Problem Solving Environments for Computational Science

The theme of this book is the rapidly evolving technology of problem-solving environments (PSEs), defined in [Gall 91] as "a computer system that provides all the computational facilities necessary to solve a target class of problems", for scientific computing. It is meant to provide an assessment of the state-of-the-art and a roadmap to the future in the area of PSEs for scientific computing. The 1991 [Gall 94] and 1995 [Rice 95] workshops on PSEs for physical simulations defined this research area and identified the pertinent research issues. The first workshop made several recommendations that led to the 1995 NSF initiative on PSEs. These efforts have identified several PSE design goals together with the approaches for realizing them including: 1) reduce the difficulty of physical simulation by utilizing user-natural languages and application-specific terminology and automating many lower level computational tasks, 2) reduce cost and time to develop complex scientific software by reusing large and small software components, raising the level of abstraction in software, creating components for symbolic mathematics, geometric modeling, scientific visualization, etc., 3) increase the availability of scientific software components by providing a comprehensive catalog of components, a search and delivery system for components, a standard terminology and set of interfaces for them, 4) provide the infrastructure to build families of applications for science and engineering PSEs including a generic software architecture and associated kernel for such PSEs, facilities for the composition of software parts and the insertion of new parts, and facilities to measure and test properties of scientific software, 5) identify and foster a community of PSE builders, 6) raise the reliability of physical simulations by allowing the reuse of reliable components and provide test beds for new components, 7) extend the lifetime of software by creating encapsulation methodology for legacy software, using interfaces that are programming language independent, using open ended type system for software parts, and which provide extensive facilities for transforming types and representations. It was concluded in these workshops that to realize the above PSE design objectives we need to address the following research challenges:

1. Develop PSE architecture that supports the plug-and-play paradigm
   • Create a systematic framework with formally defined interfaces
   • Support dynamic assembly of software components
   • The framework must be open and tailored to the needs of scientific PSEs
2. Exploit multi-level abstractions and complex properties of science
   • Recognize the pervasive multi-level structure in science and engineering objects
   • Allow the addition of more detail and precision at all levels
   • Incorporate properties like smoothness of functions, complexity of images, shapes of objects, performance of methods (precisely defined and measured)
3. Reuse of legacy scientific software
   • Encapsulation is possible for reasonably designed software provided some key parts of legacy environment persist, e.g. language compiler for new hardware, compatible hardware
   • Recognize that there can be serious costs to using legacy software
4. Create test beds for components and combinations
   • The plug-and-play paradigm gives a good start
5. Create certain important difficult components
   • Geometric modeling, scientific visualization, symbolic mathematics, scientific databases, general optimization, object property measurement
6. Create knowledge bases for solvers and problems
   • Provide automation of (or help for) construction of the PSE, dynamic selection of solvers, selection of hardware, suitability of output, management of long term computations

The growth of computational power and networks suggests that computational science will shift from the current single physical model design to the design of a whole physical system with a large number of models interacting with each other through geometric and physical interfaces. For example, the design of an engine requires that the different domain-specific analyses involved interact in order to find the final solution. We refer to these multi-component based physical systems as multidisciplinary applications (MAs). 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 and software tools for what
George Cybenko successfully characterized as “large-scope” simulations, in order to manage the complexity and harvest the power of the expected high-performance computing and communications (HPCC) resources. A new challenge is to identify the framework for the numerical simulation of multidisciplinary applications and to develop the enabling theories and technologies needed to support and realize this framework in specific applications. We refer to this framework as a multidisciplinary PSE (MPSE) and assume that its elements are discipline-specific PSEs. The MPSE design objective is to allow the “natural” specification of multidisciplinary applications and their simulation with interacting PSEs through mathematical and software interfaces across networks of heterogeneous computational resources.

Enabled by HPCC advances, it is predicted that the future computational paradigm will be network-based where vital pieces of software and information used by a computing process are spread across the network and are identified and linked together only at run time. This paradigm is in contrast to the current software usage model where one purchases a copy (or copies) of a general-purpose, monolithic software package for use on local hosts, possibly distributed on a collection of local hosts. With network-accessible software repositories and networked computing (NC), the view of software changes from a product to a service. A network-accessible repository provides access to up-to-date copies of software components on an as-needed basis, so called “disposable software”. With networked computing, the software developer provides a computing service to interested parties over the network. The disposable software and networked computing models do not apply to all computing services; basic operating system and network access software, as well as low-level math routines that are tuned for the particular machine architecture, will be permanently resident on the user machine. One advantage of the networked computing model is that as the software provider improves upon software, there is no need to release new versions and upgrades. The user simply sees an improved service. The analogy could be to the phone system - changes in the software of the local switch are completely transparent to the user, save for the availability of additional or enhanced functionality. Similarly, the service provider can upgrade the hardware without affecting the user.

We envision that the network-based paradigm of software usage will eventually become fully automated and effectively transparent to the user. Achieving this will require research in 1) software selection: how someone specifies problems, extracts content information, builds knowledge bases, infers answers, and applies collaborative reasoning to identify software resources to support problem solving are crucial to any NC development, 2) system software: in order for the networked-computing paradigm to be realized, a) both low-level and high-level library interfaces must be standardized, b) network-based software delivery systems must be able to transparently resolve issues related to local and network computational resources, and configurations together with a series of issues related to software licensing and accounting, c) the architectures of network servers and clients, GUIs of clients, client-server communication protocols, security and software reliability must be addressed in the context of scientific software, d) implementation technologies such as WWW, CGI, Java, client-server computing, Java RMI (Remote Method Invocation), Joe (Java’s CORBA compliant ORB (object request broker)), Inferno, Oblets, Legion and Aglets must be evaluated, 3) software testing: the “network” is ideal for linking test suites in various scientific domains forming national repositories and making them available in the form of servers; reliable testing requires standardization of testing data and automatic ways to collect and analyze the results of this process, 4) scientific software servers: the effectiveness of any computing paradigm depends in great part on the availability of powerful problem solving capabilities for application domains (in the jargon of telematics, this is referred as content).

The 28 papers in this book are grouped into four parts and the fifth part is an extensive bibliography. Part I is "Problem Solving Environments: The Enabling Technology for Computational Science" which presents eight papers that define problem solving environments and discusses where they came from and where they are going. The key paper here is the full report from the 1989 workshop that crystalized the PSE concept. Part II is "Domain Specific PSEs: Characteristics for Computational Science" which presents seven papers describing PSEs in a variety of computational science areas. From these, one can see the range of facilities and capabilities that PSEs should have even though none of these yet possesses all the characteristics envisaged for PSEs. Part III is "Frameworks, Middleware and Software: Delivering the Problem Solving Power" which presents seven papers describing aspects of the design and implementation of PSEs. The focus is on how to locate and organize the software and to utilize the diverse computing resources, especially in the highly connected networks of the future. Part IV is "Steering, Generation and Validation:"
Tools for Building and Using PSEs" which presents six papers discussing various facilities needed for PSEs that are somewhat domain specific and yet where some version of these facilities are needed for almost every computational science PSE. Finally, Part V is "Bibliography" which contain XXX reference items about PSEs. It is an extension of the bibliography created in 1990 as a follow-on to the workshop that defined the PSE concept.

An examination of the topics and sources of these papers illustrates the extremely broad and interdisciplinary nature of the creation and application of PSEs. The authors represent academia, government laboratories and industry, they come from eight distinct disciplines (chemical engineering, computer science, ecology, electrical engineering, mathematics, mechanical engineering, psychology and wood sciences) and eight countries (Australia, France, Greece, Italy, Japan, the Netherlands, the United Kingdom, and the United States). This breadth and diversity extends into the computer science aspects of PSEs. These papers involve very substantial amounts of ten specialties (artificial intelligence, computer-human interaction, control, data mining, graphics, languages (design and implementation), networking, numerical analysis, performance evaluation, and symbolic computing) and, of course, the whole topic is one of software engineering and architecture.

Elias Houstis
S. Gallopoulos
J. R. Rice
R. Brambley