# A Problem Solving Environment Based on Commodity Software

David Lancaster and J.S. Reeve

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

**Abstract.** Following the common use of of commodity hardware to build clusters, we argue that commodity software should be harnessed in a similar way to support Scientific and Engineering work. Problem Solving Environments (PSE) provide the arena where commodity software technology can modernise the development and execution environment. We describe a PSE prototype based on the standard software infrastructure of Java Beans and CORBA that illustrates this idea and provides advanced PSE functionality at minimum effort for medium sized heterogeneus platforms.

## 1 Introduction

In recent years, scientists and engineers in poorly funded University departments have realised that clusters of PC's provide a cost-effective computing platform for their computationally intensive needs. These clusters [1] are constructed from cheap, off-the-shelf, commodity hardware components and usually run free operating systems such as Linux.

The same forces that have led to the flood of cheap hardware components have also led to the availability of cheap commodity software. It is natural to propose that this software should also be harnessed towards the needs of academic scientists. G. Fox [2] has been a proponent of this idea and has identified the software supporting distributed information systems as being most relevant. In particular, the standardised software that supports web applications, such as CORBA[3], Java[4] and XML[5], is widely available, often freely or for nominal sums.

The proposed role for this commodity software is to modernise the software environment within which scientists and engineers develop and execute applications. The typical software environment for scientific work has changed little over many years, and often remains at the level of command line driven compilers and debuggers. To compile and execute code, the detailed idiosyncracies of each platform in a typically heterogeneus system must be recalled. Problem Solving Environments (PSE's) have been proposed [6] as a more modern environment. A PSE is a software environment that provides support for all stages in both the development and execution of problem solving code. Some PSE's provide further functionality but the essential ingredients are:

– **Development**: The PSE should follow modern software trends by being modular and allowing graphical programming. The components or modules perform well-defined computational functions relevant to the target field of the PSE: for example in engineering they may be solvers of various kinds or grid generators. The solution to a problem is achieved by composing diverse components into a full application that addresses the problem. Composition, which consists in wiring up the components may be done using graphical programming which is widely used in commercial fields and is generally held to improve programmer productivity. This approach enables novel ideas to be tested by rapidly prototyping complex combinations of components.
– **Execution**: A distributed hardware system is assumed, and the the PSE must be able to run each component (typically requiring substantial computation) on remote machines in a transparent manner. The hardware environment available to the users we have in mind is heterogeneous and there is considerable scope for improvements in efficiency by scheduling components to appropriately chosen machines. This is a time-consuming management task, and one gain from using a PSE will be that it avoids the need for the user to be concerned with details of the respective platforms. Infrastructure supporting seamless computing of this kind is provided by various projects such as Globus[7] and Unicore[8].

In this paper we show how a basic PSE with these features can rapidly and simply be put together using commodity software technologies: JavaBeans[4] and CORBA[3]. JavaBeans enable the PSE to use sophisticated graphical tools for development and CORBA addresses the requirements of transparently executing code on heterogeneous platforms.

As befits a prototype system we emphasise simplicity of design. The PSE is designed to use the commmodity layers in such a way as to minimize the additional infrastructure code that must be written and should be regarded as an experimental architecture. In this first step, we have tried to include all the basic functionality but we have not been deeply concerned about the performance. Even so we have incorporated some techniques, notably in data transfer and non-blocking CORBA calls, in order to hide latency.

In the remaining sections of this paper we give a more detailed description of the commodity software technologies, introduce the prototype PSE and explain the roles that the software layers play in this PSE. The next section describes the relevant features of JavaBeans and of CORBA before we proceed to a fuller description of the PSE prototype architecture in section 3. After a brief overview of related work, the conclusion summarises our experiences and mentions some of the directions for further research.

## 2 Commodity Software Technologies

The commodity technologies related to distributed computing that we have in mind include CORBA, Microsoft's DCOM, Java based software and proposals put forward by the World Wide Web Consortium. It is not apparent which, if any, of these will survive in the future. Nonetheless they do have common characteristic features: they are cheap (often with free implementations), widely available and standardized. As a result of strong competition, implementations are robust and of high quality. These are the features needed in designing new software environments for Scientific/Engineering purposes. The resultant systems are intended to combine the necessary perfomance needed for computationally intensive applications with the rich functionality of commodity systems. In this paper we emphasise the functionality, but in the conclusion we will return to discuss the performance.

Our prototype PSE uses JavaBeans and CORBA. These are appropriate for a prototype that is intended to be used with a heterogeneous collection of medium sized machines such as PC clusters, small shared memory machines and individual workstations that run a variety of operating systems. 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. To better organize the computational components and allow them to be easily shared, a large PSE might also benefit from software standards such as XML.

### 2.1 JavaBeans

JavaBeans are reusable software components that can be manipulated visually in a Builder tool such as the Beanbox[9]. The beans are written in standard Java but conform to certain naming patterns that the builder tool uses to discover their properties.

Java beans fulfill the need for a sophisticated user interface that allows graphical programming by connecting computational components. In the prototype PSE, each bean represents a component that performs some computational function. The development stage of the PSE consists of selecting beans from a palette and wiring them together in a builder tool in such a way as to compose the computational components into an application that provides the means of solving the problem at hand. An approach of this sort should be familiar to anyone who has used tools such as AVS [10].

A fully functional PSE would have its own visual tool that provides all the builder functionality to wire up beans besides supporting the organisation of the palette of beans, various pieces of run-time support along with help and guidance tools. However, the aim of this work is to demonstrate how far one can proceed without such proprietary development, and although commercial builder tools may not provide all the functionality desired, they are interchangeable and support the majority of usage scenarios.

### 2.2 CORBA

CORBA provides a framework for distributed computing. The features that endear it to industry are that it is object oriented, that it is inter-operable across platforms and a variety of languages and that it is already a long-lived and well supported standard [3]. To enable the interoperablity, interfaces are defined in a language known as IDL and communication takes place transparently through an ORB. There are many suppliers of ORB's [11] such as Sun who include an ORB in their JDK1.2.

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. The components, or computational modules of the PSE are presented to the user in the form of the JavaBeans described above. These beans are only used to allow graphical composition, the actual computational code is not written in Java as part of the bean, but is in some other language (possibly legacy code) that resides on a different machine. The Java beans are hollow and merely make CORBA calls to the computational code.

For a small PSE with few components, it might be possible to decide in advance where each piece of computational code should run. Then, with the help of the CORBA nameservice the appropriate machine could be selected at runtime. In more sophisticated PSE's a scheduler would be used to select a machine to run the code depending on load and other factors.

## 3 Prototype

The discussion above should already have clarified the overall architecture of our prototype and the roles of the JavaBeans and CORBA. Before proceeding to describe the prototype in more detail, it may be worth being more explicit about one aspect of the overall architecture: the control flow. The whole PSE is based on a data flow model which is intuitive in this context and is made manifest in the way that the beans are wired together in the builder tool. Indeed, the information about the control flow has its origin in the act of wiring, and remains encapsulated in the builder tool. 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. We emphasise this only to contrast it with the case where the control flow information in the builder tool is first abstracted and passed to a separate dispatcher (often combined with scheduler). Our choice of architecture is made for simplicity and to minimize the additional software that has to be written. It turns out that this architecture is extremely flexible and well adapted for computational steering.

To illustrate the use of the PSE prototype and to make the following discussion more concrete we have implemented some simple components that perform operations to diagonalize a matrix. The method of diagonalization is standard and described in [12,13]. Below we list and briefly describe these components. Figure 1 shows how the components would typically be wired together in order to solve a certain problem in solid state physics. However, one of the main purposes of a PSE is that the components can be reused to solve other problems.

– **Hamiltonian Matrix**: Generate a matrix. The matrix corresponds to the Hamiltonian for a simple problem in solid state physics. It is symmetric and has zeros on the diagonal.
– **Random Diagonal**: Add random terms to the diagonal. This corresponds to a disordered medium in the solid state problem. The strength of the random terms is a variable parameter.
– **Householder**: Turn a symmetric matrix into tridiagonal form using a householder transformation.
– **QL**: Diagonalize a tridiagonal matrix using QL decomposition.
– **Visualize**: Display a matrix.

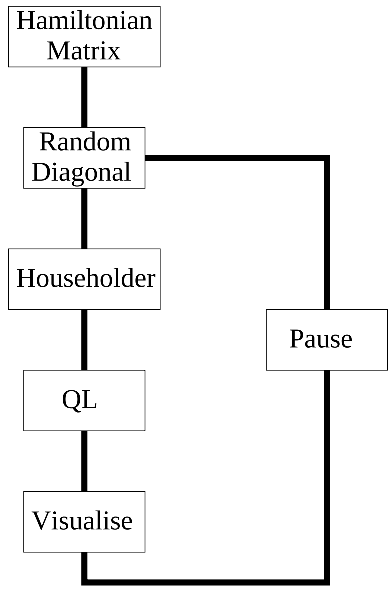


**Fig. 1.** Components wired in a Builder tool to diagonalize a matrix appearing in a problem of disordered media. The function of each component is described in the text. The direction of control flow is clockwise around the loop in this example.

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 desribing 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 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.

As shown in the example in figure 1, there is no need to avoid loops when wiring the components. In this case there is no feedback of data up the loop, merely control flow. We have shown an explicit pause component that delays firing the PropertyChangeEvent and that allows the program to be controlled directly from the builder tool. In the situation shown in the figure, a Hamiltonian matrix of given size is generated once, and then on subsequent traversals of the loop different random terms are added to the diagonal and the resulting matrix is diagonalized. This corresponds to a typical need in the study of disordered systems where many realizations of the disorder must be averaged. A full implementation would contain components that perform the averaging automatically.

As befits a design based on CORBA, the prototype works on a wide variety of platforms, and we have used both workstations and PC's running various flavours of UNIX and Windows NT. We have used this project to test several ORBS with different functionality, including the JDK1.2 ORB, Orbacus[14] and JacORB[15]. Because we needed support for JDK1.2 at the start of this project, the user interface has been based on the beanbox builder tool. However, we expect that the components we have developed can be used in any commercial builder tool.

## 4 Related Work

Although work on PSE's has been continuing for some years, before the recent flood of commodity software technology these efforts tended to use proprietary or specially developed infrastructure and have remained isolated.

The work closest to our own both in spirit and in use of commodity software is Webflow. The Webflow PSE [16] has a three tier architecture consisting of a Webflow editor front end, CORBA middle layer and Globus back end. The commodity middle layer has replaced an special purpose layer in a previous version of this tool. The main thrust of this work has been to allow meta-applications to be created simply at a high level. The connection to the Globus back end enables the PSE to target more substantial computational resources than our prototype. This work is now being used in a major PSE project called the Gateway [17].

Other PSE projects that use commodity software technology include JACO3 which couples simulations with CORBA and Java [18] and PSEware which provides a toolkit for building PSE's [19].

## 5 Conclusion

We have demonstrated the ease with which commodity software technologies can be harnessed to provide advanced problem solving environments targeted for Scientists and Engineers. Our prototype is a proof-of-concept of this approach adapted for medium sized heterogeneous platforms.

As a first step, our demonstration has concentrated on providing the necessary functionality but the main challenge remains to obtain good performance out of systems built from commodity software. This challenge should not be confused with the difficulties in obtaining good performance from any PSE: in general there is a tradeoff between perfomance and flexibility and PSE's are certainly intended to be more flexible than the old-fashioned environment. The difficulty arises in deciding the level of granularity of the components: if the components are large, they will tend to be efficient but inflexible and *vice versa.* In the prototype we have described, CORBA is used to allow each computational component to be implemented in whatever language is appropriate (Fortran can be accomodated even though it has no CORBA binding [20]) and is thus expected to be efficient. The performance problems with commodity based PSE's arise from communication between distributed objects. Work in this area, including high-performance CORBA implementations and comparisons between ORB's may be found in [21]. We have not provided any performance figures for the example included in the prototype because it was intended for illustrative purposes and is not an appropriate benchmark.

With the takeup of commodity software in this area, the range of possible PSE architecures is becoming more restricted so we may expect some convergence leading to the possibility of standardisation. Most useful would be standardisation of the form of the components, and work in this area is already in progress [22].

We also mention the improvements that would make this prototype PSE a more useful and robust tool. To allow the freer import and sharing of new components they should be described in some language such as XML along with tools that use this description to automatically create most of the bean and beaninfo code as well as providing a skeleton for the computational part of the code [23]. The other major area for improvement is in the scheduler, which should monitor load and provide a more sophisticated selection algorithm.

## Acknowledgments

David Lancaster would like to acknowledge discussions with Peter Lockey and Matthew Shields. This work was done in the context of a UK EPSRC grant entitled *Problem Solving Environments for Large Scale Simulations.*

## References

1. 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`
2. G. Fox, W. Furmanski, T. Haupt, E. Akarsu and H. Ozdemir. *HPcc as High Performance Commodity Computing on top of Integrated Java, CORBA, COM and Web Standards.* Proc. of Europar 1998, Springer. 55-74, 1998.
3. The CORBA specification is controlled by the Object Management Group.
   `http://www.omg.org/`
4. Sun's central site for information about Java is at:
   `http://java.sun.com/`
5. World Wide Web Consortium, *Extensible Markup Language (XML)*
   `http://www.w3.org/XML/`
6. 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.
7. I. Foster and C. Kesselman. *Globus: A Metacomputing Infrastructure Toolkit.* Int. J. Supercomp. Appl., **11**, 115-128, 1997.
   `http://www.globus.org`

8. Uniform Access to Computing Resources.
http://www.fz-juelich.de/unicore
9. The Beanbox is a free reference builder tool from Sun. Commercial tools are listed at:
http://java.sun.com/beans/tools.html
10. AVS is produced by Advanced Visual Systems Inc.
http://www.avs.com/
11. A list of ORB's is given at:
http://adams.patriot.net/ tvalesky/freecorba.html/
12. W.H. Press, B.P.Flannery, S.A.Teukolsky and W.T.Vetterling. *Numerical Recipes.* In various languages. CUP.
13. J. Reeve and M. Heath *An Efficient Parallel Version of the Householder-QL Matrix Diagonalization Algorithm.* Par. Comp. **25**, 311-319, 1999.
14. http://www.ooc.com/ob.html
15. http://www.multimania.com/dogweb/
16. T. Haupt, E. Akarsu and G. Fox. *WebFlow: A Framework for Web Based Meta-computing.* Proc. of HPCN 1999, Springer. 291-299, 1999.
17. http://www.osc.edu/~kenf/theGateway
18. http://www.arttic.com/projects/JACO3/default.html
19. http://www.extreme.indiana.edu/pseware/
20. Besides well known techniques of wrapping Fortran, some work on an F90 binding was done in the context of the Esprit PACHA project.
21. http://www.cs.wustl.edu/~schmidt/corba-research-performance.html
22. R. Armstron, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker and B. Smolinski. *Toward a Common Component Architecture for High-Performance Scientific Computing.* Available from the CCA Forum:
http://z.ca.sandia.gov/~cca-forum/
23. 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/