Implementing Problem Solving Environments for Computational Science

Omer F. Rana[1], Maozhen Li[1], Matthew S. Shields[1], David W. Walker[2], and David Golby[3]

[1] Department of Computer Science, University of Wales, Cardiff, POBox 916, Cardiff CF24 3XF, UK
[2] Computational Sciences Section, Oak Ridge National Laboratory, PO Box 2008, Oak Ridge TN 37831-6367, USA
[3] Department of Mathematical Modelling, British Aerospace Systems, Sowerby Research Center, PO Box 5, Filton, Bristol, BS34 7QW, UK

Abstract. A Problem Solving Environment (PSE) should aim to hide implementation and systems details from application developers, to enable a scientist or engineer to concentrate on the science. A PSE is, by definition, problem domain specific, but the infrastructure for a PSE can be problem domain independent. A domain independent infrastructure for a PSE is described, followed by two application dependent PSEs for Molecular Dynamics and Boundary Element codes that make use of our generic PSE infrastructure.

1 Introduction

A Problem Solving Environment (PSE) is a complete, integrated computing environment for composing, compiling, and running applications in a specific area [3]. PSEs have been available for several years for certain specific domains, but most of these have supported different phases of application development, and cannot be used cooperatively to improve a scientists' productivity, primarily due to the lack of a framework for tool integration and ease-of-use considerations. Extensions to current scientific programs such as Matlab, Maple, and Mathematica are particular pertinent examples of this scenario. Developing extensions to such environments enables the reuse of existing code, but may severely restrict the ability to integrate routines that are developed in other ways or using other applications. Multi-Matlab [4] is an example of one such extension for parallel computing platforms.

A PSE must contain: (1) application development tools that enable an end user to construct new applications, or integrate libraries from existing applications, (2) development tools that enable the execution of the application on a set of resources. In this definition, a PSE must include resource management tools, in addition to application construction tools, albeit in an integrated way. Component based implementation technologies provide a useful way of achieving this objective, and have been the focus of research in PSE infrastructure. Based on the types of tools supported within a PSE, we can identify two types

A. Bode et al. (Eds.): Euro-Par 2000, LNCS 1900, pp. 1345–1349, 2000.
© Springer-Verlag Berlin Heidelberg 2000

of users: (1) application scientists/engineers interested primarily in using the PSE to solve a particular problem (or domain of problems), (2) programmers and software vendors who contribute components to help achieve the objectives of the category (1) users. The PSE infrastructure must support both types of users, and enable integration of third party products, in addition to application specific libraries.

Many of these requirements are quite ambitious, and existing PSE projects handle them to varying extents. The component paradigm has proven to be a useful abstraction, and has been adopted by many research projects, making use of existing technologies such as CORBA, JavaBeans and DCOM/COM. Although a useful abstraction for integration, performance issues when wrapping legacy codes in Fortran as components have not been addressed adequately. Automatic wrapper generators for legacy codes that can operate at varying degrees of granularity, and can wrap the entire code or sub-routines within codes automatically, are still not available. Part of the problem arises from translating data types between different implementation languages (such as complex numbers in Fortran), whereas other problems are related to software engineering support for translating monolithic codes into a class hierarchy. Existing tools such as Fortran to Java translators cannot adequately handle these specialised data types, and are inadequate for translating large application codes, such as the Lennard-Jones molecular dynamics application discussed in section 2. For PSE infrastructure developers, integrating application codes provides one goal, the other being the resource management infrastructure to execute these codes. The second of these can involve workstation clusters, or tightly coupled parallel machines. We therefore see a distinction between these two tiers of a PSE, (1) a component composition environment, (2) a resource management system. A loose coupling between these two aspects of a PSE will be useful where third party resource managers are being used, whereas a strong coupling is essential for computational steering or interactive simulations.

2 Applications Using the PSE Infrastructure

We describe two applications that makes use of our generic component based PSE infrastructure described in [8].

Molecular Dynamics: The molecular dynamics application models a Lennard-Jones fluid. It is written in C, and was initially wrapped as a single CORBA object. A Java interface component is combined with a component that wraps the executable code, enabling input from the user to be streamed to the executable, and subsequently to display output results to the user. The CORBA object makes use of an MPI runtime internally, and is distributed over a cluster of workstations. The intra-communication within the object is achieved via the MPI runtime, while the interaction between the client and the object is via the ORB. An application developer does not need to know the details of how the code works, or how to distribute it over a workstation cluster. The wrapper therefore provides the abstraction needed to hide implementation details from

an application user, requiring the developer to automatically handle exceptions generated from the CORBA and MPI system. The molecular dynamics code was subsequently sub-divided into four CORBA objects, to improve re-usability, and included: (1) an `Initialization` object to calculate particle velocities and starting positions, (2) a `Moveout` object to handle communication arising from particle movement and "ghost" regions, (3) a `Force` object which calculates the force between molecules, and constitutes the main computational part of the code, (4) an `Output` object that generated simulation results for each time step. These four objects were managed by a `Controller` object, which coordinated the other four objects. A user now has more control over component interaction, but still does not know how the MPI runtime is initialised and managed within each object. The application developer can construct a molecular dynamics application by connecting components together. Each component is self-documenting based on its XML interface, and component properties can be investigated using an Expert Advisor [7].

BE2D: The BE2D code is a 2D boundary element simulation code for the analysis of electromagnetic wave scattering. The main inputs to the program are a closed 2D contour and a control file defining the characteristics of the incident wave. The original code was written in Fortran. To make the code easier to use within the PSE and also to provide runtime speed comparisons it was decided to convert the code into Java. The use of Fortran to Java converters was considered but abandoned fairly quickly because of the lack of support for complex numbers. Complex numbers caused a number of problems in converting the code, not least because of the fact that they are primitive types in Fortran, but are not directly supported in the standard Java API. A third party implementation of complex numbers for Java was found and used, from Visual Numerics called JNL [5]. Complex number arithmetic proved to be more than just a straight translation from the Fortran code to its Java equivalent. As complex numbers in Java are objects, and Java does not yet support operator overloading, the complex number objects have to use their operator methods, add, multiply, subtract and divide. These methods are either unary or binary, taking a single input parameter which is used to modify the object calling the method or taking two input parameters on a static class method that creates a new result without any modification side effects. Thus calculations involving more than an arbitrary number of parameters become far more complicated in Java than their Fortran equivalents. The converted BE2D was subsequently used as a single component within the PSE. The solver is combined with a graph generator (JChart [6]) as a third party component. The output generated from the code is illustrated in figure 1. If the complete BE2D solver is wrapped as a CORBA object, comprising a single Fortran executable unit, then the interface contains the name of the solver, and a single input file. If the data is to be streamed to the solver, a component that can convert a file (local or at a URL) into a stream is placed before the component.

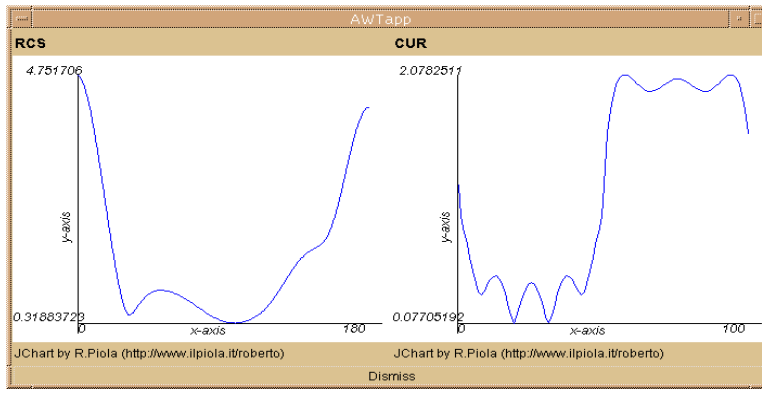

Fig. 1. *Output of the BE2D solver*

3 Conclusion and Future Work

Current advances in networking and distributed object technologies provide the infrastructure to support the creation of general purpose PSEs. The emphasis in developing PSEs should be on creating an environment that is easy to use, and intuitive for an application/domain expert. A computational scientist should therefore not have to configure hardware resources in order to undertake molecular dynamics research, for instance. An important lesson learnt from our work is the necessity to understand the impact of specialised data structures on the usage and performance of legacy codes, when these are wrapped as components within a PSE. The Java programming language can be useful in integrating various components, but it still suffers from the absence of data structures that are essential for scientific codes. Current efforts are under way within the JavaGrande forum [1] to address some of these issues, and the result of these efforts will be significant to future PSE infrastructure.

References

[1] The JavaGrande Forum. See web site at: `http://www.javagrande.org/`.
[2] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Int. Journal of Supercomputing Applications*, 11(2), 1997.
[3] E. Gallopoulos, E. N. Houstis, and J. R. Rice. Computer as Thinker/Doer :Problem-Solving Environments for Computational Science. *IEEE Computational Science and Engineering*, 1(2), 1994.
[4] Vijay Menon and Anne E. Trefethen. MultiMATLAB: Integrating MATLAB with High-Performance Parallel Computing. *Proceedings of SuperComputing97*, 1997.
[5] Visual Numerics. JNL: A numerical library for Java. See web site at: `http://www.vni.com/products/wpd/jnl/`.
[6] Roberto Piola. The JChart package. See web site at: `http://www.ilpiola.it/roberto/jchart/`.

[7] M. Shields, O. F. Rana,D. W. Walker, M. Li, and D. Golby, A Java/CORBA based Visual Program Composition Environment for PSEs, Concurrency: Practice and Experience (in press), 2000

[8] D. Walker, M. Li, O. Rana, M. Shields, and Y. Huang. The Software Architecture of a Distributed Problem Solving Environment. Technical report, Oak Ridge National Laboratory, Computer Science and Mathematics Division, PO Box 2008, Oak Ridge, TN 37831, USA, December 1999. Research report no. ORNL/TM-1999/321.