available mathematical software advisory systems. *Math. Comput. Simul.*, 31:453–463.

[Boisvert and Kahaner, 1991] Boisvert, R. F. and Kahaner, D. K. (1991). DEQSOL and ELLPACK: Problem solving environments for partial differential equations. *Office of Naval Research Asian Office Scientific Information Bulletin (NAVSO P-3580)*, 16(1):7–19.

[Bonadio, 1989] Bonadio, A. (1989). *Theorist*. Prescience Corp., San Francisco.

[Bonadio, 1992] Bonadio, A. (1992). Mathematical user interfaces for graphical workstations. In [Gaffney and Houstis, 1992], pages 331–339.

[Bonomo and Dyksen, 1990] Bonomo, J. and Dyksen, W. R. (1990). XELLPACK: An interactive problem-solving environment for elliptic partial differential equations. In [Houstis et al., 1990c], pages 331–341.

[Boonmee and Kawata, 1998a] Boonmee, C. and Kawata, S. (1998a). Computer-Assisted Simulation Environment for Partial-Differential-Equation Problem: 1. Data Structure and Steering of Problem Solving Process. *Trans. of the Japan Society for Computational Engineering and Science*, (Paper No. 19980001).

[Boonmee and Kawata, 1998b] Boonmee, C. and Kawata, S. (1998b). Computer-Assisted Simulation Environment for Partial-Differential-Equation Problem: 2. Visualization and Steering of Problem Solving Process. *Trans. of the Japan Society for Computational Engineering and Science*, (Paper No. 19980002).

[Boonmee and Kawata, 1997] Boonmee, C. and Kawata, S. (July-August 1997). Visualization and Steering of Program Generation Process in NCAS System. In *Proc. IASTED Conf. Appl. Modeling and Simulation*, pages 96–99, Banff, Canada.

[Boonmee et al.,] Boonmee, C., Kawata, S., Fujii, S., Manabe, Y., and Tago, Y. Visual steering of the simulation process in scientific numerical simulation environment. In this book.

[Borst et al., 1994] Borst, W. N., Goldman, V. V., and Van Hulzen, J. A. (1994). GENTRAN 90: a REDUCE package for the generation of Fortran 90 code. In [ISSAC, 1994], pages 45–51.

[Boubez et al., 1992] Boubez, T., Froncioni, A., and Peskin, R. (March 1992). A prototyping environment for differential equations. *ACM Trans. Math. Softw.*, 18(1):1–10.

[Boudreaux, 1985] Boudreaux, J. C. (1985). Problem solving and the evolution of programming languages. In [Jernigan et al., 1985], pages 103–126.

[Bramley et al.,] Bramley, R., Gannon, D., Stuckey, T., Villacis, J., Akman, E., Balasubramanian, J., Breg, F., Diwan, S., and Govidaraju, M. The linear system analyzer. In this book.

[Breg et al., 1998] Breg, F., Diwan, S., Villacis, J., Balasubramanian, J., Akman, E., and Gannon, D. (1998). *Concurrency: Practice and Experience*, volume 10, chapter Java RMI performance and object model interoperability: experiments with Java/HPC++, pages 941–955. John Wiley&Sons, Ltd.

[Brodlie et al., 1992] Brodlie, K., Berzins, M., Dew, P., Poon, A., and Wright, H. (1992). Visualization and its use in scientific computations. In [Gaffney and Houstis, 1992], pages 293–304.

[Brodlie et al., 1993] Brodlie, K., Poon, A., Wright, H., Brankin, L., Banecki, G., and Gay, A. (1993). GRASPARC - A problem solving environment integrating computation and visualization. In Nielson, G. M. and Bergeron, D., editors, *Proceedings of the Visualization '93 Conference*, pages 102–109, San Jose, CA. IEEE Computer Society Press.

[Bronstein, 1993] Bronstein, M., editor (1993). *ISSAC'93: proceedings of the 1993 International Symposium on Symbolic and Algebraic Computation, July 6–8, 1993, Kiev, Ukraine*, New York, NY 10036, USA. ACM Press.

[Brooks, Jr., 1987] Brooks, Jr., F. P. (Apr. 1987). No silver bullet: Essence and accidents of software engineering. *IEEE Comput.*, 20:10–19.

[Broughan, 1992] Broughan, K. (1992). SENAC: Lisp as a platform for constructing a problem solving environment. In [Gaffney and Houstis, 1992], pages 351–360.

[Broughan et al., 1991] Broughan, K. A., Keady, G., Robb, T. D., Richardson, M. G., and Dewar, M. C. (July 1991). Some symbolic computing links to the NAG numeric library. *SIGSAM Bulletin*, 25(3):28–37.

[Bruaset and Langtangen, 1997a] Bruaset, A. and Langtangen, H. (1997a). A comprehensive set of tools for solving partial differential equations; Diffpack. In [Dæhlen and Tveito, 1997], pages 45–90.

[Bruaset and Langtangen, 1997b] Bruaset, A. and Langtangen, H. (1997b). Basic tools for linear algebra. In [Dæhlen and Tveito, 1997], pages 27–44.

[Bunnin et al., pear] Bunnin, F., Guo, Y., Ren, Y., and Darlington, J. (To appear). Design of high-performance financial modeling environment. *Parallel Computing*.

[Butler, 1996] Butler, G. (1996). Software architectures for computer algebra: a case study. In Calmet, J. and Limongelli, C., editors, *Design and Implementation of Symbolic Computation Systems*, volume 1128 of *Lecture Notes in Computer Science*, pages 277–286. Springer, Berlin. Proc. Int'l. Symp., DISCO'96, Karlsruhe.

[Butler and Cannon, 1990] Butler, G. and Cannon, J. (1990). The design of Cayley - a language for modern algebra. In [Miola, 1990], pages 10–19.

[Calmet, 1998] Calmet, J. (1998). Computer algebra and artificial intelligence. *Math. Comput. Simul.*, 45:73–82.

[Calmet and Limongelli, 1996] Calmet, J. and Limongelli, C., editors (1996). *Design and implementation of symbolic computation systems: International Symposium, DISCO '96, Karlsruhe, Germany, September 18–20, 1996: proceedings*, volume 1128 of *Lecture Notes in Computer Science*, Berlin, Germany / Heidelberg, Germany / London, UK / etc. Springer Verlag.

[Candy and Edmonds, 1995] Candy, L. and Edmonds, E. (1995). Creativity in knowledge work: A process model and requirements for support. In *Proc. OZCHI'95*, pages 242–248.

[Carey et al., 1992] Carey, G., Schmidt, J., Singh, V., and Yelton, D. (1992). A scalable, object-oriented finite element solver for partial differential equations on multicomputers. In *Proc. 6th ACM International Conference on Supercomputing*, pages 387–396, New York. ACM Press.

[Casanova and Dongarra, pear] Casanova, H. and Dongarra, J. (to appear). Using agent-based software for scientific computing in the NetSolve system. *Parallel Computing*.

[Casanova et al., 1998] Casanova, H., Dongarra, J., and Moore, K. (1998). Network-enabled solvers and the NetSolve project. *SIAM News*, 31(1).

[Cassanova and Dongarra, 1997] Cassanova, H. and Dongarra, J. (1997). NetSolve: A network server for solving computational science problems. *Int'l. J. Supercomput. Appl. and High Perf. Comput.*, 11:212–223.

[Chaitin-Chatelin and Frayssé, 1996] Chaitin-Chatelin, F. and Frayssé, V. (1996). *Lectures on Finite Precision Computations*. SIAM, Philadelphia.

[Chandy, 1994] Chandy, K. (1994). Concurrent program archetypes. In *Proc. Scalable Parallel Library Conf.*

[Chapman et al.,] Chapman, B., Haines, M., Mehrotra, P., Zma, H., and van Rosendale, J. Opus: A coordination language for multidisciplinary applications. *Scientific Programming*, 6(4):345–362.

[Char, 1986] Char, B., editor (1986). *Proceedings of the 1986 Symposium on Symbolic and Algebraic Computation: Symsac '86, July 21–23, 1986, Waterloo, Ontario*, New York, NY 10036, USA. ACM Press.

[Char et al., 1991] Char, B., Geddes, K., Gonnet, G., Leong, B., Monagan, M., and Watt, S. (1991). *Maple V Language Reference Manual*. New York.

[Char, 1989] Char, B. W. (1989). Automatic reasoning about numerical stability of rational expressions. In [Gonnet, 1989], pages 234–241.

[Char, 1990] Char, B. W. (1990). Progress report on a system for general-purpose symbolic algebraic computation. In [Watanabe and Nagata, 1990], pages 96–103.

[Chevenet, 1994] Chevenet, F. (April 1994). *Un environnement coopératif de résolution de problèmes pour l'analyse statistique en écologie*. PhD thesis, Laboratoire de Biométrie, Génétique et Biologie des Populations, URA CNRS n. 243, Université Claude Bernard, Lyon, France.

[Chew et al., 1995] Chew, P., Constable, R., Pingali, K., Vavasis, S., and Zippel, R. (1995). Collaborative mathematics environment. Dept. of Computer Sci., Cornell University. At URL http://www.cs.cornell.edu in /rz/MathBus95/TechSummary.html.

[Clarkson, 1992] Clarkson, M. (1992). Expert systems as an intelligent user interface for symbolic algebra. In [Gaffney and Houstis, 1992], pages 205–213.

[Codenotti et al.,] Codenotti, B., Leoncini, M., and Resta, G. A Java framework for Internet distributed computations. In this book.

[ComputationalStatistics.99, 1999] ComputationalStatistics.99 (1999). Comput. statistics. 14(1). Special Issue: Interactive graphical data analysis.

[Computer.Graphics.ACM.SIGRAPH, 1987] Computer.Graphics.ACM.SIGRAPH (Nov. 1987). *Comput. Graph.*, 21(6). Special issue on Visualization in Scientific Computing, edited by B. H. McCormick, T. A. DeFanti and M. D. Brown.

[Cook Jr., 1992] Cook Jr., G. (1992). Code generation in ALPAL using symbolic techniques. In [Wang, 1992], pages 27–35.

[Cook, Jr., 1990] Cook, Jr., G. O. (1990). ALPAL: A program to generate physics simulation codes from natural descriptions. *Int'l. J. Modern Phys.*, 1(1):1–51.

[Cook, Jr. and Painter, 1992] Cook, Jr., G. O. and Painter, J. F. (1992). ALPAL: A tool to generate simulation codes from natural descriptions. In [Houstis et al., 1992], pages 401–419.

[Cook, Jr. et al., 1991] Cook, Jr., G. O., Painter, J. F., and Brown, S. A. (Feb. 1991). How symbolic computation boosts productivity in the simulation of partial differential equations. Technical Report UCRL-JC-106442, Lawrence Livermore National Laboratory, Livermore.

[Cornea-Hasegan et al., 1994] Cornea-Hasegan, M., Costian, C., Marinescu, D., Martin, I., and Rice, J. (1994). Towards problem-solving environments for high performance computing. In *High Performance Computing'94*, pages 354–366, National Supercomputer Research Center, Singapore.

[Coughran, 1997] Coughran, W. (1997). Network-based scientific computation via Inferno. In [Boisvert, 1997], pages 267–269.

[Cowell, 1984] Cowell, W. R., editor (1984). *Sources and Development of Mathematical Software*. Prentice-Hall, Englewood Cliffs, NJ.

[Cryer, 1992] Cryer, C. (1992). The ESPRIT project FOCUS. In [Gaffney and Houstis, 1992], pages 371–380.

[Culler and Fried, 1963] Culler, G. and Fried, B. (1963). An on-line computing center for scientific problems. In *Proc. 1963 Pacific Computer Conf.*, pages 221–242. IEEE.

[Cybenko, 1996] Cybenko, G. (1996). Large-scope computing: A challenge for the 21st century. *IEEE Computational Science & Engineering Mag.*, 3(2):1,10.

[Czyzyk et al., 1997] Czyzyk, J., Owen, J., and Wright, S. (1997). NEOS: optimization on the internet. Technical Report OTC-97/04, Argonne National Laboratory.

[Czyzyk et al., 1999] Czyzyk, J., Wisniewski, T., and Wright, S. (1999). Optmization case studies in the NEOS guide. *SIAM Rev.*, 41(1):148–163.

[Dabdub et al.,] Dabdub, D., Chandy, K., , and Hewett, T. Mapping specificity and generality: Tailoring general archetypal PSEs to specific users. In this book.

[Dæhlen and Tveito, 1997] Dæhlen, M. and Tveito, A. (1997). *Numerical Methods and Software Tools in Industrial Mathematics*. Birkhäuser, Boston.

[Davenport, 1989] Davenport, J. H., editor (1989). *EUROCAL'87: European Conference on Computer Algebra, Leipzig, GDR, June 1987*, number 378 in Lecture Notes in Computer Science, Berlin. Springer-Verlag.

[Davies, 1994] Davies, R. (1994). Writing a matrix package in C++. In *OON-SKI'94 Proceedings of the Second Annual Object-Oriented Numerics Conference*, pages 207–213.

[Della Dora and Fitch, 1989] Della Dora, J. and Fitch, J., editors (1989). *Computer Algebra and Parallelism*. Academic Press, London.

[Demmel, 1997] Demmel, J. (1997). On parallel numerical software libraries. In Keyes, D., Sameh, A., and Venkatakrishnan, V., editors, *Parallel Numerical Algorithms*, pages 17–54. Kluwer Academic Publishers, Dordrecht, The Netherlands.

[Derby et al., 1996] Derby, T., Schnabel, R., and Zorn, B. (1996). A new language for prototyping numerical computation. *Scientific Programming*, 5:279–300.

[DeRose et al., 1995a] DeRose, L., Gallivan, K., Gallopoulos, E., Marsolf, B., and Padua, D. (1995a). FALCON: A MATLAB Interactive Restructuring Compiler. In C.-H. Huang, et al., editor, *Lecture Notes in Computer Science: Languages and Compilers for Parallel Computing*, pages 269–288. Springer-Verlag, New York.

[DeRose et al., 1995b] DeRose, L., Gallivan, K., Gallopoulos, E., Marsolf, B., and Padua, D. (Nov. 1995b). FALCON: An Environment for the Development of Scientific Libraries and Applications. In *Proc. of the First Int'l. Workshop on Knowledge-Based Systems for the (re)Use of Program Libraries (KBUP'95)*, pages 149–160, Sophia-Antipolis.

[DeRose et al.,] DeRose, L., Marsolf, B., Gallopoulos, E., and Padua, D. FALCON: An environment for the development of numerical programs using MATLAB. In this book.

[DeRose, 1996] DeRose, L. A. (1996). *Compiler Techniques for MATLAB Programs*. PhD thesis, University of Illinois at Urbana-Champaign.

[Dewar, 1992] Dewar, M. (1992). Using computer algebra to select numerical algorithms. In [Wang, 1992], pages 1–8.

[Dewar, 1989] Dewar, M. C. (1989). IRENA - an integrated symbolic and numerical computation environment. In [Gonnet, 1989], pages 171–179.

[Dewar and Richardson, 1990] Dewar, M. C. and Richardson, M. G. (1990). Reconciling symbolic and numeric computation in a practical setting. In [Miola, 1990], pages 195–204.

[Di Gregorio et al., 1997] Di Gregorio, S., Rongo, R., Spataro, W., Spezzano, G., and Talia, D. (1997). High performance scientific computing by a parallel cellular environment. *Future Generation Computer Systems*, 12:357–369.

[diffpack,] Diffpack. Numerical Objects (Diffpack) page. At URL http://www.nobjects.com/.

[Disz et al., 1995] Disz, T., Evard, R., Henderson, M., Nickless, W., Olson, R., Papka, M., and Stevens, R. (1995). Designing the future of collaborative science: Argonne's futures laboratory. *IEEE Parallel and Distributed Technology*, 3(2):14–21.

[Dodson et al., 1991] Dodson, D. S., Grimes, R. G., and Lewis, J. G. (1991). Sparse extensions to the Fortran basic linear algebra subprograms. *ACM Trans. Math. Softw.*, 17(2):253–263.

[Doi et al., 1997] Doi, S., Fujio, H., and Sugihara, K. (1997). Automatic parallel program generation for finite element analysis. In [Boisvert, 1997], pages 255–266.

[Doleh and Wang, 1990] Doleh, Y. and Wang, P. S. (1990). SUI: A system independent user interface for an integrated scientific computing environment. In [Watanabe and Nagata, 1990], pages 88–95.

[Dongarra et al., 1992] Dongarra, J., Lumsdaine, A., Niu, X., Pozo, R., and Remington, K. (1992). A sparse matrix library in C++ for high performance architectures. In *Proc. Second Object Oriented Numerics Conf.*, pages 214–218.

[Dongarra et al., 1978] Dongarra, J. J., Bunch, J. R., Moler, C. B., and Stewart, G. W. (1978). *LINPACK User's Guide*. SIAM Pub., Philadelphia, PA.

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

[Drashansky et al., 1997a] Drashansky, T., Houstis, E., Joshi, A., Rice, J., and Weerawarana, S. (1997a). Collaborating problem-solving agents for multi-physics problems. In *15th IMACS World Congress*, volume 4, pages 541–546. Wissenschaft and Technik Verlag.

[Drashansky et al., 1997b] Drashansky, T., Joshi, A., and Rice, J. (1997b). Multidisciplinary problem-solving using agents in a cluster environment. In *Cluster Computing Conference*. URL http://www.mathcs.edu/ ccc97/sessions.html, 4 pages.

[Drashansky et al., 1997c] Drashansky, T., Joshi, A., Rice, J., Houstis, E., and Weerawarana, S. (1997c). A multi-agent environment for MPSEs. In *Proc. Parallel Processing for Scientific Computing*. SIAM Pub. At the URL http://www.siam.org in /meetings/pp97/pp97home.html, 8 pages.

[du Toit et al., 1991] du Toit, D., George, P., Laug, P., Pate, P., Steer, D., and Vidrascu, M. (1991). Introduction á MODULEF, Guide n. 1.

[Dubois-Pélerin et al., 1992] Dubois-Pélerin, Y., Zimmermann, T., and Bomme, P. (1992). Object-oriented finite element programming: II. A prototype program in Smalltalk. *Comput. Meth. Appl. Mech. Engrg.*, 98(3):361–397.

[Dunbar, 1995] Dunbar, K. (1995). How scientists really reason: Scientific reasoning in real-world laboratories. In Sternberg, R. and Davidson, J., editors, *The Nature of Insight*, pages 365–395. The MIT Press, Cambridge, MA.

[Dupée and Davenport, 1996] Dupée, B. and Davenport, J. (1996). An intelligent interface to numerical routines. In Calmet, J. and Limongelli, C., editors, *Design and Implementation of Symbolic Computation Systems*, volume 1128 of *Lecture Notes in Computer Science*, pages 252–262. Springer, Berlin. Proc. Int'l. Symp., DISCO'96, Karlsruhe.

[Duval and Jung, 1992] Duval, D. and Jung, F. (1992). Examples of problem solving using computer algebra. In [Gaffney and Houstis, 1992], pages 133–141.

[Dyksen and Ribbens, 1987] Dyksen, W. and Ribbens, C. (1987). Interactive ELLPACK: An interactive problem-solving environment for elliptic partial differential equations. *ACM Trans. Math. Softw.*, 13:113–132.

[Dyksen and Gritter, 1992] Dyksen, W. R. and Gritter, C. R. (1992). Scientific computing and the algorithm selection problem. In [Houstis et al., 1992], pages 19–32.

[Edmonds and Candy, 1993a] Edmonds, E. and Candy, L. (1993a). Knowledge support for conceptual design: The amplification of creativity. In *Proc. HCI International*, pages 350–355.

[Edmonds and Candy, 1993b] Edmonds, E. and Candy, L. (1993b). Knowledge support systems for conceptual design: The amplification of creativity. In *Proceedings of the Fifth International Conference on Human-Computer Interaction*, volume 2 of *II. Software Tools*, pages 350–355.

[Engquist and Smedsaas, 1980] Engquist, B. and Smedsaas, T. (1980). Automatic computer code generation for hyperbolic and parabolic differential equations. *SIAM J. Sci. Stat. Comput.*, 1:249–259.

[Engquist and Smedsaas, 1984a] Engquist, B. and Smedsaas, T. (1984a). Automatic analysis in PDE software. In [Engquist and Smedsaas, 1984b], pages 399–409.

[Engquist and Smedsaas, 1984b] Engquist, B. and Smedsaas, T., editors (1984b). *PDE Software: Modules, Interfaces and Systems*. North-Holland.

[Erdélyi et al., 3 55] Erdélyi, A., Magnus, W., Oberhettinger, F., and Tricomi, F. (1953-55). *Higher Transcendental Functions*, volume 1-3. McGraw Hill, New York.

[Erhel and Philippe, 1991] Erhel, J. and Philippe, B. (July 1991). Aquarels: A problem-solving environment for numerical quality. In Vichnevetsky, R. and Miller, J., editors, *Proc. IMACS Conf.*, pages 45–46, Dublin.

[Fateman,] Fateman, R. Problem solving environments and symbolic computing. In this book.

[Fateman, 1990] Fateman, R. J. (Aug. 1990). Advances and trends in the design and construction of algebraic manipulation systems. In [Watanabe and Nagata, 1990], pages 60–67.

[Fateman, 1989] Fateman, R. J. (March 1989). A review of Macsyma. *IEEE Trans. Knowledge and Data Eng.*, 1(1):133–145.

[Fateman and Kahan, 1987] Fateman, R. J. and Kahan, W. (1987). Improving exact integrals from symbolic algebra systems. Unpublished note.

[Feldman, 1992] Feldman, S. (1992). Environments for large-scale scientific computation. In [Gaffney and Houstis, 1992], pages 147–152.

[Fenves et al., 1984] Fenves, S. J., Maher, M. L., and Sriram, D. (Oct. 1984). Expert systems: C.E. potential. *Civil Engineering*, 54:44–48.

[Ferris et al., 1998] Ferris, M., Fourer, R., and Gay, D. (1998). Expressing complementarity problems in an algebraic modeling language and communicating them to solvers. Technical Report MP-TR-98-02, University of Wisconsin, Madison.

[Field, 1986] Field, D. A. (1986). From solid modeling to finite element analysis. In Kunii, T. L., editor, *Application Development Systems*, pages 220–249. Springer-Verlag, Tokyo.

[Fitch, 1990] Fitch, J. (1990). A delivery system for REDUCE. In [Watanabe and Nagata, 1990], pages 76–81.

[Fitch, 1989] Fitch, J. P. (1989). Can REDUCE be run in parallel? In [Gonnet, 1989], pages 155–162.

[Fitch and Hall, 1987] Fitch, J. P. and Hall, R. G. (1987). Symbolic computation and the finite element method. In [Davenport, 1989], pages 95–96.

[Fleeter et al., pear] Fleeter, S., Houstis, E., Rice, J., and Zhou, C. (to appear). GasTurbnLab: A problem-solving environment for gas turbine engine simulation. *Comp. Engineering Science Journal*.

[Ford and Chatelin, 1987] Ford, B. and Chatelin, F., editors (1987). *Problem Solving Environments for Scientific Computing: Proc. IFIP TC 2/WG 2.5 Working Conf.* North-Holland, Amsterdam.

[Ford and Iles, 1987] Ford, B. and Iles, R. (1987). The what and why of problem-solving environments for scientific computing. In [Ford and Chatelin, 1987], pages 3–18.

[Foster and Kesselman, 1998] Foster, I. and Kesselman, C., editors (1998). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Mateo.

[Foster et al., 1998] Foster, I., von Laszewski, G., Thiruvanthukal, G., and Toonen, B. (1998). A computational framework for telemedicine. *Future Generation Computer Systems*, 14:109–123.

[Fourer and Gay, 1995] Fourer, R. and Gay, D. M. (1995). Expressing special structures in an algebraic modeling language for mathematical programming. *ORSA J. Computing*, 7(2):166–190.

[Fourer et al., 1990] Fourer, R., Gay, D. M., and Kernighan, B. W. (1990). A modeling language for mathematical programming. *Management Science*, 36(5):519–554. See also the URL http://www.ampl.com/cm/cs/what/ampl/REFS/index.html.

[Fox, 1997a] Fox, G. (1997a). Editorial Preface to Special Issue on *Java for Computational Science and Engineering - Simulation and Modeling. Concurrency: Practice and Experience*, 9(6):413–414.

[Fox, 1997b] Fox, G. (1997b). Editorial Preface to Special Issue on *Java for Computational Science and Engineering - Simulation and Modeling II. Concurrency: Practice and Experience*, 9(11):1001–1002.

[Fox and Furmanski, 1997a] Fox, G. and Furmanski, W. (1997a). Java for parallel computing and as a general language for scientific and engineering simulation and modeling. *Concurrency: Practice and Experience*, 9(6):415–425.

[Fox and Furmanski, 1997b] Fox, G. and Furmanski, W. (Mar./Apr. 1997b). Petaops and Exaops: Supercomputing on the Web. *IEEE Internet Computing Mag.*, 1(2):38–46.

[Frederickson and Mackerle, 1983] Frederickson, B. and Mackerle, J. (1983). Partial list of major finite element programs and description of some of their capabilities. In Noor, A. and Pikley, W., editors, *State of the Art Surveys on Finite Element Technology*, pages 363–403. The American Society of Mechanical Engineers.

[Frick, 1985] Frick, I. (1985). SHEEP and classification in general relativity. In Buchberger, B. and Caviness, B. F., editors, *Proc. EUROCAL'85, vol. 2*, number 204 in Lecture Notes in Computer Science, pages 161–162, Berlin. Springer-Verlag.

[Fritzson and Fritzson, 1992] Fritzson, P. and Fritzson, D. (1992). The need for high-level programming support in scientific computing applied to mechanical analysis. *Computers and Structures*, 45(2):387–395.

[Fujita et al., 1999] Fujita, A., Teramoto, T., Minami, K., Boonmee, C., Moriguchi, K., Toriyama, K., Tago, Y., and Kawata, S. (1999). Computer-Assisted Parallel program Generation System P-NCAS from mathematical Model - Visualization and Steering of Parallel Program Generation Process. In *Proc. Conf. Computational Engineering and Science*, volume 4:1, pages 257–260.

[Gaffney and Houstis, 1992] Gaffney, P. and Houstis, E. N., editors (1992). *Programming Environments for High-Level Scientific Problem Solving.* North-Holland, Amsterdam.

[Gallivan et al., 1994] Gallivan, K., Jalby, W., Marsolf, B., and Sameh, A. (1994). On the Development of Libraries and Their Use in Applications. Technical Report 1341, Center for Supercomputing Research and Development, University of Illinois.

[Gallivan and Marsolf, 1994] Gallivan, K. and Marsolf, B. (1994). Practical issues related to developing object-oriented numerical libraries. In *OON-SKI'94 Proceedings of the Second Annual Object-Oriented Numerics Conference*, pages 93–106.

[Gallivan et al., 1997] Gallivan, K., Marsolf, B., and Gallopoulos, E. (1997). On the use of algebraic and structural information in a library prototyping and development environment. In *Proc. 15th IMACS World Congress on Scientific Computation, Modelling and Applied Mathematics*, volume 4, pages 565–570, Berlin.

[Gallivan et al., 1990] Gallivan, K. A., Heath, M. T., Ng, E., Ortega, J. M., Peyton, B. W., Plemmons, R. J., Romine, C. H., Sameh, A., and Voigt, R. G. (1990). *Parallel Algorithms for Matrix Computations.* SIAM, Philadelphia.

[Gallopoulos et al., 1994] Gallopoulos, E., Houstis, E., and Rice, J. (1994). Computer as thinker/doer: Problem solving environments for CSE. *IEEE Computational Science & Engineering Mag.*, 1(2):11–23.

[Gallopoulos et al., 1995] Gallopoulos, E., Houstis, E., and Rice, J. (1995). Workshop on problem-solving environments: Findings and recommendations. *ACM Computing Surveys*, 27(2):277–279.

[Gallopoulos et al., 1992] Gallopoulos, E., Houstis, E., and Rice, J. (Oct. 1992). *Future Research Directions in Problem Solving Environments for Computational Science.* Report of a Workshop on Research Directions in Integrating Numerical Analysis, Symbolic Computing, Computational Geometry, and Artificial Intelligence for Computational Science. Technical Report 1259, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign. Workshop held at NSF, Washington, D.C., Apr. 1991.

[Gallopoulos and Sameh, 1997] Gallopoulos, E. and Sameh, A. (1997). CSE: Content and product. *IEEE Computational Science & Engineering Mag.*, 4(2):39–43.

[GAMS,] GAMS. NIST guide to available mathematical software (GAMS). At URL http://math.nist.gov.

[Gander et al., 1996] Gander, W., Masaryk, J., and Hrebicek, J. (1996). *Solving Problems in Scientific Computing Using Maple and Matlab.* Springer-Verlag, Boston, third edition.

[Gannon et al., 1998] Gannon, D., Bramley, R., Stuckey, T., Villacis, J., Balasubramanian, J., Akman, E., Breg, F., Diwan, S., and Govindaraju, M. (1998). Component architectures for distributed scientific problem-solving. *IEEE Computational Science & Engineering Mag.*, 5(2):50–63.

[Ganzha and Liska, 1989] Ganzha, V. and Liska, R. (1989). Application of the REDUCE computer algebra system to stability analysis of difference schemes. In Kaltofen, E. and Watt, S., editors, *Proc. Computers and Mathematics '89*, pages 119–129. Springer-Verlag, New York.

[Ganzha and Vorozhtsov, 1993] Ganzha, V. and Vorozhtsov, E. (1993). A probabilistic symbolic-numerical method for the stability analyses of difference schemes for PDEs. In [Bronstein, 1993], pages 9–13.

[Garbey et al., 1991] Garbey, M., Kaper, H. G., and Kwong, M. K. (1991). Symbolic manipulation software and the study of differential equations. In Kaper, H. G. and Garbey, M., editors, *Asymptotic Analysis and the Numerical Solution of Partial Differential Equations*, volume 130 of *Lecture Notes in Pure and Applied Mathematics*, pages 241–265. Marcel Dekker, Inc., New York.

[Geddes and Fee, 1992] Geddes, K. O. and Fee, G. J. (1992). Hybrid symbolic-numeric integration in MAPLE. In [Wang, 1992], pages 36–41.

[Gianni, 1989] Gianni, P., editor (1989). *Symbolic and algebraic computation: International Symposium ISSAC '88, Rome, Italy, July 4–8, 1988: proceedings*, volume 358 of *Lecture Notes in Computer Science*, Berlin, Germany / Heidelberg, Germany / London, UK / etc. Springer Verlag. Conference held jointly with AAECC-6.

[Gilbert et al., 1992] Gilbert, J. R., Moler, C., and Schreiber, R. (1992). Sparse matrices in MATLAB: Design and implementation. *SIAM J. Matrix Anal. Appl.*, 13(1):333–356.

[Golding et al., 1998] Golding, D., Venneri, S., and Noor, A. (1998). Beyond incremental change. *IEEE Computer*, 31(10):31–39.

[Goldman and Gabriel, 1989] Goldman, R. and Gabriel, R. P. (July 1989). Qlisp: Parallel processing in Lisp. *IEEE Software*, 6(4):51–59.

[Goldman and Cats, 1996] Goldman, V. and Cats, G. (1996). Adjoint modelling within a program generation framework: A case study for a weather forecasting grid-point model. In Berz, M., Bischof, C., Corliss, G., and Griewank, A., editors, *Computational Differentiation*. SIAM, Philadelphia.

[Goldman et al., 1995] Goldman, V., van Hulzen, J., Mynett, A., Posthuma, A., and van Zuylen, H. (1995). The application of computer algebra for the discretization and coding of the Naver-Stokes equations. In *Computer Algebra in Industry 2*, pages 131–149. John Wiley & Sons.

[Gonnet, 1989] Gonnet, G., editor (1989). *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation: ISSAC '89 / July 17–19, 1989, Portland, Oregon*, New York, NY 10036, USA. ACM Press.

[Gray et al., 1996] Gray, S., Kajler, N., and Wang, P. (1996). Pluggability issues in the multi protocol. In Calmet, J. and Limongelli, C., editors, *Design and Implementation of Symbolic Computation Systems*, volume 1128 of *Lecture Notes in Computer Science*, pages 343–356. Springer, Berlin. Proc. Int'l. Symp., DISCO'96, Karlsruhe.

[Gray et al., 1998] Gray, S., Kajler, N., and Wang, P. (1998). Design and implementation of MP, a protocol for efficient exchange of mathematical expressions. *J. Symbolic Computation*, 25:217–237.

[Greenberg, 1991] Greenberg, D. P. (Feb. 1991). Computers and architecture. *Scientific American*, pages 104–109.

[Gries et al., 1989] Gries, D., Walker, T., and Young, P. (1989). 1988 Snowbird report: A discipline matures. *IEEE Comput.*, 22(2):72–75.

[Griewank and Corliss, editors, 1991] Griewank, A. and Corliss, editors, G. F. (1991). *Automatic Differentiation of Algorithms: Theory, Implementation and Application*. SIAM, Philadelphia.

[Gross et al., 1991] Gross, L., Sternercker, P., and Schönauer, W. (1991). The finite element tool package, VECFEM (version 1.1). Technical Report 45/91, University of Karlsruhe.

[Grosse, 1997] Grosse, E. (1997). Real inferno. In [Boisvert, 1997], pages 270–279.

[Grosz,] Grosz, L. VECFEM: The solver for non-linear partial differential equations. In this book.

[Gustafsson, 1993] Gustafsson, K. (1993). Object-oriented implementation of software for solving ordinary differential equations. *Scientific Programming*, 2:217–225.

[Hague, 1992] Hague, S. (1992). Using FOCUS technology to build front ends. In [Gaffney and Houstis, 1992], pages 383–392.

[Hammer et al., 1993] Hammer, R., Hocks, M., Kulisch, U., and Ratz, D. (1993). *Numerical Tolbox for Verified Computing I. Basic Numerical Problems*, volume 21 of *Series in Computational Mathematics*. Springer-Verlag, Berlin.

[Hariri et al.,] Hariri, S., Topcuoglou, H., Furmanski, W., Kim, D., Kim, Y., , Ilkyeun, R., Bing, X., Bouquing, Y., and Valente, J. A problem-solving environment for network computing. In this book.

[Hartmanis and Lin, 1992] Hartmanis, J. and Lin, H., editors (1992). *Computing the Future: A broader agenda for computer science and engineering*. National Academy Press, Washington, D.C.

[Hartwig, 1976] Hartwig, R. (1976). Interactive storing, retrieval, transformation, and evaluation of scientific data with an experimental system. *Informatik Fachberichte, GI-6.JAHRESTAGUNG Springer-Verlag*, 5:251–266.

[Hearn, 1996] Hearn, A. (1996). Computer algebra and the World Wide Web. In Calmet, J. and Limongelli, C., editors, *Design and Implementation of Symbolic Computation Systems*, volume 1128 of *Lecture Notes in Computer Science*, pages 263–270. Springer, Berlin. Proc. Int'l. Symp., DISCO'96, Karlsruhe.

[Hearn, 1971] Hearn, A. C. (1971). Reduce 2: A system and language for algebraic manipulation. In [Petrick, 1971], pages 128–133.

[Hearn, 1991] Hearn, A. C. (1991). Algebraic computation: The quiet revolution. In Lifschitz, V., editor, *Artificial Intelligence and Mathematical Theory of Computation. Papers in Honor of John McCarthy*, pages 177–186. Academic Press, San Diego.

[Hearn, 1995] Hearn, A. C. (1995). *REDUCE User's Manual*. The Rand Corporation, Santa Monica.

[Hearn et al., 1989] Hearn, A. C., Boyle, A., and Caviness, B. F. (1989). Symbolic Computation: Directions for Future Research. Report of a Worshop on Symbolic and Algebraic Computation. Philadelphia.

[Heath and Etheridge, 1991] Heath, M. T. and Etheridge, J. A. (Sep. 1991). Visualizing the Performance of Parallel Programs. *IEEE Software*, 8(5):29–39.

[Hewett, 1995] Hewett, T. (1995). Towards a generic strategy for empirical evaluation of interactive computing systems. In Perlman, G., Green, G., and Wolgalter, M., editors, *Human Factors Perspective on Human-Computer Interaction*, pages 167–171. Human Factors and Ergonomics Society, Santa Monica, CA.

[Hewett and DePaul,] Hewett, T. and DePaul, J. Toward a human centered scientific problem-solving environment. In this book.

[Hewett and Meadow, 1986] Hewett, T. and Meadow, C. (1986). On designing for usability: An application of four key principles. In *Proc. Conference on Human Factors in Computing Systems (CHI'86)*, New York. ACM Press.

[Higham, 1995] Higham, N. (Sept. 1995). The Test Matrix Toolbox for MATLAB (version 3.0). Technical Report 276, Manchester Centre for Computational Mathematics.

[Hilfinger and Collela, 1989] Hilfinger, P. N. and Collela, P. (1989). FIDIL: A language for scientific programming. In Grossman, R., editor, *Symbolic Computation: Applications to Scientific Computing*, pages 97–138. SIAM, Philadelphia.

[Hindmarsh, 1983] Hindmarsh, A. C. (1983). ODEPACK, A systematized collection of ODE solvers. In R. S. Stepleman, et al., editor, *Scientific Computing*, pages 55–64. North Holland, Amsterdam.

[Hindmarsh, 1984] Hindmarsh, A. C. (1984). ODE solvers for time-dependent PDE software. In [Engquist and Smedsaas, 1984b], pages 325–341.

[Hoffmann et al., 1994] Hoffmann, C., Houstis, E., Rice, J., Catlin, A., Gaitatzes, M., Weerawarana, S., Wang, N.-H. L., Takoudis, C., and Taylor, D. (1994). SoftLab - A virtual laboratory for computational science. *Math. Comput. Simul.*, 36:479–491.

[Houstis et al., 1998] Houstis, E., , Rice, J., Weerawarana, S., Catlin, A., Gaitatzes, G., Papachiou, P., and Wang, K. (1998). PELLPACK: A Problem-Solving Environment for PDE-based Applications on Multicomputer Platforms *ACM Transactions on Mathematical Software*, 24(1):30–73.

[Houstis et al., 1997a] Houstis, E., Gallopoulos, E., Bramley, R., and Rice, J. (1997a). Problem-solving environments for computational science. *IEEE Computational Science & Engineering Mag.*, 4(3):18–21.

[Houstis et al., 1995a] Houstis, E., Joshi, A., and Rice, J. (1995a). MPSE: Multidisciplinary problem-solving environments (white paper). In *America in the Age of Information: A Forum*. Comm. on Information and Communications, Nat. Sci. Tech. Council. At URL http://www.hpcc.gov/cic/forum/White_Papers, 10 pages.

[Houstis et al., 1997b] Houstis, E., Joshi, A., Rice, J., Drashansky, T., and Weerawarana, S. (1997b). Towards multidisciplinary problem-solving environments. *High Performance Computing Users News*, 1. At URL http://www.hpcu.org/hpcunews/vol1, 6 pages.

[Houstis et al., 1990a] Houstis, E., Papatheodorou, T., and Rice, J. (1990a). Parallel ELLPACK: An expert system for the parallel processing of partial differential equations. In [Houstis et al., 1990c], pages 63–73.

[Houstis and Rice, 1992a] Houstis, E. and Rice, J. (1992a). The architecture of PDE solving systems. In Vichnevetsky, R., editor, *Computer Methods for Partial Differential Equations*, volume VII, pages 363–370, New Brunswick, NJ. IMACS.

[Houstis and Rice, 1992b] Houstis, E. and Rice, J. (1992b). Parallel ELLPACK, a development environment and problem solving environment for high performance computing machines. In [Gaffney and Houstis, 1992], pages 229–241.

[Houstis et al., 1990b] Houstis, E., Rice, J., Chrisochoides, N. P., Karathanasis, H. C., Papachiou, P. N., Samartzis, M. K., Vavalis, E. A., Wang, K.-Y., and Weerawana, S. (1990b). //Ellpack: A numerical simulation programming environment for parallel MIMD machines. In *Proc. 1990 Int'l Conf. Supercomput.*, pages 96–107, Amsterdam. ACM.

[Houstis et al., 1997c] Houstis, E., Rice, J., Markus, S., and Weerawarana, S. (1997c). Network based scientific problem-solving environments. *High Performance Computing Users News*. URL http://www.hpcu.org/hpcunews/vol1/issue1/netswp.html, 5 pages.

[Houstis et al., 1994a] Houstis, E., Rice, J., and Weerawarana, S. (1994a). An open structure for PDE solving systems. In *Proc. 14th IMACS World Congress*, volume 3, pages 1296–1299. IMACS.

[Houstis et al., 1994b] Houstis, E., Rice, J., and Weerawarana, S. (1994b). A software platform for integrating symbolic computation with a PDE solving environment. In *Proc. 14th IMACS World Congress*, volume 1, pages 482–485. IMACS.

[Houstis et al.,] Houstis, E., Verykios, V., Catlin, A., Ramakhrisnan, N., and Rice, J. A testing and data mining environment for scientific software. In this book.

[Houstis et al., 1995b] Houstis, E., Weerawarana, S., Joshi, A., and Rice, J. (1995b). The PYTHIA project. In S.K. Aityan, et al., editor, *Neural, Parallel, and Scientific Computations*, pages 215–218. Dynamic Publications.

[Houstis et al., 1990c] Houstis, E. N., Rice, J., and Vichnevetsky, R., editors (1990c). *Intelligent Mathematical Software Systems*. North-Holland, Amsterdam.

[Houstis et al., 1992] Houstis, E. N., Rice, J., and Vichnevetsky, R., editors (1992). *Expert Systems for Scientific Computing*. North-Holland, Amsterdam.

[hpcc, 1991] hpcc (1991). *Grand Challenges: High Performance Computing and Communications. A report by the committee on Physical, Mathematical, and Engineering Sciences*. Office of Science and Technology Policy.

[Hunt and Cremer, 1997] Hunt, K. and Cremer, J. (1997). Refiner: A problem solving environment for ODE/DAE simulations. *SIGSAM Bulletin (ACM Special Interest Group on Symbolic and Algebraic Manipulation)*, 31(3):42–43. Poster abstract only.

[ibm.special.issue.visualization, 1991] ibm.special.issue.visualization (1991). *IBM J. Res. Dev.*, 35(1/2). Special Issue on Visual Interpretation of Complex Data.

[Isenhour et al., 1997] Isenhour, P., Begole, J., Heagy, W., and Shaffer, C. (1997). Slieve: A Java-based collaborative visualization environment. In *Proc. IEEE Visualization'97 (Late Breaking Hot Topics), Phoenix, AZ*, pages 13–16, Los Alamitos.

[ISSAC, 1994] ISSAC (1994). *ISSAC '94: Proceedings of the 1994 International Symposium on Symbolic and Algebraic Computation: July 20–22, 1994, Oxford, England, United Kingdom*, New York, NY 10036, USA. ACM Press.

[Jackson et al., 1991] Jackson, R. H. F., Boggs, P. T., Nash, S. G., and Powell, S. (1991). Guidelines for reporting results of computational experiments. Report of the ad hoc committee. *Math. Progr.*, 49:413–425.

[Jacobson et al., 1992] Jacobson, P., Kågström, B., and Rännar, M. (1992). Algorithm development for distributed memory multicomputers using CONLAB. *Scientific Programming*, 1:185–203.

[Jenks and Sutor, 1992] Jenks, R. and Sutor, R. (1992). *Axiom: The Scientific Computation System*. Springer Verlag.

[Jenks et al., 1988] Jenks, R., Sutor, R., and Watt, S. (1988). Skratchpad II: An abstract datatype system for mathematical computation. In Rice, J., editor, *Mathematical Aspects of Scientific Software*, volume 14 of *The IMA Volumes in Mathematics and its Applications*, pages 157–182. Springer-Verlag, New York.

[Jernigan et al., 1985] Jernigan, R., Hamill, B. W., and Weintraub, D. W., editors (1985). *The Role of Language in Problem Solving*, volume I. North-Holland, Amsterdam.

[Joshi et al., 1997] Joshi, A., Drashansky, T., Rice, J., Weerawarana, S., and Houstis, E. (1997). Multiagent simulation of complex heterogeneous models in scientific computing. *Math. Comput. Simul.*, 44:43–59.

[Joshi et al.,] Joshi, A., Ramakrishnan, N., and Houstis, E. Multiagent systems to support networked scientific computing. In this book.

[Joshi et al., 1994] Joshi, A., Weerawarana, S., and Houstis, E. (1994). The use of neural networks to support intelligent scientific computing. In *Proc. IEEE International Conference on Neural Networks*, pages 2197–2202. IEEE Press.

[Joshi et al., 1996] Joshi, A., Weerawarana, S., Ramakrishnan, N., Houstis, E., and Rice, J. (1996). *IEEE Computational Science & Engineering Mag.*, 3(3):44–46.

[Kajler, 1992] Kajler, N. (1992). CAS/PI: A Portable and extensible interface for computer algebra systems. In [Wang, 1992], pages 376–386.

[Kajler and Soiffer, 1998] Kajler, N. and Soiffer, N. (1998). A survey of user interfaces for computer algebra systems. *J. Symbolic Computation*, 25:127–159.

[Kamel and Enright, 1992] Kamel, M. and Enright, W. H. (1992). ODEXPERT: A knowledge based system for automatic selection of initial value ODE system solvers. In [Houstis et al., 1992], pages 33–54.

[Kant, 1993] Kant, E. (1993). Synthesis of mathematical modeling software. *IEEE Software*, 10(3):30–41.

[Kant et al., 1990] Kant, E., Daube, F., MacGregor, W., and Wald, J. (1990). Automated synthesis of finite difference programs. In [Noor et al., 1990], pages 45–61.

[Kant and Steinberg, 1997] Kant, E. and Steinberg, S. (1997). Automatic program synthesis from abstract pde specifications. In *Proc. 15th IMACS World Congress on Scientific Computation, Modeling and Applied Mathematics*, pages 24–29, Berlin.

[Kawata et al., 1997] Kawata, S., Boonmee, C., Teramoto, T., Drska, L., Limpouch, J., Liska, R., and Sinor, M. (1997). Computer-assisted particle-in-cell code development. Technical Report NIFS-533, Nat'l. Inst. for Fusion Science.

[Kawata et al., 1999] Kawata, S., Fujita, A., Machide, S., Hirama, T., and Yoshimoto, K. (1999). Computer-assisted simulation system ncas and a future pse. In *Proc. 2nd Problem Solving environment(PSE) Workshop*, pages 89–94.

[Kawata et al., 1994] Kawata, S., Iijima, K., Boonmee, C., and Manabe, Y. (1994). Computer assisted scientific computation / simulation software development system - include a visualization system. *IFIP Transactions*, A-48:145–153.

[Kawata et al., 1996] Kawata, S., Takura, N., and Manabe, Y. (1996). Grid generation with orthogonality and uniformity of line-spacing ratio. *Computer Physics Communications*, 94:19–24.

[Kessler, 1996] Kessler, C. (1996). Pattern-driven automatic parallelization. *Scientific Programming*, 5:251–274.

[Klerer and Reinfelds, 1968] Klerer, M. and Reinfelds, J. (1968). *Interactive Systems for Experimental Applied Mathematics*. Academic Press, New York.

[Knox et al., 1997] Knox, R., Kalb, V., Levine, E., and Kendig, D. (1997). A problem-solving workbench for interactive simulation of ecosystems. *IEEE Computational Science & Engineering Mag.*, 4(3):52–60.

[Konno et al., 1990] Konno, C., Umetani, Y., Igai, M., and Ohta, T. (1990). Interactive/visual DEQSOL: Interactive creation, debugging, diagnosis, and visualization of numerical simulation. In [Houstis et al., 1990c], pages 301–317.

[Konno et al., 1987] Konno, C., Yamabe, M., Saji, M., Sagawa, N., Umetani, Y., Hirayama, H., and Ohta, T. (1987). Automatic code generation method of DEQSOL. *J. Inform. Proc.*, 11(1):15–21.

[Kredel, 1989] Kredel, H. (1989). Software development for computer algebra or from ALDES/SAC-2 to WEB/Modula-2. In [Gianni, 1989], pages 447–455. Conference held jointly with AAECC-6.

[Kuck, 1992] Kuck, D. (1992). A user's view of high-performance scientific and engineering software systems in the mid-21st century. In [Houstis et al., 1992], pages 69–87.

[Kuck, 1994] Kuck, D. (1994). What do users of parallel computers really need? *Int'l. J. Parallel Programming*, 22(1):99–127.

[Kuck, 1996] Kuck, D. (1996). *High Performance Computing: Challenges for Future Systems*. Oxford Univ. Press, New York.

[Kulisch, 1981] Kulisch, U. (1981). *Computer Arithmetic in Theory and Practice*. Academic Press, New York.

[Kulisch, 1997] Kulisch, U. (1997). The XSC tools for extended scientific computing. In [Boisvert, 1997], pages 280–285.

[Laffey and Musser, 1996] Laffey, J. and Musser, D. (Oct. 1996). Building internet-based electronic performance support for teaching and learning. WebNet'96, San Francisco, CA.

[Lakshman, 1996] Lakshman, Y., editor (1996). *ISSAC '96: Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation, July 24–26, 1996, Zurich, Switzerland*, New York, NY 10036, USA. ACM Press.

[Lakshman et al.,] Lakshman, Y., Miller, M., and Lombeyda, S. TechTALK: A Web based system for mathematical collaboration. In this book.

[Langtangen, 1999] Langtangen, H. (1999). *Computational Partial Differential Equations, Numerical Methods and Diffpack Programming*, volume 2 of *Lecture Notes in Computational Science and Engineering*. Springer-Verlag, Berlin.

[Lau, 1996] Lau, L. (1996). *Implementation of Scientific Applications on Heterogeneous Parallel Architectures*. PhD thesis, University of Queensland.

[Laub, 1985] Laub, A. (1985). Numerical linear algebra aspects of control design computations. *IEEE Trans. Automat. Control*, AC-30:97–108.

[Leake, 1996] Leake, D. (1996). Case-based selection of problem-solving methods for scientific computation. At http://www.cs.indiana.edu/hyplan/leake/cbmatrix.html.

[Lee and Gannon, 1991] Lee, J. K. and Gannon, D. (November 1991). Object-oriented parallel programming experiments and results. *Proc. Supercomputing'91*, pages 273–282.

[Leuze, editor, 1990] Leuze, editor, M. R. (1990). Scalable parallel libraries workshop report. Preproceedings of a workshop conducted at Oak Ridge National Laboratory.

[Levelt, 1995] Levelt, A., editor (1995). *ISSAC '95: Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation: July 10–12, 1995, Montreal, Canada*, ISSAC -PROCEEDINGS- 1995, New York, NY 10036, USA. ACM Press.

[Li et al., 1999a] Li, M., Rana, O., and Walker, D. (Nov. 1999a). CB-PSE: A Component-Based Problem Solving Environment. In *Proc. First Int'l. Conf. on Information Reuse and Integration*, pages 7–10. International Society of Computers and their Applications.

[Li et al., 1999] Li, Z., Reif, J., and Gupta, S. (1999). Synthesizing efficient out-of-core programs for block recursive algorithms using block-cyclic distributions. *IEEE Trans. Paral. Distr. Syst.*, 10(3):297–315.

[Licklider, 1960] Licklider, J. (1960). Man-computer symbiosis. *IRE Transactions on Human Factors in Electronics*.

[Liska et al., 1993] Liska, R., Shashkov, M., and Solovjov, A. (1993). Support-operators method for PDE discretization: Symbolic algorithms and realization. *Math. Comput. Simul.*, pages 173–183.

[Lowry and McCartney, 1991] Lowry, M. and McCartney, R., editors (1991). *Automating software design*. AAAI Press / The MIT Press, Menlo Park, CA.

[Lucks and Gladwell, 1992] Lucks, M. and Gladwell, I. (March 1992). Automated selection of mathematical software. *ACM Trans. Math. Softw.*, 18(1):11–34.

[Maeder, 1996] Maeder, R. (1996). *Programming in Mathematica*. Addison-Wesley, Boston, third edition.

[Manabe et al., 1997] Manabe, Y., Kawata, S., and Boonmee., C. (1997). Use of stored-numerical-data information extracted from a computing program in scientific visualization. *Int. J. Modeling and Simulation*, 17:166–168.

[Marchant et al., 1996] Marchant, M., Weatherill, N., Turner-Smith, E., Zheng, Y., and Sotirakos, M. (1996). A parallel simulation user environment for computational engineering. In Soni, B., Thompson, J., Haeuser, J., and Eiseman, P., editors, *Numerical Grid Generation in Computational Field Simulations (Proc. 5th International Conference, Mississippi, USA, 1996)*, volume 1, pages 741–751. Mississippi State University.

[Markus et al., 1997] Markus, S., Weerawarana, S., Houstis, E., and Rice, J. (1997). Scientific computing via the Web: The Net Pellpack PSE server. *IEEE Computational Science & Engineering Mag.*, 4(3):43–51.

[Marsolf et al., 1998] Marsolf, B., Gallivan, K., and Gallopoulos, E. (1998). The interactive restructuring of MATLAB programs using the FALCON environment. In Veidenbaum, A. and Joe, K., editors, *Innovative Architecture for Future Generation High-Performance Processors and Systems*, pages 3–12. IEEE Press.

[Marsolf et al., 1999] Marsolf, B., Gallivan, K., and Wijshoff, H. (1999). The utilization of matrix structure to generate optimized code from MATLAB programs. *Int'l. J. Parallel Programming*, 27(2):73–96.

[Marsolf, 1997] Marsolf, B. A. (1997). *Techniques for the Interactive Development of Numerical Linear Algebra Libraries for Scientific Computation*. PhD thesis, University of Illinois at Urbana-Champaign.

[Martin and Fateman, 1971] Martin, W. A. and Fateman, R. J. (1971). The MACSYMA system. In [Petrick, 1971], pages 59–75.

[Marzinkewitsch, 1991] Marzinkewitsch, R. (1991). Operating computer algebra systems by handprinted input. In [Watt, 1991], pages 411–413.

[mathml,] MATHML, the World-Wide Web consortium. At URL http://www.w3.org/Math.

[matlab5.3, 1999] matlab5.3 (1999). MATLAB 5.3. URL http://www.mathworks.com.

[Mehrotra and van Rosendale, 1987] Mehrotra, P. and van Rosendale, J. (1987). The BLAZE language: A parallel language for scientific programming. *Parallel Comput.*, 5:339–361.

[Menon and Trefethen, 1997] Menon, V. and Trefethen, A. (1997). MultiMATLAB: Integrating MATLAB with high-performance parallel computing. In *Proc. Supercomputing'97*.

[Messina and Sterling, 1993] Messina, P. and Sterling, T., editors (1993). *System Software and Tools for High Performance Computing Environments*. SIAM, Philadelphia.

[Miller and Wrathall, 1980] Miller, W. and Wrathall, C. (1980). *Software for roundoff analysis of matrix algorithms*. Academic Press, New York.

[Miola, 1990] Miola, A., editor (1990). *DISCO'90: Design and Implementation of Symbolic Computation Systems*. Number 429 in Lecture Notes in Computer Science. Springer-Verlag, Berlin.

[Mitchell, 1990] Mitchell, W. J. (1990). Afterword: The design studio of the future. In McCullough, M., Mitchell, W. J., and Purcell, P., editors, *The Electronic Design Studio*, pages 479–494. MIT Press, Cambridge, MA.

[Mitchell and McCullough, 1991] Mitchell, W. J. and McCullough, M. (1991). *Digital Design Media. A Handbook for Architects & Design Professionals*. Van Nostrand Reinhold, New York.

[Moore et al., 1989] Moore, P. K., Ozturan, C., and Flaherty, J. E. (1989). Towards the automatic numerical solution of partial differential equations. *Math. Comput. Simul.*, 31:325–332.

[Mossberg et al., 1997] Mossberg, E., Otto, K., and Thuné, M. (1997). Object-oriented software tools for the construction of preconditioners. *Scientific Programming*, 6(3):285–295.

[Mu, 1999] Mu, M. (1999). Solving composite problems with interface relaxation. *SIAM J. Sci. Comput.*, 20(4):1394–1416.

[Noor et al., 1990] Noor, A., Elishakoff, I., and Hulbert, G., editors (1990). *Symbolic Computations and their Impact on Mechanics*, volume PVP-205. The American Society of Mechanical Engineers, New York.

[Norman and Fitch, 1996] Norman, A. and Fitch, J. (1996). Interfacing REDUCE to Java. In Calmet, J. and Limongelli, C., editors, *Design and Implementation of Symbolic Computation Systems*, volume 1128 of *Lecture Notes in Computer Science*, pages 271–276. Springer, Berlin. Proc. Int'l. Symp., DISCO'96, Karlsruhe.

[Novitski, 1991] Novitski, B. J. (Aug. 1991). CADD holdouts. *Architecture*, 80(8):97–99.

[O'Boyle and Bull, 1996] O'Boyle, M. and Bull, J. (1996). Expert programming versus parallelizing compiler: A comparative study of two approaches for distributed shared memory. *Scientific Programming*, 5:63–68.

[Oden, 1991] Oden, J. T. (1991). Smart algorithms and adaptive methods for compressible and incompressible flow: Optimization of the computational process. In Mesirov, J. P., editor, *Very Large Scale Computation in the 21st Century*, pages 87–99. SIAM, Philadelphia.

[OON,] OON. The Object-Oriented Numerics page. At URL http://oonumerics.org.

[OpenMath,] The OpenMath Society page. At URL http://www.openmath.org/index.html.

[Oppe et al., 1989] Oppe, T. C., Joubert, W. D., and Kincaid, D. R. (1989). An overview of NSPCG: A nonsymmetric preconditioned conjugate gradient package. *Comput. Phys. Commun.*, 53:283–293.

[Paalvast et al., 1990] Paalvast, E. M., van Gemund, A. J., and Sips, H. J. (1990). A method for parallel program generation with an application to the Booster language. In *Proc. 1990 Int'l. Conf. Supercomput.*, pages 457–469, New York. ACM Press.

[Pantazopoulos and Houstis, 1997] Pantazopoulos, K. and Houstis, E. (1997). Modern software techniques for computational finance. In [Arge et al., 1997a], pages 227–246.

[Paolini and Santangelo, 1991] Paolini, G. V. and Santangelo, P. (1991). An interactive graphic tool to plot the structure of large sparse matrices. In [ibm.special.issue.visualization, 1991], pages 231–237. Special Issue on Visual Interpretation of Complex Data.

[Parker et al., 1998] Parker, S., Miller, M., Hansen, C., and Johnson, C. (1998). An integrated problem solving environment: the SCIRun computational steering system. In *31st Hawaii International Conference on System Sciences (HICSS-31)*. submitted.

[Parker et al., 1997] Parker, S., Weinstein, D., and Johnson, C. (1997). The SCIRun computational steering software system. In [Arge et al., 1997a], pages 1–40.

[Parlett, 1978] Parlett, B. N. (July 1978). Progress in numerical analysis. *SIAM Rev.*, 20(3):443–455.

[Paxson and Saltmarsh, 1993] Paxson, V. and Saltmarsh, C. (1993). Glish: A use-level software bus for loosely-coupled distributed systems. In *Proc. Winter'93 USENIX Conference*, pages 271–276. Usenix Association.

[Peskin, 1990] Peskin, R. L. (1990). Symbolic manipulation in engineering user interface systems. In [Noor et al., 1990], pages 97–111.

[Peskin et al., 1990] Peskin, R. L., Walther, S. S., and Froncioni, A. M. (1990). SMALLTALK – The next generation scientific computing interface? In [Houstis et al., 1990c], pages 257–267.

[Petrick, 1971] Petrick, S. R., editor (1971). *Proc. Second Symposium on Symbolic and Algebraic Manipulation. Los Angeles, California*, New York. ACM SIGSAM, ACM Press.

[Phanouriou and Abrams, 1997] Phanouriou, C. and Abrams, M. (1997). Transforming command-line driven systems to Web applications. In *Sixth International World Wide Web Conference*, pages 1–3, Santa Clara, CA.

[Piessens et al., 1983] Piessens, R., de Doncker-Kapenga, E., Uberhuber, C. W., and Kahaner, D. K. (1983). *Quadpack, A subroutine package for automatic integration*. Springer-Verlag, Berlin.

[Ponder, 1988] Ponder, C. G. (1988). *Evaluation of "Performance Enhancements" in algebraic manipulation systems*. PhD thesis, University of California, Berkeley. Also Tech. Rep. UCB 88/438.

[President's Information Technology Advisory Committee (PITAC), 1999] President's Information Technology Advisory Committee (PITAC) (Feb. 1999). *Report to the President. Information Technology Research: Investing in Our Future*. Also at URL http://www.ccic.gov/ac/report.

[Press et al., 1992] Press, W., Teukolsky, S., Vettering, W., and Flannery, B. (1992). *Numerical Recipes in FORTRAN. The Art of Scientific Computing*. Cambridge University Press, Cambridge, second edition.

[PSE,] Problem Solving Environments home page. At URL http://www.cs.purdue.edu in /research/cse/pses.

[PSEware, 1996] PSEware (1996). The PSEware project: A toolkit for building problem solving environments. At URL http://www.extreme.indiana.edu/pseware.

[Purtilo, 1986a] Purtilo, J. (1986a). Applications of a software interconnection system in mathematical problem solving environments. In [Char, 1986], pages 16–23.

[Purtilo, 1989] Purtilo, J. M. (1989). Minion: an environment to organize mathematical problem solving. In [Gonnet, 1989], pages 147–154.

[Purtilo, 1992] Purtilo, J. M. (1992). Dynamic software reconfiguration supports scientific problem solving activites. In [Gaffney and Houstis, 1992], pages 245–254.

[Purtilo, 1994] Purtilo, J. M. (Jan. 1994). The Polylith software bus. *ACM Trans. Progr. Lang. Syst.*, 16(1):151–174.

[Purtilo, 1986b] Purtilo, J. M. (Sept. 1986b). A software interconnection technology to support specification of computational environments. Technical Report R-86-1269, Department of Computer Science, University of Illinois at Urbana-Champaign.

[Purtilo et al., 1988] Purtilo, J. M., Reed, D. A., and Grunwald, D. C. (1988). Environments for prototyping parallel algorithms. *J. Paral. Dist. Comput.*, 5:421–437.

[Radford and Stevens, 1988] Radford, A. and Stevens, G. (1988). *CADD Made Easy: A Comprehensive Guide for Architects & Designers*. McGraw-Hill Book Co., New York.

[Ramakrishnan et al., 1998] Ramakrishnan, N. ., Houstis, E. ., and Rice, J. . (1998). Recommender systems for problem-solving networks. In *Recommender Systems, AAAI Report WS-98-08*, pages 91–95. AAAI Press, Menlo Park, CA.

[Ramakrishnan et al., 1997] Ramakrishnan, N., Joshi, A., Houstis, E., and Rice, J. (1997). Neuro-fuzzy approaches to collaborative scientific computing. In *Proc. International Conference on Neural Networks (ICNN'97)*, volume 1, pages 473–478. IEEE Press.

[Ramakrishnan et al., 1995] Ramakrishnan, N., Joshi, A., Weerawarana, S., Houstis, E., and Rice, J. (1995). Neuro-fuzzy systems for intelligent scientific computing. In Dagli, et al., C., editor, *Intelligent Engineering Through Artificial Neural Networks, Vol. 5: Fuzzy Logic and Evolutionary Programming*, pages 279–284. ASME Press, New York.

[Rana et al., 1999] Rana, O., Li, M., Walker, D., and Shields, M. (Sept. 1999). An XML Based Component Model for Generating Scientific Applications and Performing Large Scale Simulations in a Meta-Computing Environment. In *Proc. of the First Int'l. Symposium on Generative and Component-Based Software Engineering, Erfurt, Germany*. Only available on CD-ROM.

[Rana and Walker, 1999] Rana, O. and Walker, D. (June 1999). Bringing Together Mobile Agents and Numerical Analysis in PSEs. In Arabnia, H., editor, *Proc. 1999 Int'l. Conf. Parallel and Distributed Processing Techniques and Applications, las Vegas, USA*. CSREA Press.

[Randall et al., 1998] Randall, C., Kant, E., and Chhabra, A. (1998). Using program synthesis to price derivatives. *Journal of Computational Finance*, 1(2):97–128.

[Rechenmann and Rousseau, 1992] Rechenmann, F. and Rousseau, B. (1992). A development shell for knowledge based systems in scientific computing. In [Houstis et al., 1992], pages 157–173.

[Regli, 1997] Regli, W. (1997). Internet-enabled computer-aided design. *IEEE Internet Computing*, 1(1):39–50.

[Renes et al., 1991] Renes, W., Vanbegin, M., Dooren, P. V., and Beckers, J. (July 1991). The MATLAB gateway compiler. A tool for automatic linking of Fortran routines to MATLAB. In *IFAC Symp. on CADCS*, pages 95–100, Swansea, UK.

[Resnick and Varian, 1997] Resnick, P. and Varian, H. (1997). Recommender systems. *Comm. ACM*, 40(3):56–58.

[Rice, 1971] Rice, J. (1971). *Mathematical Software*. Academic Press, New York.

[Rice, 1988] Rice, J. (1988). Mathematical aspects of scientific software. In Rice, J., editor, *Mathematical Aspects of Scientific Software*, volume 14 of *The IMA Volumes in Mathematics and its Applications*, pages 1–40. Springer-Verlag, New York.

[Rice, 1989] Rice, J. (1989). Libraries, software parts and problem-solving systems. In Fosdick, C. and Huang, H.-C., editors, *Symposium on Scientific Software*, pages 191–203. Tsinghua University Press.

[Rice, 1990] Rice, J. (1990). Mathematical software and ACM publications. In Nash, S. G., editor, *A History of Scientific Computing*, pages 217–227. ACM Press, Addison Wesley, Reading, Mass.

[Rice, 1992a] Rice, J. (1992a). Future research directions in Problem Solving Environments for Computational Science. In [Gaffney and Houstis, 1992], pages 363–369.

[Rice, 1992b] Rice, J. (1992b). *Numerical Methods, Software and Analysis*. McGraw-Hill, New York, second edition.

[Rice, 1994] Rice, J. (Spring 1994). Academic programs in computational science & engineering. *IEEE Computational Science & Engineering Mag.*, 1(1):13–21.

[Rice, 1995] Rice, J. (Winter 1995). Computational science and the future of computing research. *IEEE Computational Science and Engineering Magazine*, pages 35–41.

[Rice and Boisvert, 1996] Rice, J. and Boisvert, R. (1996). From scientific software librares to problem-solving environments. *IEEE Computational Science & Engineering Mag.*, 3(3):44–52.

[Rice and Boisvert, 1985] Rice, J. and Boisvert, R. F. (1985). *Solving Elliptic Problems using ELLPACK*. Springer-Verlag, New York.

[Rice and Rosen, 1966] Rice, J. and Rosen, S. (1966). NAPSS - A numerical analysis problem solving system. In *Proc. ACM Nat'l. Conf.*, pages 51–56.

[Rice and Schwetman, 1983] Rice, J. and Schwetman, H. D. (1983). Interface issues in a software parts technology. In *ITT Proc. Workshop on Reusability in Programming*, pages 129–137. Reprinted in *Software Reusability*, (P. Freeman, ed.), IEEE Tutorial, Computer Soc. Press, 1987, pp. 96-104. Revised version in *Software Reusability*, (T. Biggerstaff and A. J. Perlis, eds.), ACM Press, 1989, pp. 125-139.

[Rice, 1976] Rice, J. R. (1976). Statistical computing: The vanguard of the revolution in education. In Hoaglin, D. and Welsh, R., editors, *Ninth Interface Symposium on Computer Science and Statistics*, pages 1–4. Prindle, Weber & Schmidt, Boston.

[Rosing and Schnabel, 1987] Rosing, M. and Schnabel, R. (1987). An overview of DINO - a new language for numerical computation on distributed memory multiprocessors. In Rodrigue, G., editor, *Proc. Third SIAM Conf. Parallel Processing for Sci. Comput.*, pages 312–316, Philadelphia. SIAM.

[Ruppelt and Wirtz, 1989] Ruppelt, T. and Wirtz, G. (1989). Automatic transformation of high-level object-oriented specifications into parallel programs. *Parallel Computing*, 10(1):15–28.

[Saad, 1990] Saad, Y. (August 1990). SPARSKIT: A basic tool kit for sparse matrix computation. Technical Report 1029, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign.

[Sagawa et al., 1992] Sagawa, N., Rinn, D., and Hurley, N. (1992). An integrated problem solving environment for numerical simulation of engineering problems. In [Gaffney and Houstis, 1992], pages 191–199.

[Schultze and Cryer, 1988] Schultze, K. and Cryer, C. W. (1988). NAXPERT: a prototype expert system for numerical software. *SIAM J. Sci. Stat. Comput.*, 9:503–515.

[Schwarz, 1988] Schwarz, F. (Sept. 1988). Symmetries of differential equations: From Sophus Lie to Computer Algebra. *SIAM Review*, 30(3):450–481.

[Seager, 1989] Seager, M. K. (1989). A SLAP for the masses. In Carey, G. F., editor, *Parallel Supercomputing: Methods, Algorithms and Applications*, pages 135–155. John Wiley & Sons, Chichester.

[Sewell, 1985] Sewell, G. (1985). *Analysis of Finite Element Method-PDE/PROTRAN*. Springer-Verlag.

[Sewell, 1993] Sewell, G. (1993). PDE2D: Easy to use software for general two-dimensional partial differential equations. *Advances in Engineering Software*, 17:105–112.

[Shah, 1998] Shah, A. (1998). Symphony: A Java-based composition and manipulation framework for distributed legacy resources. Master's thesis, Dept. Computer Sciences, Virginia Polytechnic Institute and State University, Blacksburg, VA.

[Sharma, 1988] Sharma, N. (1988). Generating finite element programs for Warp machine. In [Bau et al., 1988], pages 93–102.

[Sharma and Wang, 1990] Sharma, N. and Wang, P. S. (1990). Generating finite element programs for shared memory multiprocessors. In [Noor et al., 1990], pages 63–79.

[Shneiderman, 1985] Shneiderman, B. (1985). Overcoming limitations imposed by current programming languages. In [Jernigan et al., 1985], pages 253–275.

[Simon, 1981] Simon, H. A. (1981). *The Sciences of the Artificial*. MIT Press, Cambridge, Mass., second edition.

[Singh et al., 1994] Singh, J., Gupta, A., and Levoy, M. (July 1994). Parallel visualization algorithms: Performance and architectural implications. *IEEE Computer*, 27(7):45–55.

[Smarr and Catlett, 1992] Smarr, L. and Catlett, C. (1992). Metacomputing. *Comm. ACM*, 35(6):44–52.

[Smith, 1994] Smith, B. (Sept. 1994). Extensible PDE solvers package users manual. Technical Report ANL-94/40, Argonne National Laboratory, Argonne, IL.

[Smith et al., 1997] Smith, T., Gower, A., and Boning, D. (1997). A matrix math library for Java. *Concurrency: Practice and Experience*, 9(11):1127–1137.

[Soiffer, 1991] Soiffer, N. M. (1991). *The design of a user interface for computer algebra systems*. PhD thesis, University of California, Berkeley, CA.

[Soloway et al., 1996] Soloway, E., Jackson, S., Klein, J., Quintana, C., Reed, J., Spitulnik, J., Stratford, S., Studer, S., Eng, J., and Scala, N. (1996). Learning theory in practice: Case studies of learner-centered design. In Tauber, M., Bellotti, V., Jeffries, R., Mackinlay, J., and Nielsen, J., editors, *Proceedings of the Conference on Human Factors in Computing Systems : Common Ground*, pages 189–196, New York. ACM Press.

[Stevenson, 1994] Stevenson, D. (Dec. 1994). Science, Computational Science, and Computer Science: At crossroads. *Comm. ACM*, 37(12):85–96.

[Subcommittee on Computing, Information, and Communications R&D., 1999] Subcommittee on Computing, Information, and Communications R&D. (April 1999). *High Perf. Computing and Commun.: Information Technology Frontiers for a New Millenium*. Also at URL http://www.ccic.gov/pubs/blue00/.

[Swayne and Klinke, 1999] Swayne, D. and Klinke, S. (1999). Introduction to the special issue on interactive graphical data analysis. In [ComputationalStatistics.99, 1999], pages 1–6. Special Issue: Interactive graphical data analysis.

[Szelényi and Zecca, 1991] Szelényi, F. and Zecca, V. (1991). Visualizing parallel execution of FORTRAN programs. In [ibm.special.issue.visualization, 1991], pages 270–282. Special Issue on Visual Interpretation of Complex Data.

[Tan, 1988] Tan, H. (1988). Symbolic derivation of material property matrices in finite element analysis. In [Bau et al., 1988], pages 111–116.

[Techexplorer,] Techexplorer. The techexplorer hypermedia browser home page. At the URL site http://www.software.ibm.com/enetwork/techexplorer.

[Teitelman, 1996] Teitelman, W. (1996). Pilot: A step toward man-computer symbiosis. Technical Report MAC-TR-32, Project Mac, MIT.

[ten Cate, 1993] ten Cate, H. (1993). *Towards formal specification and proof of finite element software within the ATES development system.* PhD thesis, University of Twente, The Netherlands.

[ten Cate, 1995] ten Cate, H. (1995). Applying abstraction and formal specification in numerical software design. *Comput. Math. Appl.*, 29(12):81–102.

[ten Cate and Vollebregt, 1996] ten Cate, H. and Vollebregt, E. (1996). On the portability and efficiency of parallel algorithms and software. *Paral. Comput.*, 29(22):1149–1163.

[Thuné, 1986] Thuné, M. (July 1986). Automatic GKS stability analysis. *SIAM J. Sci. Stat. Comput.*, 7(3):959–977.

[Tong, 1989] Tong, S.-S. (1989). Coupling symbolic manipulation and numerical simulation for complex engineering designs. *Math. Comput. Simul.*, 31:419–430.

[Tong, 1992] Tong, S.-S. (1992). Integration of symbolic and numerical methods for optimizing complex engineering systems. In [Gaffney and Houstis, 1992], pages 3–18.

[Tuchman and Berry, 1990] Tuchman, A. and Berry, M. (1990). Matrix Visualization in the Design of Numerical Algorithms. *ORSA Journal of Computing*, 2(1):84–92.

[Ullrich, 1992] Ullrich, C. P. (1992). The programming toolbox of the numerical analyst. In [Atanassova and Herzberger, 1992], pages 69–85.

[Umetani et al., 1992] Umetani, Y., Konno, C., and Ohta, T. (1992). Visual PDEQSOL: A visual and interactive environment for numerical simulation. In [Gaffney and Houstis, 1992], pages 259–267.

[Umetani et al., 1987] Umetani, Y., Tsuji, M., Iwasawa, K., and Hirayama, H. (1987). DEQSOL: A numerical simulation lanuage for vector/parallel processors. In [Ford and Chatelin, 1987], pages 147–162.

[Unwin, 1999] Unwin, A. (1999). Requirements for interactive graphics software for exploratory data analysis. *Comput. Statistics*, 14(1):7–22.

[van den Boom et al., 1991] van den Boom, A., Brown, A., Dumortier, F., Geurts, A., Hammarling, S., Kool, R., Vanbegin, M., Dooren, P. V., and Huffel, S. V. (July 1991). SLICOT, a subroutine library for control and system theory. In *Preprints IFAC Symp. on CADCS*, pages 89–94, Swansea, UK.

[van den Heuvel et al., 1989] van den Heuvel, P., van Hulzen, J. A., and Goldman, V. V. (1989). Automatic generation of FORTRAN-coded Jacobians and Hessians. In [Davenport, 1989], pages 120–131.

[van Engelen, 1998] van Engelen, R. (1998). *CTADEL: A generator of efficient numerical codes.* PhD thesis, Rijksuniversiteit Leiden.

[van Engelen et al., 1997] van Engelen, R., Heitlager, I., Wolters, L., and Cats, G. (1997). Incorporating application dependent information in an automatic code generating environment. In *Proc. 11th ACM Int'l. Conf. Supercomputing*, pages 180–187, New York. ACM.

[van Engelen et al.,] van Engelen, R., Wolters, L., and Cats, G. The CTADEL application driver. In this book.

[van Liere et al., 1997] van Liere, R., Mulder, J., and van Wijk, J. (1997). Computational steering. *Future Generation Computer Systems*, 12:441–450.

[Vanecek, 1989] Vanecek, G. (Nov. 1989). Protosolid: An inside look. Technical Report CAPO-89-26, Dept. Comput. Sci., Purdue University.

[Vermeulen and Chapman, 1993] Vermeulen, A. and Chapman, M. (1993). OON-SKI: An introduction. *Scientific Programming*, 2:109–110. Introduction to special issue devoted to the First Annual Object-Oriented Numerics Conference (OON-SKI).

[Viklund and Fritzson, 1995] Viklund, L. and Fritzson, P. (1995). ObjectMath - An object-oriented language and environment for symbolic and numerical processing in scientific computing. *Scientific Programming*, 4:229–250.

[Vollebregt, 1997] Vollebregt, E. (1997). Abstract level parallelization of finite difference methods. *Scientific Programming*, 6(4):331–344.

[Walker, 1999] Walker, D. (June 1999). Advanced environments and tools for high performance computing: Report on the 1999 European Research Conference on Problem-Solving Environments. This file is now available from the URL `http://www.cs.cf.ac.uk:8008/User/David.W.Walker/`.

[Wang, 1986] Wang, P. (1986). Finger: A symbolic system for automatic generation of numerical programs in finite element analysis. *Journal of Symbolic Computation*, 2:305–316.

[Wang, 1992] Wang, P., editor (1992). *Proceedings of ISSAC '92. International Symposium on Symbolic and Algebraic Computation*, New York, NY 10036, USA. ACM Press.

[Wang, 1990] Wang, P. S. (1990). Applying advanced computing techniques in finite element analysis. In [Noor et al., 1990], pages 45–61.

[Watanabe and Nagata, 1990] Watanabe, S. and Nagata, M., editors (1990). *ISSAC '90: proceedings of the International Symposium on Symbolic and Algebraic Computation: August 20–24, 1990, Tokyo, Japan*, New York, NY 10036, USA and Reading, MA, USA. ACM Press and Addison-Wesley.

[Watt, 1986] Watt, S. (1986). *Bounded parallelism and computer algebra*. PhD thesis, Univ. Waterloo. Avail. Univ. Waterloo CS dept. tech. rep. CS-86-12.

[Watt, 1991] Watt, S., editor (1991). *ISSAC '91: proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation, July 15–17, 1991, Bonn, Germany*, New York, NY 10036, USA. ACM Press.

[WebEQ,] WebEQ. WebEQ, a suite of Java programs for creating and diplaying interactive scientific Web documents . At URL http://www.webeq.com/webeq.

[Weerawarana, 1994] Weerawarana, S. (1994). *Problem solving environments for partial differential equation based applications*. PhD thesis, Department of Computer Sciences, Purdue University.

[Weerawarana et al., 1995] Weerawarana, S., Houstis, E., Catlin, A., and Rice, J. (1995). //ELLPACK: A system for simulating partial differential equations. In deSilva, C. and Hanza, M., editors, *Modeling and Simulation*, pages 122–126. IASTED-ACTA Press, Anaheim, CA.

[Weerawarana et al., 1992] Weerawarana, S., Houstis, E., and Rice, J. (1992). An interactive symbolic-numeric interface to parallel ELLPACK for building general PDE solvers. In Donald, B., Kapur, D., and Mundy, J., editors, *Symbolic and Numerical Computation for Artificial Intelligence*, pages 303–321. Academic Press.

[Weerawarana et al., 1994] Weerawarana, S., Houstis, E., Rice, J., Catlin, A., Crabhill, C., Chui, C., and Markus, S. (1994). PDELab: An object-oriented framework for building problem-solving environments for PDE based applications. In Vermeulen, A., editor, *Proc. Second Annual Object-Oriented Numerics Conference*, pages 79–92, Corvallis, OR. Rogue-Wave Software.

[Weerawarana et al.,] Weerawarana, S., Houstis, E., Rice, J., Catlin, A., Gaitatzes, M., Crabhill, C., Markus, S., and Drashansky, T. Towards a kernel for building PSEs. In this book.

[Weerawarana et al., 1996] Weerawarana, S., Houstis, E., Rice, J., Joshi, A., and Houstis, C. (1996). PYTHIA: A knowledge based system to select scientific algorithms. *ACM Trans. Math. Softw.*, 22(4):447–468.

[Weerawarana et al., 1997] Weerawarana, S., Joshi, A., Houstis, E., Rice, J., and Catlin, A. (1997). Notebook interfaces for networked scientific computing: Design and WWW implementation. *Concurrency: Practice and Experience*, 9:675–695.

[Weerawarana and Wang, 1992] Weerawarana, S. and Wang, P. (1992). A portable code generator for Cray Fortran. *ACM Trans. Math. Softw.*, 18:241–255.

[Weerawarana and Wang, 1989] Weerawarana, S. and Wang, P. S. (1989). GENCRAY: a portable code generator for Cray Fortran. In [Gonnet, 1989], pages 186–191.

[Willamowski et al., 1994] Willamowski, J., Chevenet, F., and Jean-Marie, F. (1994). A development shell for cooperative problem-solving environments. *Math. Comput. Simul.*, 36:361–379.

[Wolfram, 1999] Wolfram, S. (1999). *The Mathematica Book*. Cambridge University Press, Cambridge, fourth edition.

[Wu and Houstis, 1997] Wu, P. and Houstis, E. (1997). EPPOD: A problem-solving environment for parallel electronic prototyping of physical objects design. *J. Paral. Distr. Comput.*, 42:157–172.

[Zhao et al., 1996] Zhao, Y., Sakurai, T., Sugiura, H., and Torii, T. (1996). A methodology for parsing mathematical notation for mathematical computation. In [Lakshman, 1996], pages 292–300.

[Zheng et al.,] Zheng, Y., Weatherhill, N., Turner-Smith, E., Sotirakos, M., Marchant, M., and Hassan, O. Visual steering of grid generation in a parallel simulation user environment. In this book.

[Zimmermann et al., 1992] Zimmermann, T., Dubois-Pélerin, Y., and Bomme, P. (1992). Object-oriented finite element programming: I. Governing principles. *Comput. Meth. Appl. Mech. Engrg.*, 98:291–303.