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
Phone: 540-231-6931, FAX: 540-231-6075
{shaffer,ltw,kafura,naren}@cs.vt.edu

September 20, 1999

Abstract

We describe some persistant 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 include a brief survey of ongoing PSE research efforts. 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. We conclude with a discussion of the lessons learned from our initial experience with the Sieve/Symphony approach.

Keywords: Problem Solving Environments, Computational Science, Components.

1 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.

The first purpose of this paper is to describe the 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. 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. We conclude with a discussion of the lessons learned from our initial experience with the Sieve/ Symphony approach.

2 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 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.

3 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 [19]. 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 scientist or engineer in a computational science domain 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 [46, 21] is a typical example of this, though many other similar efforts are now available (see Section 4).

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 judgement 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 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, 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.

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 [18] or Legion [23].

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 discipline on the part of 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 [35]. Recommenders will also help scientists and engineers achieve increased levels of interactivity as they work together to solve common problems [36]. 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.

4 Prior Research on PSEs

PSE research includes (1) developing problem-specific PSEs and (2) developing general tools for building PSEs. Issues such as developing a general architecture for PSEs; leveraging the Web; supporting distributed, collaborative problem solving; and providing software infrastructure ("middleware") are also being addressed. Most of the items listed in the previous section as desirable features of a PSE for computational science are incorporated in one or more of the projects described in this section.

One problem domain where PSEs are common is the numerical solution of partial differential equations (PDEs). An early example is ELLPACK [7], a portable Fortran 77 system for solving two and three dimensional elliptic PDEs. Its strengths include a high-level language which allows users to define problems and solution strategies in a natural way (with little coding), and a relatively open architecture which allows expert users to contribute new problem solving modules. ELLPACK's descendents include Interactive ELLPACK [16], which adds a graphical user interface to support better user interaction, and Parallel ELLPACK (PELLPACK) [27], which includes a more sophisticated and portable user interface, incorporates a wider array of solvers, and can take advantage of multiprocessing.

PELLPACK also includes an expert or "recommender" component named PYTHIA [30, 47]. PYTHIA also interfaces with other application specific PSEs to recommend software for specific categories of problems such as partial differential equations and numerical quadrature [37, 28]. In addition to selecting software for these domains, it interfaces with the GAMS mathematical software ontology (`http://gams.nist.gov`) to direct the user to an appropriate location from where the software can be retrieved. In other words, software recommendation is *complete* for these classes of problems. Furthermore, the PYTHIA kernel supports the rapid prototyping of recommender systems. This kernel abstracts the architecture of a recommender system as a layered system with clearly defined subsystems for problem formulation, knowledge acquisition, performance evaluation and modeling, and knowledge discovery.

Another system which provides a high level, problem-oriented environment for PDE-solving is SciNapse [1], a code-generation system that transforms high-level descriptions of PDE problems into customized C or Fortran code, in an effort to eliminate the need for programming by hand. Other PSEs in the PDE problem domain include DEQSOL [44], PDEase2D [48], and PDESOL [40].

PSEs are being built for a number of other scientific domains as well. For example, Parker et al. [38] describe SCIRun, a PSE that allows users to interactively compose, execute, and control a large-scale computer simulation by visually "steering" a dataflow network model. SCIRun supports parallel computing and output visualization well, but has no mechanisms for experiment managing and archiving, optimization, real-time collaboration, or modifying the simulation models themselves. Bramley et al. [8, 20] have developed Linear System Analyzer, a component-based PSE, for manipulating and solving large-scale sparse linear systems of equations. Dabdub et al. [13] have built a PSE for modeling air pollution in urban areas. The WISE environment [31] lets researchers link models of ecosystems from various subdisciplines. An object oriented environment for optimization is DAKOTA [17], which provides support for legacy code, high level component composition, and parallel computing. Lacking are integrated visualization, collaboration support, experiment management and archiving, and support for modifying the underlying simulation models.

The Information Power Grid [5] being envisioned by NASA and the national laboratories is a general, all-encompassing PSE. While some of the requisite technologies are in place (e.g., Globus [18] for distributed resource management, and PETSc [3, 24] for a scientific software library), it is unclear how the remaining components can be built and integrated. At this time, IPG is a vision rather than a working prototype. The law of conservation at work here seems to be that the power and level of integration of a PSE is directly proportional to the specificity of the problems being addressed by the PSE. The CACTUS [2] system for the relativistic Einstein equations for astrophysics supports distributed computing, visualization, collaboration, experiment management, and model development. However, to adapt CACTUS to a different problem class is likely to be rather difficult, as the component tools are tailored to solving astrophysics equations.

An important goal of PSE researchers is to define a

generic architecture for PSEs and to develop middleware (typically object-oriented) to facilitate the construction and tailoring of problem-specific PSEs [19]. This emphasis, along with work in Web-based, distributed, and collaborative PSEs, characterizes much of the current research in PSEs. An example is PDELab [10], a multilayered, object-oriented framework for creating high-level PSEs. PDELab supports PDESpec, a PDE specification language that allows users to specify a PDE problem in terms of PDE objects and the relationships and interactions between them. Parallel Application WorkSpace (PAWS) [34] is a CORBA-based, object-oriented server for connecting parallel programs and objects. Other researchers investigating object-oriented frameworks for PSE-building include Gannon et al. [20], Balay et al. [4], and Long and Van Straalen [32].

With the rise of the Web, PSEs are now beginning to support distributed problem solving and collaboration. Regli [39] describes Internet-enabled computer-aided design systems for engineering applications. Net PELLPACK [33], PELLPACK's Web-based counterpart, lets users solve PDE problems via Java applets. Other Web-based PSEs include NetSolve [9] and NEOS [12]. Current PSE-related research projects that emphasize distributed collaboration include LabSpace [14], the Intelligent Synthesis Environment (ISE) [22], Habanero [11], Tango [6], Symphony [41], and Sieve [29].

5 Component Frameworks and PSEs

Readers familiar with components and distributed internet-based applications will recognize that many of the goals of the PSE described in Section 3 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 [49] and AVS [45].

6 A Framework for Implementing PSEs

Our own research efforts have been aimed at developing an environment in which to create PSEs much as described in Section 3 [29, 41]. 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 [45, 15] or the Khoros [49] image processing system.

Our PSE framework is known as Sieve/Sympony, from the names of the two parts that make up the framework. The implementation is based on JavaBeans [26]. 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.

6.1 Sieve

Sieve [42] 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 visualization modules to be generic, with all data-source–specific details hidden by the source modules. Data-flow 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.

The resulting data-flow network is fully interactive, using an event mechanism to notify interested modules of changes to the data or to the configuration of

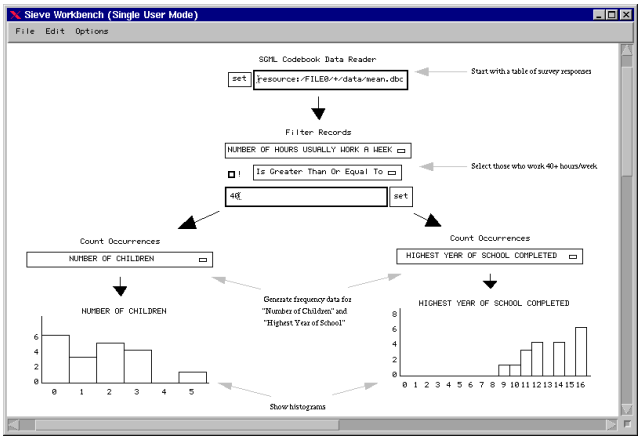

Figure 1: Example of a Sieve workspace with dataflow and annotations.

the network. These modules can then retrieve new or modified data from their source. Users modify the network directly on the workspace, with changes to both the structure of the network and the modules themselves reflected to all collaborators as quickly as processing power and network speed permit.

Sieve supports flexible collaborations and provides real-time information about participants' actions and locations in the workspace. A range of collaboration styles are supported by providing location relaxed WYSIWIS (What You See Is What I See) [43] where collaborators may view and manipulate the same or different parts of the shared workspace. Hence, while changes made to the workspace are propagated to all remote collaborators, all collaborators need not be working in the same part of the workspace simultaneously.

Sieve provides workspace awareness (continuous knowledge of remote participants' interactions and locations [25]) through two interface elements: telepointers and a multiuser overview of the workspace. A telepointer represents a remote user's mouse pointer position and thus provides location awareness. Collaborators can also use telepointers to gesture at items on the workspace to augment communication. To provide additional workspace awareness information, Sieve uses a multiuser overview, or *radar view* [25] of the workspace. The radar view displays a rectangle for each user representing that user's viewport into the workspace, providing additional location information. Remote users' mouse positions are also indicated on the radar view.

The state of each Sieve session is stored on the server, allowing "late joiners" to be brought up to date. This persistence mechanism also allows Sieve

to be used for asynchronous collaboration. Collaborators may work at different times, leaving their modifications for coworkers to manipulate later.

Sieve also provides a set of whiteboard-style tools for annotating the workspace. Lines, arrows, text, images, and even arbitrary Java objects can share the workspace with data-flow networks. As with data-flow components, whiteboard components are shared across all collaborating sessions. These tools enhance Sieve's support for asynchronous collaboration, since one collaborator can leave notes on the workspace for later review by other collaborators.

The JavaBeans concept of "bound properties" (binding attributes of one object to compatible attributes of another object) is extended by Sieve to support collaboration. JavaBeans provides mechanisms for detecting and propagating changes to bound properties of local objects. In Sieve, we extend this to support propagation of these changes to each replica of an object in the collaborating sessions. This allows many kinds of components to be written without knowledge that they will be used collaboratively.

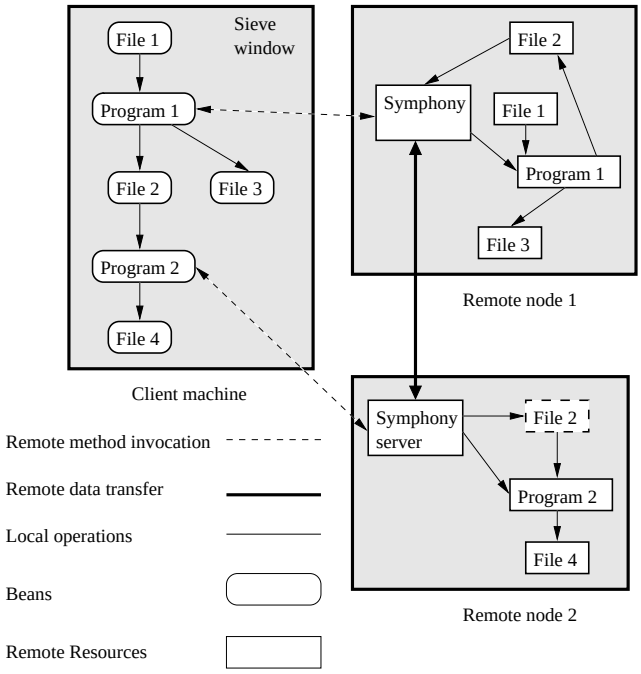

Figure 2: Interaction of a Symphony bean network with Symphony servers and remote resources.

6.2 Symphony

Symphony is a collection of JavaBeans designed to permit the representation, composition, and manipulation of remote resources. Each Symphony bean serves as a surrogate or representative 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.

While the Symphony beans denote a resource type (e.g., the Program bean denotes an executable resource which can be started), 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. Similarly, the properties of other Symphony beans can be customized as appropriate to their type.

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 stan-

dard output stream to become the data stream seen by another program on its standard input.

The structure created by composing Symphony beans is subject to two different verifications. First, when the user attempts to create a connection between two beans, the affected beans will verify that the connection is structurally valid, that is, the implied relationship between the two real resources is meaningful. For example, a Socket bean cannot be connected to a File bean because a file cannot respond to socket operations. The beans will not permit invalid connections to be made. Second, when asked by the user, each bean can verify the existence of the actual resource to which the bean's customization refers. For example, a Program bean will check that the hostname refers to a reachable machine, and that the pathname and filename refer to an existing executable program or script. Any problems encountered are displayed to the user.

Each Symphony bean is a surrogate in that operations performed on the bean effect a corresponding operation on the actual resource. For example, a Program bean may represent a simulation, a visualization system, or an agent. Actions applied to the bean are made to occur on the resource itself. Thus, asking the simulation bean to start causes the corresponding simulation program to start. Actions are applied to a bean directly by the user or by other beans. For example, in Figure 2, Program 1 could be started by the user. The Program 1 bean would automatically inform the File 2 bean when execution has completed. This notification would be conveyed by the File 2 bean to the Program 2 bean, which would then initiate its own execution without direct user intervention.

6.3 Recommender Systems for Runtime Application Composition

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. While traditional recommenders are designed *off-line* (by organizing a battery of benchmark problems and algorithm executions, and subsequently mining it to obtain high-level recommendation rules), the design of a runtime recommender system is difficult because such a database is not readily available and needs to be 'captured' on the fly. Thus, a runtime recommender interacts dynamically with its environment and learns through interactions with its environment. Moreover, a runtime recommender system will provide a useful conceptual paradigm for the effective design and utilization of metasystems such as Legion and Globus.

7 Lessons Learned

Our experience in building and using the Sieve/-Symphony system leads us to believe that the basic concept of a component-based approach to building PSEs is correct. However, we have learned that significant changes in our specific design and implementation are required. The notion of a composition-oriented environment in which individual components can be customized and linked to create ensembles of programs and data still appears in retrospect to be a useful way to proceed. Also in retrospect, however, we find two key areas that must be changed: The relationship between flows of data and control; and the fundamental interface for novice users.

The first required change comes from the recognition that the current design entangles unnecesarily the issues of the user interface, the logical relationships among the files and programs that comprise the metaprogram, and the execution mechanics of how the metaprogram's files are transferred and how its programs are executed. The entanglement of these issues can be seen in summarizing a number of the key responsibilities assigned to each bean. Each bean directly projects its visual representation in the user interface and directly responds to low-level events in the user interface such as mouse clicks. Each bean also generates and reacts to internally generated events that correspond to changes in the logical structure or execution state of the metaprogram. These events represent new connections being made between beans during composition and the completion of run-time steps during execution. Finally, each bean directly interacts with fixed, known servers (such as the Symphony server or the FTP server) to enforce the execution behavior expected of the bean. All of the design and implementation decisions related to these three issues are present in each bean.

Symphony's structure should be redesigned to

achieve better separation among the design decisions related to the user interface, the logical relations among the beans, and the execution machinery. The redesign would make each Symphony bean primarily responsible for maintaining the logical structure of the metaprogram and lessen or eliminate its responsibilities in the other two areas. This can be accomplished as follows.

First, each Symphony bean could implement a programatic interface through which an external component can query its state, request actions, and be notified of changes in the state of the bean. The bean is not concerned with drawing its representation or interpreting low-level user interface events. In effect, the Symphony bean becomes the "model" element in the model-view-controller (MVC) design pattern. The view and control elements would be implemented separately and interact with a Symphony bean only through the defined programatic interface.

Second, Symphony beans would be insulated from the details of the execution mechanism by a second programmatic interface. Through this interface a Symphony bean could request that files be transferred and programs be executed. However, the individual beans would have only limited knowledge of how these actions were carried out.

The separation of the Symphony beans from the execution machinery allows Symphony to become more useful to the distributed high performance computing community that has invested significant resources in the development of metacomputing systems. Examples of these metacomputing systems are Globus [18] and Legion [23]. Metacomputing systems are increasingly used in large-scale experiments involving high-performance computing and communications because they provide to the user a single computing "machine" that is easier to use than the actual distributed systems out of which it is constructed. The metacomputing system transparently provides scheduling and security services to its users in the same way that these services are provided to users of desktop computing. The underlying details of the movement of data and execution of code is hidden from the user whenever possible and desired. The Symphony redesign would allow a layer of software to be written that implements the interface between the Symphony beans and the execution machinery. The distinction between the metacomputing system and Symphony is that the metacomputing system is concerned with scheduling and executing a single program while Symphony is concerned with the orchestrated execution of a collection of such programs. Symphony plays the role of the "metaprogram" for the virtual "metacomputer" created by the metacomputing system. The proposed redesign is beneficial to Symphony, because it can be used in more execution environments, and also beneficial to the metacomputing community, because their systems can now be programmed in a more extensive way.

The second change in Symphony is a recognition that the linking of a large number of components in the current system is untenable because it provides only a single, "flat" space. A metaprogram with hundreds of beans is clearly unmanageable from a user's perspective. A more richly structured interface must be created. The goal is to provide an aggregation structure in which a number of beans can be aggregated into a single, logical entity. This entity can itself be composed with other individual beans or other aggregates. Beyond its immediate advantage as an organizing device, aggregation also permits the customization of bean properties to be done more easily, as a customization applied to an aggregate can be applied automatically to each bean within the aggregate. This requires that a model of aggregate customization be developed. This model would allow composers to define how the properties of individual beans are related to the properties of the aggregate in which they are contained. Questions also arise about which individual beans are connected when one aggregate is connected to another aggregate.

The separation of the beans' responsibilities and the ability to support aggregation can be implemented using a common means. The Enterprise Java Beans framework defines a BeanContext that acts as a container for individual beans and other BeanContexts. The composable nature of the BeanContext creates the basic structure needed to implement aggregation. In addition, the container can play the role of an intermediary between the individual beans within the container and the world outside of the container. The beans can query the container to discover services that are available to them. In our case, a bean would query the container to discover an execution entity that the bean would access through a well defined, programmatic interface. The actual entity might be different on different executions or even change dynamically during the life of a single execution. In a similar way, entities outside of the container could discover the contained beans which could then be accessed through a well defined programmatic interface.

The separation achieved by these two interfaces increases the flexibility of the PSE infrastructure. Different user interfaces can use the same underlying beans. In fact, the beans might even exist in an envi-

ronment where there is no user interface at all, such as when the collection of beans is controlled by another program (e.g., a metaprogram submitted for execution at a later point in time when needed resources are available). In the same way, the beans could use different execution servers at different times or even different servers at different stages of its execution.

In summary, we believe that the basic concept of Sieve/Symphony is a good one and that its utility can be improved significantly by the redesign outlined above.

8 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

[1] R. Akers, E. Kant, C.J. Randall, S. Steinberg, and R.L. Young, SciNapse: a problem-solving environment for partial differential equations, *IEEE Computational Science and Engineering 4*, 3(1997), 32–42.

[2] G. Allen, T. Goodale, and E. Seidel, 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, 1999, 36–41.

[3] S. Balay, W.D. Gropp, L.C. McInnes, and B.F. Smith, Efficient management of parallelism in object-oriented numerical software libraries, in *Modern Software Tools in Scientific Computing*, E. Arge, A. M. Bruaset, and H. P. Langtangen, Eds., Birkhauser Press, 1997.

[4] S. Balay, W.D. Gropp, L.C. McInnes, and B. Smith, *A microkernel design for component-based parallel numerical software systems*, Technical Report ANL/MCS-P727-0998, Argonne National Laboratory, 1998.

[5] S. Barnard, R. Biswas, S. Sain, R. Van der Wijngaart, M. Yarrow, L. Zechtzer, I. Foster, and O. Larsson, Large-scale distributed computational fluid dynamics on the information power grid using Globus, in *Proceedings of the 7th Symposium on the Frontiers of Massively Parallel Computation*, IEEE Computer Society, Los Alamitos, CA, 1999, 60–67.

[6] L. Beca, G. Cheng, G.C. Fox, T. Jurga, K. Olszewski, M. Podgorny, P. Sokolowski, and K. Walczak, Java enabling collaborative education, health care and computing, *Concurrency Practice and Experience 9*, 1997, 521–534.

[7] R.F. Boisvert and J.R. Rice, *Solving elliptic problems using Ellpack*, Springer-Verlag, New York, 1985.

[8] R. Bramley, D. Gannon, T. Stuckey, J. Villacis, E. Akman, J. Balasubramanian, F. Breg, S. Diwan, and M. Govindaraju, *The linear system analyzer*, Technical Report TR-511, Computer Science Dept, Indiana University, 1998.

[9] H. Casanova and J. Dongarra, NetSolve: a network server for solving computational science problems *International Journal of Supercomputer Applications and High Performance Computing 11*, 1997, 212–223.

[10] A.C. Catlin, C. Chui, C. Crabill, E.N. Houstis, S. Markus, J.R. Rice, and S. Weerawana, 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, 1994, 79–92.

[11] A. Chabert, E. Grossman, L. Jackson, S. Pietrovicz, and C. Seguin, Java object-sharing in Habanero, *Communications of the ACM 41*, 6(June, 1998), 69-76.

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

[13] D. Dabdub and R. Manohar, Performance and portability of an air quality model, *Parallel Computing 23*, 14(1997), 2187–2200.

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

[15] D.S. Dyer, A dataflow toolkit for visualization, *IEEE Computer Graphics and Applications 10*, 4(July, 1990), 60–69.

[16] W.R. Dyksen, and C.J. Ribbens, Interactive ELLPACK: an interactive problem solving environment for elliptic partial differential equations, *ACM Transactions on Mathematical Software 13*, 1987, 113–132.

[17] M.S. Eldred and W.E. Hart, Design and implementation of multilevel parallel optimization on the Intel TeraFLOPS, AIAA Paper AIAA-98-4707, 1998.

[18] I. Foster and C. Kesselman, Globus: A metacomputing infrastructure toolkit, *Internationl Journal of Supercomputer Applications 11*, 2(Summer 1997), 115–128.

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

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

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

[22] D.S. Goldin, S.L. Venneri, and A.K. Noor, Beyond incremental change, *IEEE Computer 31*, 10(1998), 31–39.

[23] A.S. Grimshaw, W.A. Wulf, and the Legion team, The Legion vision of a worldwide virtual computer, *Communications of the ACM*, 1(January 1997), 39–45.

[24] W.D. Gropp and B.F. Smith, Scalable, extensible, and portable numerical libraries, *Proceedings of Scalable Parallel Libraries Conference*, IEEE, Los Alamitos, CA, 1994, 87–93.

[25] C. Gutwin, S. Greenberg, and M. Roseman, A usability study of awareness widgets in a shared workspace groupware system, in *Computer-Supported Cooperative Work* 1996, ACM Press, 258–67.

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

[27] E.N. Houstis, J.R. Rice, S. Weerawarana, A.C. Catlin, P. Papachiou, K.Y. Wang, and M. Gaitatzes, PELLPACK: a problem solving environment for PDE-based applications on multicomputer platforms, *ACM Transactions on Mathematical Software 24*, 1998, 30–73.

[28] E.N. Houstis, V.S. Verykios, A.C. Catlin, N. Ramakrishnan, and J.R. Rice, PYTHIA II: A K/DB System for Recommending/Testing Scientific Sofware, to appear in *ACM Transactions on Mathematical Software*, 1999.

[29] P.L. Isenhour, J.B. 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.

[30] A. Joshi, S. Weerawarana, N. Ramakrishnan, E.N. Houstis, and J.R. Rice, Neuro-fuzzy support for problem-solving environments: a step toward automated solution of PDEs, Special Joint Issue of *IEEE Computer and IEEE Computational Science and Engineering 3*, 1(1996), 44–56.

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

[32] K. Long and B. Van Straalen, PDESolve: an object-oriented PDE analysis environment, in Proceedings of the SIAM Workshop on Object Oriented Methods for Inter-operable Scientific and Engineering Computing, Philadelphia, PA, 1998.

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

[34] S.M. Mniszewski, P.H. Beckman, P.K. Fasel, and W.F. Humphrey, Efficient coupling of parallel applications using PAWS, in *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC7)*, Chicago, IL, 1998.

[35] N. Ramakrishnan, E.N. Houstis, and J.R. Rice, 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, 1998, 91–95.

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

[37] N. Ramakrishnan, J.R. Rice, and E.N. Houstis, GAUSS: An Online Algorithm Recommender System for One-Dimensional Numerical Quadrature, to appear in *ACM Transactions on Mathematical Software*, 1999.

[38] S.G. Parker, D.M. Weinstein, and C.R. Johnson, The SCIRun computational steering software system in *Modern Software Tools in Scientific Computing*, E. Arge, A.M. Bruaset, and H.P. Langtangen (Eds.), Birkhauser Press, 1997, 1–40.

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

[40] W.E. Schiesser, *Computational Mathematics in Engineering and Applied Science: ODEs, DAEs, and PDEs*, CRC Press, Boca Raton, 1994.

[41] 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.

[42] Sieve, URL: `simon.cs.vt.edu/Sieve`.

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

[44] Y. Umetani, M. Tsuji, K. Iwasawa, and H. Hirayama, DEQSOL: A numerical simulation language for vector/parallel processors in *Problem Solving Environments for Scientific Computing*, B. Ford and F. Chatelin (Eds.), Elsevier Science Publishers, North-Holland, 1987, 147–162.

[45] 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.

[46] WBCsim, URL: `wbc.forprod.vt.edu/pse/`.

[47] S. Weerawarana, E.N. Houstis, J.R. Rice, and A. Joshi, PYTHIA: a knowledge based system to select scientific algorithms, *ACM Transactions on Mathematical Software 22*, 1996, 447–468.

[48] C.F. Weggel, Versatile 2-D analysis, *IEEE Spectrum 34*, 8(1997), 92–93.

[49] M. Young, D. Argiro and J. Worley, An object oriented visual programming language toolkit, *Computer Graphics 29*, 2(May 1995), 25–28.