Computational Steering in Problem Solving Environments

David Lancaster and J.S. Reeve

Electronics & Computer Science, University of Southampton, Southampton SO17 1BJ, U.K.
djl@ecs.soton.ac.uk

Abstract. A strong motive to build Problem Solving Environments (PSE's) is the ability to interactively steer computations. We analyse the requirements that steering puts on the architecture of such a PSE and propose a design that does not separate the development and execution parts of the PSE. We describe a prototype implementation of this design based on the standardized software infrastructure of Java Beans and CORBA that enables a high degree of steering.

1 Introduction

The desire to solve ever larger and more complex scientific and engineering problems has been one of the main driving forces in computer development. The improvement in hardware performance has been phenomenal, but the software environment in which solution methods are constructed has also evolved: notably through high level languages and scientific and mathematical libraries. Problem Solving Environments (PSE's) have been proposed [1] as a further stage in this development. A PSE is a software environment that provides support for all stages in both the development and execution of problem solving code. This paper is concerned with PSE's that incorporate, and indeed emphasise, computational steering for manipulating the course of problem solving code as it proceeds. Computational steering implicitly puts requirements of interactive responsiveness on the architecture of a PSE. In the next section we analyse some possible PSE designs in terms of these requirements. We argue that a design that does not separate the development and execution parts of the PSE allows computational steering to fit better into the control flow management and is therefore favored.

2 PSE Architecture

The commonly agreed central requirements of a PSE are to support development and execution and can be designed in two parts separately responsible for each requirement. The development part consists of a sophisticated user interface which guides the user in building a solution to his problem. The output of this development stage is a taskgraph that describes the individual tasks that must be run along with their submission order and data dependencies. Tasks consist of pieces of code and possibly parameters that tune their function. The taskgraph can be described in whatever format is convenient, a modern choice would be XML [2]. The execution part of the design takes the taskgraph and schedules it on whatever platforms are available in such a way as to satisfy all the dependency constraints encoded in the taskgraph.

This design, which we shall call the "separated design", forms the basis of many existing PSE's. It is quite apparent that this separated design limits the ability to interact with the system at run time because the taskgraph becomes immutable once it has been passed to the execution stage.

By contrast, the "combined design" integrates computational steering from the beginning and does not prejudge the kind of steering that may be necessary at run-time. The user may halt the execution, change the set of tasks and their connections, and restart the processes while preserving the state of their data. For example, a visualization module might be attached or a more accurate solver module substituted for a less accurate one. In the language of the separated design: the user is allowed to modify the taskgraph at run time.

To understand the distinction between the two designs consider the management of control flow. In general, the control flow in a PSE is determined by two factors: data flow and steering. The data flow model is intuitive in the PSE context and is made manifest in the way that the components are connected together in the graphical composer of the development part of the PSE. In the separated design, the taskgraph is an abstract representation of this data flow, possibly with hooks for a steering controller. The taskgraph is dispatched by a separate execution system and the controller manages the steering part of the control as another separate system. In the combined design, the control flow is never abstracted away from the original form as determined by the user's selection of modules and connection pattern in the graphical compositor. The components manage their own dispatching; as soon as one component has completed its task and generated its data, control passes to the next component. Computational steering fits in naturally because it is performed at the level of the graphical compositor using the same tools that were used to set up the original problem and no new control structure is needed.

3 Prototype PSE

Software developments for commodity markets over the last few years have provided us with the some basic tools that can be used to construct PSE's. There are strong arguments to use these software layers in designing software for scientific and engineering purposes, notably: they are cheap (often with free implementations), widely available and most importantly they are standardized.

Our prototype PSE uses JavaBeans [3] and CORBA [4]. These are appropriate for a prototype that is intended to be used with medium sized platforms such as clusters [5] small shared memory machines and individual workstations. A more highly featured PSE that was intended to be used with more substantial computing resources would also need software layers that connect to queuing systems and databases.

JavaBeans fulfill the need for a sophisticated user interface that allows graphical programming by connecting computational components. CORBA provides a platform independent way of distributing the computations performed by the PSE so as to take advantage of high performance machines and software implementations.

To illustrate the use of the PSE and to make the following discussion more concrete we have implemented some simple components that perform operations that diagonalize a matrix [6]. These components include one to generate the matrix and components Householder and QL that manipulate the form of the matrix.

The beans representing these components have interfaces described in the BeanInfo class that prescribe how they can be wired together in the builder tool. Besides listing any parameters, this class lists the type and form of the inputs and outputs that the component expects. The JavaBean code is essentially limited to describing these interfaces and the CORBA client code needed to call the computational module residing on a server. For this reason we call these beans "hollow". In order to avoid unnecessary data movement and thereby improve performance, the form of the inputs/outputs are not explicit matrices, but are CORBA references to data objects that contain the matrices.

The CORBA part of the prototype development requires an IDL for the system including the data objects for the matrices. This IDL is too long and detailed to include in a paper but is available on `http://gather.ecs.soton.ac.uk/PSE/Docs`. The CORBA servers have evolved since the beginning of this project and now provide facilities to manage the life-cycle of computational components.

We employ a performance enhancement that makes non-blocking CORBA calls to hide latency following a standard CORBA technique. Consider for example the Householder component which accepts a symmetric matrix and generates a tridiagonal matrix. When this bean starts to run, it first contacts the scheduler which provides an object reference on some remote machine to the computational class that implements the householder transformation. The bean then uses this reference to submit the object reference of the symmetric matrix data class and control then passes back to the bean which then waits. Meanwhile, the Householder object uses the matrix reference to transfer the matrix data to its local address space and then proceeds with the computation, finally creating a new matrix object in which to store the resulting tridiagonal matrix. The object reference to this tridiagonal matrix object is returned to the bean indicating completion of the job.

For the purposes of this work we implemented a scheduler that allows several versions of each component to exist on different machines and selects which one to use on request. This scheduler is dumb in the sense that the algorithm for selection is extremely simple, being either random or based on prior user choices. It does however provide the same kind of interface expected in a more sophisticated version. CORBA initialization employs a nameserver which is used to register servers that are started by hand on whatever remote machines are available. The scheduler contacts the nameserver to obtain all the information about which components are available on which machines.

The control flow information has its origin in the act of wiring JavaBeans together and remains encapsulated in the builder tool. It is never abstracted into a taskgraph. As soon as one component has completed its task and generated its data, control passes to the next bean which dispatches a request to the CORBA object that implements the computational work of the component. The flow of control between one component and another takes place using the standard bean event mechanism and PropertyChangeEvents trigger this flow. For example, when the householder bean in the example above, obtains the reference to the tridiagonal matrix object, it fires a PropertyChangeEvent that is picked up by the next bean, in this case the QL bean, that has been wired to listen for such events.

The fact that control flow resides entirely in the standard beans on the builder tool is the essential aspect that realizes the combined design and allows the computational steering freedom that we have emphasised throughout.

4 Conclusion

We have analysed the design of a PSE intended to provide good support for computational steering. Then by implementing a prototype incorporating this design we have shown that JavaBeans and CORBA provide an appropriate software basis for small/medium sized PSEs, allowing a high degree of run-time freedom to steer the computation.

We have in mind several improvements to the prototype implementation which would make this PSE a more useful and robust tool. To extend the type of steering that can be done, we would like to construct our own builder tool that would allow beans to be deleted or substituted at run time. To allow the freer introduction and sharing of new components they should be described in some language such as XML [7] along with tools that use this description to create most of the beans and BeanInfo classes as well as providing a skeleton for the computational part of the code. The other major area for improvement is in the scheduler, which should monitor load and provide more sophisticated selections on the basis of some knowledge of the intended taskgraph. However, a significant conclusion of this paper is that the scheduler should not be allowed to gain control of dispatching – or the freedoms of computational steering will be forfeited.

Acknowledgments

DL would like to acknowledge discussions with Peter Lockey and Matthew Shields. This work was supported by a UK EPSRC grant entitled, *Problem Solving Environments for Large Scale Simulations.*

References

1. E. Gallopoulos, E. Houstis and J.R. Rice. *Problem Solving Environments for Computational Science*, IEEE Comput. Sci. Eng., **1**, 11-23, 1994.
   E. Gallopoulos, E. Houstis and J.R. Rice. *Workshop on Problem Solving Environments: Findings and Recommendations*, ACM Comp. Surv., **27**, 277-279, 1995.
2. World Wide Web Consortium, *Extensible Markup Language (XML)* `http://www.w3.org/XML/`
3. `http://java.sun.com/beans/`
4. The CORBA specification is controlled by the Object Management Group `http://www.omg.org/`
5. D. Ridge, D. Becker, P. Merkey and T. Sterling. *Beowulf: Harnessing the Power of Parallelism in a Pile-Of-PC's*, Proc. 1997 IEE Aerospace Conference.
   See the Beowulf Project page at CESDIS:
   `http://cesdis.gsfc.nasa.gov/linux/beowulf/beowulf.html`
6. D. Lancaster and J.S. Reeve. *A Problem Solving Environment Based on Commodity Software*, Proc. HPCN2000. Lecture Notes in Computer Science **1823**, Ed. M. Bubak, H. Afsarmanesh, R. Williams and B. Hertzberger.
7. O.F. Rana, M. Li, D.W. Walker and M.Shields. *An XML based Component Model for Generating Scientific Applications and Performing Large Scale Simulations in a Meta-Computing Environment*, available from:
   `http://www.cs.cf.ac.uk/PSEweb/`