Component Frameworks for Problem Solving Environments in Computational Science

Clifford A. Shaffer, Layne T. Watson, and Dennis G. Kafura
Department of Computer Science
Virginia Tech
Blacksburg, VA 24061
{shaffer,watson,kafura}@cs.vt.edu

ABSTRACT

We describe some persistant problems in software infrastructure encountered by scientists and engineers working in application domains requiring extensive computer simulation and modeling. These problems may be mitigated by use of a Problem Solving Environment. We argue that such a Problem Solving Environment can best be implemented using a component-based approach.

KEYWORDS: Problem Solving Environments, Computational Science, Components.

Many scientific and engineering research groups depend on simulation and modeling as the core of their research effort. We are familiar with 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, 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 surprising similar.

The purpose of this paper is to describe in further detail the complex of problems that appear to be universal within academic research labs conducting this sort of software model-based research. I then go on to define the needs of these researchers in terms of a Problem Solving Environment. I conclude by describing how a workspace environment implemented within a component framework may some day provide an effective solution.

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 do not make the most of their existing software and computing 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. Pragmatically, the issue 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 adherence to common protocols of data interchange and the use of clearly identified interfaces. The notions of interfaces and protocols lead directly to the domains of object-oriented software and distributed computing. Thus, a key issue is how to unify legacy codes, tied to specific machine architectures, into an effective whole. Conceptually, the key 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.[1] 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.

Intranet Accessibility to Legacy Codes The initial reason why a scientist or engineer in one of these computational science domains approaches our research group is that they would like to make their legacy modeling code web-accessible. In its simplest form, this typically means creating a Java applet as a front end for filling in a form. The contents of this form are passed to a server on the host storing the legacy code which, typically by means of a perl script, invokes the legacy code with the appropriate parameters and input files. WBCSim [7, 3] is a typical example of this, though many other similar efforts are now available.

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 multi-dimensional 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: Feed a parameter vector to the model; observe the results; generate a new parameter vector based on 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 annotating 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, either 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 be working together in the PSE at the same time.

Optimization As noted above, these research efforts are often cast in the form of an optimization problem. As such, the research effort can often be improved by applying automated optimization techniques. 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.

High Performance Computing Often, simulations used by computational scientists require access to significant computing resources, such has a parallel supercomputer or an "information grid" of computing resources. In such cases, the PSE should integrate a computing resource mangagement subsystem such as Globus or Legion.

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 simulation 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 programmer discipline. Con-

ceivably, 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 nigh impossible for nonexperts to use productively, and by providing advice (via an expert system shell) the PSE can make the legacy codes and knowledge usable by nonexperts. For multidisciplinary work this expert advice for nonexpert users is indispensable.

Integration While each feature described in this list is important in its own right, the important aspect of a PSE for computational 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 of the goals of the PSE described above are also goals of other distributed applications. While the details differ, the fundamental goals of integrating various sub-sections of 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 described above 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 helping programmers to build better programs. The motivation is that users will be given better applications, but developers are only now considering how components will otherwise affect users. In other words, the programmer 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 support users as much as programmer.

Part of our own research efforts have been aimed at developing an environment in which to create PSEs much as described above.[4, 5] We embody the PSE in a (collaborative) workspace, in which the user places various objects. The fundamental interface objects are components that represent individual simulations, optimizer tools, visualization tools, etc. These components are linked together by the user to form networks that indicate the flow of data or control. 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 fed to the visualization, with the results displayed on the user's screen. The fundamental interface design is similar to that of a Modular Visualization Environment[6, 2] or the Khoros[8] image processing system.

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 themselves, there will still remain problems of translating incompatable data formats, the common occurance 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 computational science.

REFERENCES

1. E. Gallopoulos, E. Houstis, J.R. Rice, Problem-solving environments for computational science, *IEEE Computational Science & Engineering 1*, 1994, 11–23.

2. D.S. Dyer, A Dataflow Toolkit for Visualization, *IEEE Computer Graphics and Applications 10*, 4(July, 1990), 60–69.

3. A. Goel, C. Phanouriou, F. A. Kamke, C. J. Ribbens, C. A. Shaffer, and L. T. Watson, "WBC-Sim: A prototype problem solving environment for wood-based composites simulations", to appear in *Engineering with Computers.*

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

5. A. Shah and D. Kafura, 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, 1999, Los Angeles, CA.

6. C. Upson, T.A. Faulhaber, Jr., D. Kamins, D. Laidlaw, D. Schlegel, J Vroom, R. Gurwitz, and A. van Dam, The Application Visualization System: A Computational Environment for Scientific Visualization, *IEEE Computer Graphics and Applications 9*, 4(July, 1989), 30–42.

7. WBCsim, URL: `wbc.forprod.vt.edu/pse/`.

8. M. Young, D. Argiro and J. Worley, An Object Oriented Visual Programming Language Toolkit, *Computer Graphics 29*, 2(May 1995), 25–28.