Features of Problem Solving Environments for Computational Science

Clifford A. Shaffer, Layne T. Watson, Dennis G. Kafura, and Naren Ramakrishnan
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061
{shaffer,ltw,kafura,naren}@cs.vt.edu

Keywords: Problem Solving Environments, Computational Science, Components.

ABSTRACT

We describe some persistent software infrastructure problems encountered by scientists and engineers who work in application domains requiring extensive computer simulation and modeling. These problems may be mitigated by use of a Problem Solving Environment (PSE), but not all of them are currently being addressed by the PSE research community. We then discuss an approach to designing a toolkit for building PSEs. We argue that PSEs can best be implemented using a component-based approach. We present Sieve/Symphony, our initial efforts at creating a component-based collaborative framework for building Problem Solving Environments.

INTRODUCTION

Many scientific and engineering research groups depend on simulation and modeling as the core of their research effort. There exist research groups in diverse disciplines such as aircraft design, materials science, biological modeling, hydrology, wireless communications systems design, and manufacturing processes for wood-based composites, to name only a few, all with roughly the same operating paradigm. This operating paradigm is to design and implement computer models and simulations of complex physical phenomenon, from which are inferred new discoveries about the real-world process being modeled, or to create new materials and products. While the form and application of the models may vary in significant ways, the approach and problems of these researchers as it relates to software development and infrastructural needs are surprisingly similar.

We describe a complex of problems that appear to be universal within academic research labs conducting this sort of software model-based research. Many researchers are now engaged in developing Problem Solving Environments (PSEs) whose purpose is to aide research in computational science (see for example Akers *et al.* 1997; Allen *et al.* 1999; Catlin *et al.* 1994). Our goal is to explicitly list the problems being encountered by the domain scientists and engineers, not all of which are being addressed by the current PSE efforts. We argue that these persistent problems should be guiding PSE research efforts.

The second purpose of this paper is to describe our architecture for a PSE-building environment. We argue that PSEs can best be implemented using a component-based approach. We present Sieve/Symphony, our initial efforts at creating a component-based collaborative framework for building Problem Solving Environments.

THE PROBLEM IN COMPUTATIONAL SCIENCE

For many scientists and engineers today, the most annoying computing challenge is not creating new high-performance simulations or visualizations. Often the scientists feel competent to develop such software, and funding to support model development is widely available. Rather, many scientists and engineers are expressing frustration that their software and computing resources are a heterogeneous mix of incompatible simulations and visualizations, often spread across differing computer hardware. The specialized software that drives a given lab's research is typically incompatible with that of potential collaborators. Thus, researchers today generally do not make the most of their existing software and comput-

ing resources. Nor does their computing environment yet support on-line, real-time collaboration between researchers seeking to do multidisciplinary work.

The researchers typically voice the following complaints.

1. It is difficult to integrate software from multiple disciplines developed by a diverse group of people on multiple platforms located at widespread locations.
2. It is difficult to share software between potential collaborators in a multidisciplinary effort — difficult even for a team to continue using research software once the author has left the group.
3. Current tools for synchronous collaboration are inadequate.

These issues are of concern throughout a wide research community, as evidenced by numerous NSF workshops and conferences on topics such as Problem Solving Environments, Workflow, and Process Management for scientific and engineering environments. The field of Computer Supported Cooperative Work also has much to offer in solving the communications problems involved in multidisciplinary efforts.

Integrating codes from different disciplines raises both pragmatic and conceptual issues. A pragmatic concern is how best to support the interoperability of independently-conceived programs residing on diverse, geographically distributed computing platforms. Another pragmatic concern is that large, complicated codes now exist that cannot simply be discarded and rewritten for a new environment. However, interoperability is best achieved by adhering to common protocols of data interchange and using clearly identified interfaces. The notions of interfaces and protocols lead directly to the domains of object-oriented software and distributed computing. The key pragmatic issue becomes how to unify legacy codes, tied to specific machine architectures, into an effective whole. The key conceptual issue is how to foster coordinated problem solving activities among multiple experts in different technical domains, and leverage existing codes and computer hardware resources connected by the Internet.

PROBLEM SOLVING ENVIRONMENTS

A Problem Solving Environment (PSE) provides an integrated set of high-level facilities that support users engaged in solving problems from a proscribed domain (Gallopoulos *et al.* 1994). PSEs allow users to define and modify problems, choose solution strategies, interact with and manage the appropriate hardware and software resources, visualize and analyze results, and record and coordinate problem solving tasks.

Based on experiences with the various disciplines listed above, the following is a list of particular issues that should be addressed by a PSE for any Computational Science application.

Internet Accessibility to Legacy Codes The initial reason why a computational scientist or engineer approaches our research group is that they would like to make their legacy modeling code Web-accessible. We typically make legacy code Web-accessible by creating a Java applet that allows the user to fill in a form. The contents of this form are passed to a server on the host computer that stores the legacy code. The server, typically by means of a Perl script, invokes the legacy code with the parameters and input files defined by the user's form. WBCSim (Goel *et al.* 1999) is a typical example of this, though there are many other similar efforts.

Visualization Users of these models typically wish to visualize the output, rather than simply analyze the numbers and text produced by the program. Sometimes the visualization may be generated by a generic tool, but more often an ad hoc visualization tool has been produced along with the modeling code. Regardless, the researcher would like to integrate the visualization process with invocation of the model.

Experiment Management The focus of the research can often be cast as an attempt to solve an optimization problem. A given run of the model is typically an evaluation at a single point in a multidimensional space. In essence, the goal is to supply to the model that vector of parameters that yields the best result under an objective metric. It is not unusual for members of the research team to spend considerable time in the following loop:

- run the model using a certain parameter vector;
- observe the results;
- generate a new parameter vector based on judgment and past history;
- repeat until exhaustion sets in.

Under this operating procedure, the user would like to have the results of the simulation runs be stored automatically in some systematic way that permits recovery of previous runs along with the parameters that initiated the run. Ideally, a mechanism for anno-

tating the results, and a method for searching based on inputs, results, or annotations, would be provided.

Multidisciplinary Support An eventual goal of PSE research is to support the ability of researchers to combine together to form larger, multidisciplinary teams. In practice, this means that the models from the various disciplines involved should be combinable in some way. Perhaps this would be done by linking individual PSEs for the disciplines, or perhaps the various models would operate within the same PSE environment.

Collaboration Support Researchers would like to work together, when initiating/steering the computation or when analyzing the results. While the ability to save and restore prior results can be used to provide asynchronous collaboration, ideally a PSE would allow multiple users at multiple workstations to work together in the PSE at the same time.

Optimization We have noted these research efforts are often cast in the form of an optimization problem. Thus, the process can often be improved by applying automated optimization techniques, rather than have someone manually try a large number of parameter sets. In some disciplines, this is well known and optimizers are an integrated part of the model. But many other disciplines do not typically use optimization techniques. A PSE would ideally allow various models to be combined with various automated optimization techniques. (See Czyzyk *et al.* 1997 for an example of optimization over the WWW.)

High Performance Computing Often, simulations used by computational scientists require access to significant computing resources, such as a parallel supercomputer or an "information grid" of computing resources. In such cases, the PSE should integrate a computing resource management subsystem such as Globus (Foster and Kesselman 1997) or Legion (Grimshaw *et al.* 1997).

Usage Documentation An aspect of providing improved interfaces for simulation codes is implicit and explicit documentation for use of the code, specifically with respect to parameters and other inputs. The interface could provide advice on reasonable interactions of parameters, or which submodels to use in particular circumstances. At the PSE creation level, PSE-building tools could provide a convenient mechanism for adding and accessing such documentation. Documenting is in part a matter of discipline for the developers. Conceivably, PSE implementation tools could enforce good documenting discipline.

Preservation of Expert Knowledge Just like books in libraries, computer programs codify and preserve expert knowledge about the application domain. A PSE can serve two important roles in this regard. First, by using and preserving legacy code, the expert knowledge embodied in the legacy codes continues to be (indirectly) employed. Second, state-of-the-art codes are often nearly impossible for non-experts to use productively, and by providing advice (via an expert system shell) the PSE can make the legacy codes and knowledge usable by non-experts. For multidisciplinary work this expert advice for non-expert users is indispensable.

Recommender Systems Most existing PSEs assume that the choice of method (algorithm) to solve a given scientific problem is fixed *a priori* (static) and that appropriate code is located, compiled and linked to yield static programs. The user (scientist) still needs to select suitable software for the problem at hand in the presence of practical constraints on accuracy, time and cost. A recommender system for a PSE serves as an intelligent front-end and guides the user from a high level description of the problem through every stage of the solution process, providing recommendations at each step (Ramakrishnan *et al.* 1998). Recommenders will also help scientists and engineers achieve increased levels of interactivity as they work together to solve common problems (Ramakrishnan 1999). Further, they will enable and hence encourage an increased flow of information and knowledge among these scientists, their organizations, and professional communities.

Integration While each feature described in this list is important in its own right, the important aspect of a PSE for computational science research such as we have described would be the synergy that should result from integrating these features into a single system. In particular, a collaborative system that provides Internet-based access (perhaps through a Web browser) to an integrated set of models, optimizers, visualizations, and experimental results database, would be a powerful tool indeed.

COMPONENT FRAMEWORKS AND PSES

Readers familiar with components and distributed internet-based applications will recognize that many goals of the PSE described above are also goals of

other distributed applications. While the details differ, the fundamental goals of integrating various components in an application, and access to a database (in this case the database of experimental runs) are not unique to computational science. While supporting legacy code is often central to computational science applications, this need is by no means novel.

However, the combination of issues embodied in the PSE presents novel problems. These include the fact that individual runs of a simulation can take hours; the extensive use of visualization; the inherently distributed nature of the computation (i.e., certain submodels may need to run on differing systems for reasons related to resource needs, or simply because they are legacy codes written for differing systems); the desire for synchronous collaboration; and the needs of multidisciplinary users, no one of which is an expert in all aspects of the larger system.

Most component technology today is aimed at "visual programming," that is, helping programmers to build programs faster and with fewer bugs through greater code reuse. The motivation is that users will be given better applications, but the component research community is only now considering how components will otherwise affect users. An application programmer using component technology generally develops as though these better programs would operate within the same non-component environments as we have today. This view misses much of the potential benefits of a component-based paradigm. Components could more directly support users, in that the user might be linking components together themselves to create new capabilities. This approach is already being used by some visualization programs such a Khoros (Young *et al.* 1995) and AVS (Upson *et al.* 1989). See also (Gannon *et al.* 1998) for another discussion regarding the use of component frameworks for designing PSEs.

A PSE FRAMEWORK

Our own research efforts have been aimed at developing an environment in which to create PSEs much as described in the section on goals (Isenhour *et al.* 1997; Shah and Kafura 1999). We embody the PSE in a (collaborative) visual workspace, in which the user places various objects. These objects are components that represent individual simulations, optimization tools, visualization tools, etc. These components are linked together by the user to form networks that indicate the flow of data or control. The links between components are often represented by arrows. For example, a component representing an input file on some computer might be linked by an arrow to another component representing a model/optimizer combination. Another arrow links the model/optimizer combination to a visualizer. The intent is that the PSE will cause the input file to be moved to the machine storing the model and optimizer, and the model/optimizer will then be invoked. The output of this process will then be passed to the visualization, (perhaps on another machine) with the results displayed on the user's screen. The fundamental interface design is similar to that of a Modular Visualization Environment or the Khoros image processing system.

Our PSE framework is known as Sieve/Symphony, from the names of the two parts that make up the framework. The implementation is based on JavaBeans (Hamilton 1998). Sieve provides a collaborative workspace within which users may place the components that make up the PSE. Sieve also provides a specific collection of JavaBeans for producing and visualizing data. Symphony is a collection of JavaBeans which serve as surrogates for describing and manipulating remote resources (files and executable codes). Sieve/Symphony provides the foundation for constructing PSEs, as their combination creates a collaborative environment for controlling distributed, legacy resources.

Sieve provides an environment for collaborative component composition that supports the following:

- A Java-based system compatible with standard WWW browsers
- A convenient environment for generating visualizations through linking of data-producing modules with data-visualization modules
- Collaboration between users through a shared workspace, permitting all users to see the same visualizations at the same time
- Support for annotating the common workspace, visible to all users
- A convenient mechanism for linking in new types of components

Sieve presents the user with a large, scrollable workspace onto which data sources, processing modules, and visualization components may be dropped, linked, and edited. Figure 1 shows a Sieve workspace containing a simple data-flow network.

Our design for Sieve allows processing and visual-

Figure 1: Example of a Sieve workspace with dataflow and annotations.

ization modules to be generic, with all data-source–specific details hidden by the source modules. Dataflow modules implement an API which allows data to be viewed by adjacent modules in the network as a two-dimensional table containing objects of any type supported by the Java language. Source modules simply convert raw data into a table representation. Processing modules can manipulate these data and present an altered or extended table. Visualization modules can then produce visual representations of the data in a table. Visualization modules can serve as an interface for data selection, in which case they may also present an altered or extended table to adjacent modules.

Symphony is a collection of JavaBeans designed to permit the representation, composition, and manipulation of remote resources. Each Symphony bean serves as a surrogate for some actual resource. This resource may be physically located on a machine other than the one on which the surrogate bean itself resides. Symphony includes Program beans that represent executable entities on some machine, and several beans for representing sources or destinations of data including a File bean, a StandardInput bean, a StandardOutput bean, and a Socket bean.

Symphony requires that a Symphony server be running on each machine containing remote executable resources. Beans that represent these remote resources communicate with and control those resources through their interaction with the Symphony server. The interaction between the bean and the server is via the Java Remote Method Invocation (RMI) service. The set of Symphony servers and the beans will collaborate to transport files between machines, execute programs, and connect data streams as needed to realize the computation specified in the network of beans. An illustration of the interactions between Symphony beans and the remote Symphony servers is shown in Figure 2.

Figure 2: Interaction of a Symphony bean network with Symphony servers and remote resources.

A bean that serves as a surrogate for a specific resource (i.e., a particular executable program) is created by customizing that bean's properties. A property is a changeable attribute of a bean. For example, the customization of a Program bean allows the user to specify such properties as the *hostname* of the machine where the actual executable program or script resides, the *pathname* of the directory where the program or script can be found on its host, and the *filename* of the executable program within its directory.

Individual customized Symphony beans may be composed to describe complex computations. For example, Figure 2 shows how a set of Symphony beans could be logically composed to describe a computation involving two programs and several files. Directed arcs between the individual beans represent the logical flow of data between the actual resources for which the beans are surrogates. For example, the directed arc from a Program bean to a File bean denotes that the actual file for which the File bean is a surrogate will contain the data produced by the execution of the program for which the Program bean is a surrogate. Another flow not shown in Figure 2

allows data generated by one program on its standard output stream to become the data stream seen by another program on its standard input.

Work is also underway to use the PYTHIA recommender kernel in the context of runtime application composition systems. Specifically, PYTHIA can monitor a computational process, detect state-changes, and make selections of solution components dynamically, thus aiding knowledge-based application composition at runtime. Such a facility is important in many problem domains because: (i) the nature of the problem being solved changes as the computations are being performed, (ii) the underlying computing platform or resource availability is dynamic, or (iii) information about application performance characteristics is acquired during the actual computation rather than before.

CONCLUSIONS

The computational science problems described in this paper are real, serious, and widespread. A PSE as described herein is not a panacea for all the problems faced by computational science researchers. Aside from issues related to constructing PSEs themselves, there will still remain problems of translating incompatible data formats, the common occurrence of poor software engineering practices, and the natural inertia that results in poor or outdated documentation. Nonetheless, there is an opportunity here for component frameworks and distributed Internet-based applications to play an important role in advancing the state of the art in computational science.

REFERENCES

Akers, R.; E. Kant; C.J. Randall; S. Steinberg; R.L. Young. 1997. "SciNapse: a problem-solving environment for partial differential equations," *IEEE Computational Science and Engineering 4*, no. 3: 32–42.

Allen, G.; T. Goodale; and E. Seidel. 1999. "The Cactus computational collaboratory: Enabling technologies for relativistic astrophysics, and a toolkit for solving PDEs by communities in science and engineering," in *Proceedings of the 7th Symposium on the Frontiers of Massively Parallel Computation*, IEEE Computer Society, Los Alamitos, CA: 36–41.

Catlin, A.C.; C. Chui; C. Crabill; E.N. Houstis; S. Markus; J.R. Rice, and S. Weerawana. 1994. "PDELab: an object-oriented framework for building problem solving environments for PDE based applications," in *Proceedings of the 2nd Object-Oriented Numerics Conference*, A. Vermeulen, ed., Rogue Wave Software, Corvallis, OR: 79–92.

Czyzyk, J.; J.H. Owen; and S.J. Wright. 1997. *NEOS: Optimization on the Internet*, Technical Report OTC-97/04, Argonne National Laboratory.

Foster, I. and C. Kesselman. 1997. "Globus: A metacomputing infrastructure toolkit," *Internationl Journal of Supercomputer Applications 11*, no. 2(Summer): 115–128.

Gallopoulos, E.; E.N. Houstis; and J.R. Rice. 1994. "Computer as thinker/doer: Problem-solving environments for computational science," *IEEE Computational Science & Engineering 1*: 11–23.

Gannon, D.; R. Bramley; T. Stuckey; J. Villacis; J. Balasubramanian; E. Akman; F. Breg; S. Diwan; and M. Govindaraju. 1998. "Component archi-

tectures for distributed scientific problem solving," *IEEE Computational Science and Engineering 5*, no. 2: 50–63.

Goel, A.; C. Phanouriou; F. A. Kamke; C. J. Ribbens; C. A. Shaffer; and L. T. Watson. 1999. "WBCSim: A prototype problem solving environment for wood-based composites simulations", *Engineering with Computers 15*, no. 2: 198–210.

Grimshaw, A.S.; W.A. Wulf; and the Legion team. 1997. "The Legion vision of a worldwide virtual computer," *Communications of the ACM*, 1(Jan): 39–45.

Hamilton, G. 1998. *JavaBeans 1.01 Application Programming Interface Specification*, SUN Microsystems Inc., `http://splash.javasoft.com/beans-/docs/beans.101.pdf`.

Isenhour, P.L.; J.B. Begole; W.S. Heagy; and C.A. Shaffer. 1997. "Sieve: A Java-based collaborative visualization environment," in *Late Breaking Hot Topics Proceedings, IEEE Visualization'97*, Phoenix, AZ, October: 13–16.

Ramakrishnan, N.; E.N. Houstis; and J.R. Rice. 19998. "Recommender Systems for Problem Solving Environments," Technical Report WS-98-08 (Working Notes of *The AAAI-98 Workshop on Recommender Systems*, H. Kautz, ed., AAAI/MIT Press: 91–95.

Ramakrishnan, N. 1999. "Experiences with an Algorithm Recommender System," Working Notes of *The CHI'99 Workshop on Interacting with Recommender Systems*, P. Baudisch, ed., ACM SIGHI Press.

Shah, A. and D. Kafura. 1999. "Symphony: A Java-based composition and manipulation framework for problem solving environments" in *Proceedings of International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE'99)*, May 17–18, Los Angeles, CA.

Stefik, M.; D. G. Bobrow; G. Foster; S. Lanning; and D. Tatar. 1987. "WYSIWIS revised: Early experiences with multiuser interfaces," *ACM Transactions on Office Information Systems*, (April): 147–167.

Upson, C.; T.A. Faulhaber, Jr.; D. Kamins; D. Laidlaw; D. Schlegel; J Vroom; R. Gurwitz; and A. van Dam. 1989. "The Application Visualization System: A computational environment for scientific visualization," *IEEE Computer Graphics and Applications 9*, no. 4(July): 30–42.

Young, M.; D. Argiro; and J. Worley. 1995. "An object oriented visual programming language toolkit," *Computer Graphics 29*, no. 2(May): 25–28.