A problem solving environment based on CORBA

David Lancaster
School of Computer Science, University of Westminster, Harrow HA1 3TP, UK
E-mail: lancasd@wmin.ac.uk

We have investigated aspects of the design of Problem Solving Environments (PSE) by constructing a prototype using CORBA as middleware. The two issues we are mainly concerned with are the use of non-trivial (containing more than just a `start`() method) CORBA interfaces for the computational components, and the provision of interactivity using the same mechanisms used for flow control. After describing the design decisions that allow us to investigate these issues, and contrasting them with alternatives, we describe the architecture of the prototype and its use in the context of a study of photonic materials. We argue that having several methods on a component interface can be used to mitigate performance problems that may arise when trying to solve problems in PSE's based on small components. We describe how our mechanism allows a high degree of computational steering over all components.

Keywords: Problem Solving Environment, computational steering, distributed systems, CORBA, resource management, computational components

1. Introduction

Problem Solving Environments (PSE) are intended to support all stages in the development and execution of problem solving code [1]. Support for development is usually interpreted in terms of composition of high-level reusable computational components which form the building blocks of the problem solution code. The PSE aids solution of a problem by enabling these powerful primitive components to be wired together in an appropriate way. The creation or wrapping and incorporation of new components also form part of the development environment. Existing systems that emphasize the development aspect by providing sophisticated graphical tools include Matlab, Iris explorer and AVS [2–4]. Support for execution in PSE's is normally understood in the context of distributed systems and is concerned with resource management and the way that the computational components are scheduled. Systems that emphasize this aspect include well known schedulers such as PBS, and resource management systems such as INTREPID [5,6].

In the context of a project [7] with the aims of investigating the design, implementation, and use of PSE's, we have constructed a prototype PSE. Two design decisions incorporated into the prototype had deep impact on the overall architecture. One was the choice to use CORBA and the other was insistence that interactive control such as computational steering [8] should be at the heart of the architecture. The consequences of these design decisions are discussed at length below, but the main issues may be summarized as follows.

- The distributed object approach of CORBA certainly provides infrastructure for a more maintainable system than older RPC based approaches. Our work devotes more attention to another interesting feature: the ease with which several distinct methods can be called on a single component. We explore how this feature might be useful in a PSE.
- Interactive features tie together the development and execution aspects of the PSE. Our prototype tests an approach in which all components are controllable using only the ordinary flow control mechanisms and the original development environment.

This paper describes our prototype PSE and uses it as context to report on some of the general issues concerning both the development and execution aspects of PSE's. We note from the outset that this prototype was intended for use, and tested, in a local setup consisting of a variety of workstations, PC's and cluster machines, and this has naturally restricted the range of our investigation. Firstly we describe the two design decisions and review alternative choices. We then present the architecture of the prototype PSE. The consequences of the design decisions in the prototype architecture are explored. Detailed discussion of the execution side

Scientific Programming 9 (2001) 233–242
ISSN 1058-9244 / $8.00

of the PSE includes a description of the mechanism of data transfer, the use of the Event service and an overview of system start up. The main aspect of the development side of the PSE is the front-end and we devote a section to discussing various ways in which this may be implemented.

2. CORBA

At an early stage in the project a decision was made to use CORBA [10]. This is a significant departure from traditional execution environments that tended to rely on a RPC-like paradigm such as `rsh`. There are many consequences to this choice, so let us review the traditional approach.

2.1. Alternative execution environments

Traditional execution environments run large production applications on distributed machines in a transparent manner that avoids the need to be concerned with issues of licenses, platforms etc. These production applications typically consist of one or a series of large components, each of which is a wrapped legacy stand-alone program. For example, in an engineering context, the components might be a mesh generator and a solver. Each individual component performs substantial computation that may involve several stages of an algorithm. These large components require powerful computing resources so the execution environment often has a wide geographic range.

This kind of large-scale execution environment also addresses the requirements for coupling large existing applications such as Ocean and Atmospheric simulation codes. The aim is to ease the management of all the arcane details that must be considered when coupling codes, for example translation between file formats.

INTREPID [6] is an example of an execution environment based on `rsh` (though a separate version for Windows NT machines was later developed). It allows a set of components, a so-called "task graph", to be executed on distributed machines either directly or via interfaces to various queuing systems. This system has been used on a large-scale [6], but has been more frequently employed on the smaller scale that is the focus of this paper.

More recent and sophisticated environments operate on the scale of Grids. For example, Globus [9] is a Grid infrastructure that provides organization to manage a set of large-scale shared resources.

2.2. Distributed objects

CORBA [10] is a mature distributed object system that allows objects to communicate irrespective of implementation language, platform or Object Request Broker (ORB) vendor. Its use as infrastructure for a PSE may appear a little odd in that CORBA emphasizes transparency of location, whereas the execution layer of the PSE is intended to explicitly schedule location. Although there is nothing in principle to prevent CORBA from being used on a large-scale, there are often practical difficulties related to passing IIOP through firewalls.

What does CORBA bring to a PSE, that is not possible with the traditional approach, such as INTREPID, sketched above? There are two benefits – one is convenience and the other is related to new possibilities arising from a more sophisticated component interface.

The convenience comes from having the same approach irrespective of platform – it is straightforward to employ both UNIX and Windows machines. Language independence allows objects that require a GUI to be written simply in Java, and other objects that may interface to high performance computational components to be written in C++ (or C). Furthermore there are many CORBA services and provisions that support functionality desirable in a PSE. For example, the prototype described below makes use of the Event service as well as creation on demand and persistence. These are very real advantages in developing and maintaining a PSE, but do not necessarily move forward its capabilities.

2.3. Component interfaces

The possibility of new PSE capabilities lies in the distributed object nature of CORBA. The IDL that specifies the interface to each component can contain multiple methods, each of which is more sophisticated than a simple `rsh` command. To illustrate how this flexibility could be used in a PSE consider first the simple IDL:

```
void start();
```

This interface will reproduce the behavior of a component that is initiated using the `rsh` method. One immediate benefit of using CORBA lies in the ease of introducing error condition feedback. The `start()` method can be made to throw an exception that indicates the reason for the error. Such capability is valuable in a distributed system where the possibility of partial failure must be accounted for.

A more significant advantage of CORBA lies in the possibility of more sophisticated components that can make use of more than a single method. The essential reason for considering such components is improved performance based on better computational efficiency. Small components tend to suffer from not being able to use numerical strategies that would be possible if they were larger and performed more steps of a computation. Rapid prototyping PSE's rely on having a wide choice of flexible small components in order to compose the solutions of problems with the minimum necessity of writing code for special purpose components relevant to the application area. Performance is still important for such PSE's, and by using more sophisticated interfaces, it is possible to mitigate the performance losses incumbent in the use of small components. Although similar behavior could be imagined in a conventional RPC approach (by passing flags), it is not hard to see that this would soon become unwieldy.

The argument that small components suffer from not being able to take advantage of numerical strategies available to larger components depends strongly on the particular application. We give one such example for the prototype PSE in Section 4.2.

2.4. Other CORBA issues

The use of CORBA imposes a strongly modular structure on the overall system as it is a distributed object system. Interfaces to each object are precisely defined and objects can be located at will. This has been used to make the scheduler and system management portions of the prototype system separate objects as described in Section 4.1.

One potential problem with using CORBA for PSE concerns the way CORBA can be used to initiate and control components that are themselves parallel. We do not investigate this topic in detail, but draw the readers attention to the fact that besides various unsatisfactory workarounds, there is a proposal to extend the CORBA specification that would simplify the way CORBA and MPI work together [11].

Neither do we consider the interaction between the PSE and queuing systems that might exist on any of the target platforms. The problems that this causes are discussed in Section 4.5.

2.5. Summary

To summarize this section: CORBA provides sophisticated component interfaces that can be used to make small components that do not necessarily loose performance when combined. Such small components are flexible in that they can be combined in more ways, and are necessary for the kind of PSE that supports rapid development.

3. Computational steering

Another design decision incorporated from the start was to insist on interactive control over the execution. In particular, the prototype PSE allows computational steering in the general sense of being able to alter the choice of all components and their connections at run time.

3.1. Special steerable components

The conventional way of building interactivity into PSE's works even when the development and execution phases are separate: imagine that the development phase leads to a file describing the component connectivity, and that this file is passed to a separate resource manager for execution. Computational steering is possible as an attribute of special purpose, so called "steerable" components. At run time, these special components communicate with external tools that allow them to be manipulated [13,14]. The approach envisioned here is more general and attempts to use the development tools and standard mechanism of control flow to provide interactivity for all components.

3.2. Interactivity and flow control

The kind of general interactivity we have in mind is to exchange one solver component for another, or to add a visualisation component at run time. To control parameters of components at run time, as in the more conventional sense of steering, we merely need appropriate CORBA methods on the component and a suitable front end. The mechanism of steering is the same as that used for the ordinary control flow.

The prototype PSE is designed to enable computational steering using the same interface that the user employs for composing the problem. This brings together the development and execution phases of the PSE and forces several further design decisions: the flow of execution control resides in the development tool; scheduling is dynamic and only occurs when a component is ready for execution. The level of in-

teractivity contemplated places the control flow in the front-end.

For a more detailed discussion of computational steering in this model and comparison with the alternative see [8].

3.3. Summary

We contemplate a form of computational steering that is inherent in the design and allows changes in the components and their connection while the system is running. This binds the development and execution stages of the PSE closely together and has its most apparent consequence in a dynamic scheduler. The final scheduling choice is made at run time, just as the component is due to be executed.

4. Prototype PSE

In order to explore the potential of using CORBA in a PSE we wanted to use more sophisticated component interfaces than would be encountered by simply wrapping legacy applications. We therefore searched for an application area that would need to wire small components in a variety of ways.

The application area for the prototype is that of finite element calculations for a study of photonic materials. This is a relatively new subject, and a variety of similar calculations are being used to study the influence of various geometries and materials on the bandgap. A code happened in the process of being written [15], and it was possible to collaborate with the developers to create a component based system. As we discuss in Section 4.2, access to the details of the solver allowed us to implement more than the simple CORBA interface that a wrapped component would normally expose. Despite close contact with the developers, we found that turning a working program into components at the level of detail desired was quite time-consuming.

A notable aspect of the code was that it is written in C++, thereby allowing a direct connection with an Object Request Broker (ORB) without any intermediate wrapping stage. Much of the rest of the PSE infrastructure was written in Java as this allowed portable graphical interfaces to be quickly created via another CORBA language binding.

We employed Orbacus 4.01 [16] as this supported a POA, had the necessary language bindings and provided an Event service.

4.1. Architecture

A diagram exposing the overall structure of the prototype is shown in Fig. 1. The labeled blocks represent separate CORBA interfaces and the lines indicate the most important traffic between them. The resource management portion of the PSE is contained in the three blocks at the centre of the figure. The composition environment and driver of the system is shown on the left; we postpone discussion of how this is realized because several possibilities were considered. As usual in distributed object systems, the code representing each object or block could be run on a separate machine, but for reasons of efficiency it is sensible to co-locate certain parts. The administrator who sets up the PSE is responsible for this choice, and will normally co-locate all the resource management objects. In the event that the system becomes larger, and the scheduler more complicated, this choice would have to be reconsidered. On the right of the figure are shown a series of machines used to execute the computational components. Each of these machines has a so-called monitor module running on them. This CORBA object is used to feed the live load information and also serves as a factory-like interface that can create component objects on demand (standard POA methods are used to allow this).

The three parts of the resource management subsystem function as follows. The central database module (labeled datastore) stores information about the components and execution machines in the system. Information about components includes lists of which components are available on which machines. Information about the execution machines includes their immediate load as gathered by the monitor module. This also provides some rather basic level of fault tolerance by checking the machines are alive. The live load information is used by the scheduler module along with the datastore. Not shown are various graphical system management tools that can be used to change scheduler policies, query the central database and shutdown the system cleanly.

The CORBA interfaces used for data transfer are not shown in Fig. 1, because they only exist temporarily. The mechanism of data transfer is discussed below.

4.2. Components

In the photonics application area chosen for the prototype, there are two particularly important components. One assembles finite element matrices on the

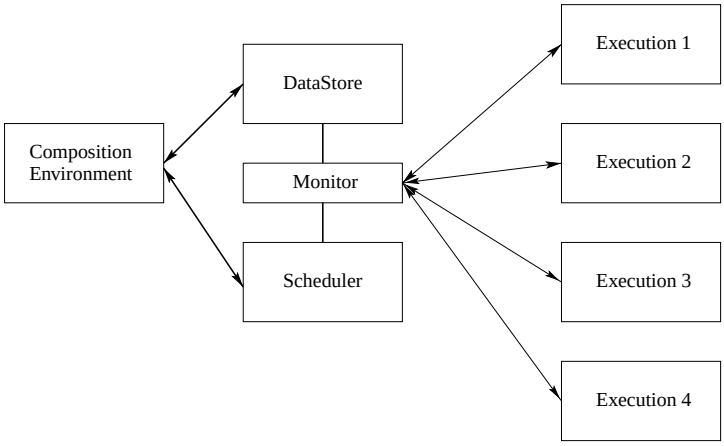

Fig. 1. Architecture of prototype PSE.

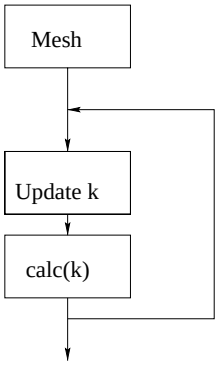

Fig. 2. Simple components connected in a loop to generate a series of eigenvalues.

basis of a mesh provided either by a stored file or another component and the other is a solver that searches for the lowest eigenvalue of the matrix. Components such as these would form the basis of any finite element calculation. It was because of our interest in providing non-trivial CORBA interfaces that we constructed the components ourselves rather than wrapping some standard publicly available code. Nonetheless, the resulting components, at least the solver, could be used in fairly general finite element problems.

One photonics problem required a whole series of such eigenvalues to be determined as parameters (a vector k) determining the matrix smoothly changed. This could be achieved using simple components and connecting them in a loop as shown in the following figure.

This loop includes a special component just to update the k parameter and provides a highly interactive way of solving the problem because the k parameter can be controlled either automatically or by the user. As it stands, the eigenvalue solver component is standard: it accepts a matrix and generates the eigenvalues.

However, this approach would be numerically inefficient because the solution strategy in the eigenvalue solver is iterative and is considerably faster when given the eigenvector solution of the previous matrix as starting iteration. In a stand-alone application, this performance improvement is achieved by combining the loop, k update and solver components shown in Fig. 2 into what is effectively one large component. Large special purpose components of this kind have been imported into PSE's [12], however this is an example of the use of PSE's to couple applications (ie. multiphysics) rather than using them to support rapid solution of new problems. The emphasis of this paper is placed on smaller, general purpose PSE's for which the components should be fairly standard. In order to have a standard solver component, yet not abandon the performance advantages mentioned above, the component can be modified so as to include in the component interface an additional method that allows the eigenvector to be stored between consecutive calls.

This example suggests that, for performance reasons, a variety of different ways of calling the solver should be provided. Although one could imagine non-CORBA components that would allow this, the provision of a set of related methods on the CORBA component interface is natural. For example, in the case above, the IDL includes the following methods:

```
void calc();
void calc(double initial_lambda);
```

By selecting the appropriate method, a way of solving the problem that retains the performance advantages

of larger special purpose components, yet allows flexible interaction can be devised. In general one might expect a much larger number of distinct but related methods.

We have not addressed the performance problem from having small components in a loop. There is certainly a CORBA overhead, but much more serious is the possibility that on subsequent traversals of the loop, the component may be scheduled on a different machine. To avoid this, we have allowed the scheduler to be overridden so that the user can specify that a certain component always runs on a particular machine. A better solution would involve a more sophisticated scheduling algorithm that takes account both of the overhead of moving a component as well as the load on the machine.

4.3. Data transfer

Data must be transferred between components that are running on different machines of the distributed system. The traditional mechanism has been to use input and output files, and many schedulers include mechanisms to stage the files between machines.

The prototype PSE has been built with a flexible data transfer mechanism. File staging is included but since the prototype has only been used for small problems it was also possible to transfer data directly from main memory. The mechanism relies on the creation of additional CORBA interfaces that allow data access. These interfaces are incarnated dynamically as the computational component runs and remain present after it has completed its work. For the matrices of the photonic problem, the interface is called:

```
DataTypes::Matrices::Basic_
Matrices;
```

This interface provides simple methods that allow the matrix data to be transferred directly. In general, the data interfaces extend a base data interface, `DataTypes`, adding additional methods dependent on the type of data that is to be transferred. The data types depend on the application area, but we anticipate a library of such types in a full PSE.

For efficiency, and to aid scaling, transfer must be directly between components rather than through some central facility. The difficulty is that the machine where the target component is to run, is unknown until the previous component has completed and the scheduler invoked. Lightweight location information (an IOR for our CORBA based system) is passed through the central system and on to the target component as soon as its location has been decided. Data transfer itself relies on a request from the newly active computational component to the data interface `DataTypes::Matrices::Basic_Matrices` that remains after the old computational component has completed.

To transfer the lightweight information describing the IOR of the data interface the component interface itself includes get and set methods such as:

```
DataTypes::Matrices::Basic_Matrices
getmatrixref();
```

A means of clearing up old data interfaces is clearly necessary and a mechanism to automatically do this was developed.

4.4. Control flow

When control passes from one component to the next, a signal indicating completion of the first component is necessary. The method by which the monitor module regularly polls the component is clearly inefficient, so methods in which the component contacts the central module are preferred.

A straightforward solution is to use callbacks, in which an IOR on the monitor module is passed to the component. In practice this method is awkward because it requires explicit thread programming on the component. This is a familiar situation in CORBA systems [17] and the solution is to use the so-called Event service.

The Event service decouples the callback and simplifies the design. The so-called “push” model is used in which the components make a push call to the event service on completion of the calculation. The front-end registers a listener to the event service that is notified when the event service receives the push call. This mechanism can be used even for the script-like driver front-ends introduced below. The CORBA notification service would be even more appropriate in this role, but was not mature at the time of writing the code.

4.5. Start up

In a working CORBA based system it is usual to have CORBA objects running permanently on all the machines of the distributed system. This appears to present a much higher overhead than is the case for `rsh` based systems. The benefits are however overwhelming, in particular to resource management, allowing

monitoring of the health of the system and providing load information to the central monitor.

CORBA does not require that the components themselves be memory resident while not in use. Early versions of the PSE prototype were explicitly coded in such a way as to avoid this, and now the CORBA 2.3 specification includes the Portable Object Adaptor (POA) that provides support for activation on demand.

It is anticipated that the PSE should be permanently available on the system, and that the component servers on each of the distributed machines should be continuously running. The responsibility of admistrator who first installs and sets the system running is first to choose machines for the resource management and execution parts of the system. He/she then installs and initialises the nameservice and eventservice, followed by the resource management modules. Execution machines are set up with the help of a short configuration file listing the computational components that they should support, as these may be different on different machines. The Component Server that is started on each execution machine starts to monitor the load on the machine and registers itself with the resource manager. The administrator is provided with some startup utilities that make use of the CORBA naming service. We have used persistent object references so that an execution machine may be restarted and still have the same IOR for the interfaces it presents. This simplifies some aspects of fault tolerance.

As was explained in the introduction, we are concerned with small-scale PSE's that are to be run on clusters of workstations. We have therefore not addressed issues relating to machines that can only be accessed via a queueing system, and upon which it is difficult to imagine a permanently running CORBA server. A different model would be needed to accomodate such systems, and given that such machines tend to have elaborate security, a CORBA based approach would suffer practical difficulties in any case.

4.6. Composition environment

The development tool in a PSE is employed by users to compose components and thereby construct the complete program to solve the problem. There are various ways in which this can be done and we experimented with several front-ends in the prototype PSE. We expect that in any product PSE, design of the front-end would rely on experts in Human Computer Interface design, and indeed on the prospective users themselves. In the meantime, the prototype was able to interface with several front-ends, some quite simply intended to drive and test the execution side, and others with some GUI support for visual composition.

The simplest composition tool is an ordinary text editor used to create scripts that call a sequence of components. The script could be written using Corbascript [18] or some other scripting language [19] that is able to communicate with CORBA. This is a very low-level approach that may be convenient for basic testing. We preferred a slightly higher-level approach based on the Java language, that uses the editor to write a simple driver class to call the components in the desired sequence. Utilities were written that allowed this to be done in such a way as to use all the PSE facilities, but minimizing the amount of code required for the driver itself. An example is shown below, corresponding to a simple sequence of two components.

```
public class driver
{
    private static PSE_CORBA.
    Central.DataStore ds_central;
    private static PSE_CORBA.
    Scheduler.Algorithm
    al_scheduler;

    public static void main
    (String args[])
    {
        PSE_CORBA.Parameters.PHEM
        Params =
            new PSE_CORBA.
            Parameters.PHEM(.....);
        NameResolver NR =
        new NameResolver(args);
        ds_central =
        NR.getDataStore();
        al_scheduler =
        NR.getAlgorithm();
        ESlistener ESL =
        new ESlistener(args);
        ESL.start();

        assembler_rep r1 = new
        assembler_rep
                  (ds_central,
                  al_scheduler,
                  ESL, Params);
        solver_rep r2 = new
        solver_rep
                  (ds_central,
```

```
                    al_scheduler,
                    ESL, Params);

        r1.addcontrolflow(r2);

        r1.calc();
    }

}
```

After some boilerplate initialisation to set up the scheduler and event service (via utilities), there are two lines creating proxies or "representations" of the components named `r1` and `r2`. The line `r1.addcontrolflow(r2);` connects these components. Finally the computations are initiated by starting the calculation on the first component. This system was simple to use and drive the execution side, but did not offer much by way of interactivity.

A simple GUI interface was constructed using Java Beans and in effect hijacking the beanbox (or in principle any bean composition tool) as the visual editor. This allowed a GUI to be constructed quickly with the minimum of explicit graphics programming [20]. This GUI allowed components to be wired up graphically and the execution to be run directly. It also allowed computational steering in the conventional sense, by varying component parameters at run time using bean properties. The more general level of interactivity was supported in so far as additional components could be added at run time. Limitations of the bean box prevented us from removing components at run time. This approach was flexible and loops of control flow could be set up, but it suffered from some inadequacy due to its origin not being specific to the problem.

A more sophisticated GUI was constructed in Cardiff [21] as part of the same project. This was a custom built piece of software that has a visual interface better adapted to showing the connections between components than the Java Bean based version. It also has a method of execution that can be adjusted for the particular task in hand.

Applet based front-end systems were also explored, but due to security issues and the lack of appropriate CORBA support in common browsers, this approach was abandoned.

5. Other PSE's

Numerous PSE's have been built over the last few years and it is not possible to review them all here. We restrict ourselves to the less common PSE's based on CORBA. These PSE's tend to be focused on particular application areas and an early example is Webflow [22]. A more recent and advanced example is Applab in the biological sciences application area [23]. This PSE has the advantage of a clear component interface model that has been adopted by OMG [24] and a set of existing applications that can be accessed. It uses XML metadata to describe the functionality of the applications and to record information about each individual analysis performed. The Distributed Resource Management (DRM) portion of the ASCI computational grid is built using the Globus toolkit, and a CORBA service layer has been implemented above this [25]. The motivation for this work was to provide a layer that would allow existing PSE's that used CORBA wrapped components to easily integrate with the DRM services. Another project that uses CORBA, allbeit only for security reasons is the Mississippi Computational Web Portal [26]. This project uses Enterprise Java Beans as middleware, but the actual access to back end resources is via a grid interface. The use of CORBA is for secure transport of web requests between the front end and middle layer. This project emphasises the collaborative and recording aspects that are important in a PSE designed to be used within an organisation that wishes to preserve expertise in a convenient form.

Our prototype PSE was built to investigate the two issues of using non-trivial CORBA interfaces to components and of implementing computational steering via the ordinary control flow mechanisms. These exploratory directions are not addressed by the PSE's mentioned above.

PSE's are now being designed to operate on the larger scale provided by grids, and issues of performance are more critical than in the smaller CORBA based systems. For example, the work on component based scientific computing [27] concentrates on maintaining the performance of assembled scientific components through a variety of optimization techniques. The same authors have also considered the problem of mapping components to grid resources using a marketplace approach.

6. Conclusion

We have considered some of the design issues that arose in our construction of a prototype PSE. The principal points concern the the incorporation of computational steering and the use of CORBA.

Our prototype demonstrates the feasability of providing a high degree of interactive control over all components. This avoids the need for any special communication channel for steerable components, but places the ordinary flow of control in the front end. This level of interactive control is probably only necessary in a small-scale PSE used for flexible rapid prototyping.

The immediate benefits of using CORBA as PSE infrastructure are the well known advantages of distributed object systems related to maintainability. We have identified another feature: that CORBA makes it easy to have several distinct methods acting on a PSE component. We have argued that this feature could be used to compensate for the numerical inefficiencies that are sometimes inherent in using standard small components. A selection of such components are necessary to allow flexible compositions of problem solutions in a small-scale rapid prototyping PSE.

The difficulty with the sophisticated (though natural in CORBA) component interfaces that are proposed here, is their standardization. None of the components of the PSE's mentioned in Section 5 are interoperable. The time for such a move is ripe and several bodies have already started work towards a standard. We expect more from CORBA through the component model, and indeed see that OMG has supported some standardization in the context of biological sciences [24]. Another body supporting standardization is the Common Component Architecture Forum (CCA). They hope to define a minimal set of standard features that a High-Performance Component Framework has to provide, or can expect, in order to be able to use components developed within different frameworks. This framework emphasizes performance [28].

Acknowledgments

This work was supported by an EPSRC grant entitled *Problem Solving Environments for Large Scale Simulations*. I would like to thank J.M. Generowicz for making available and explaining his photonics code.

References

[1] E. Gallopoulos, E. Houstis and J.R. Rice, Problem Solving Environments for Computational Science, *IEEE Comput. Sci. Eng.* **1** (1994), 11–23.
E.Gallopoulos, E.Houstis and J.R.Rice. Workshop on Problem Solving Environments: Findings and Recommendations, *ACM Comp. Surv.* **27** (1995), 277–279.

[2] Matlab is produced by MathWorks Inc., http://www. MathWorks.com/.

[3] Iris Explorer is a product of NAG, http://www.nag.com/.

[4] AVS is produced by Advanced Visual Systems Inc., http://www.avs.com/.

[5] PBS the Portable Batch System is available at: http://pbs.mrj.com/.

[6] K.E. Meacham, N. Floros and M. Surridge, Industrial Stochastic Simulations on a European Meta-Computer, Springer Verlag, *Proc. EuroPar'98 LNCS* **1470** (1998), 1131–1139.

[7] The project was entitled *Problem Solving Environments for Large Scale Simulations* and was funded by the UK EPSRC. It started in November 1998 and involved Southampton University, Southampton IT innovation Centre and Cardiff University.

[8] D. Lancaster and J.S. Reeve, Computational Steering in Problem Solving Environments, Springer Verlag, *Proc. EuroPar'00 LNCS* **1900** (2000), 1340–1344.

[9] I. Foster and C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit, *Int. J. Supercomp. Appl.* **11** (1997), 115–128, http://www.globus.org.

[10] The CORBA specification is controlled by the Object Management Group: http://www.omg.org/.

[11] T. Priol and C. Rene, COBRA: A CORBA compliant programming environment for high performance computing, Springer Verlag, *Proc. of EuroPar '98. LNCS* **1470** (1998), 1114–1122.
OMG, *Data Parallel Application Support RFP,* orbos/00-03-17 available at http://www.omg.org/.

[12] M. Li, O.R. Rana, M. Shields and D.W. Walker, *A Wrapper Generator for Wrapping High Performance Legacy Codes as Java/CORBA Components.* Proc. of SuperComputing 2000, IEEE Computer Society Press, Dallas, Texas, USA, November 2000.

[13] G.A. Geist, J.A. Kohl and P.M. Papadopoulos, CUMULVS: Providing Fault-Tolerance, Visualization and Steering of Parallel Applications, *International Journal of High Performance Computing Applications* **11** (1997), 224–236.

[14] S.G. Parker and C.R. Johnson, *SCIRun: A Scientific Programming Environment for Computational Steering,* Proceedings Supercomputing '95. ACM/IEEE Computer Society 1995, online publication on http: //www.supercomp.org/sc95/proceedings/.

[15] B.P. Hiett, J.M. Generowicz, S.J. Cox, M. Molinari, D. Beckett, G.J. Parker and K.S. Thomas, *Finite Element Modeling of Photonic Crystals*, Proc PREP 2001, 2001, pp. 87–88.

[16] Orbacus is produced by Object Oriented Concepts Inc: http://www.ooc.com/ob.html.

[17] M. Henning and S. Vinoski, *Advanced CORBA Programming with C++*, Addison Wesley, 1999.

[18] http://corbaweb/lifl.fr/CorbaScript.

[19] Several scripting languages have CORBA extensions: Perl: http://www.lunatech.com/research/corba/cope/, Python: http://www.fnorb.com/, ILU: ftp://ftp.parc.xerox.com/pub/ilu/ilu.html.

[20] D. Lancaster and J.S. Reeve, Problem Solving Environments based on Commodity Software, Springer Verlag, *Proc. HPCN'00, LNCS* **1823** (2000), 3–11.

[21] M.S. Shields, O. Rana, D.W. Walker, M. Li and D. Golby, *A Java/CORBA-based visual program composition environment for PSE's, Concurrency – Practice and Experience* **12** (2000), 687–704.

[22] T. Haupt, E. Akarsu and G. Fox, *WebFlow: A Framework for Web Based Metacomputing*, Proc. of NPCN 1999, Springer, 1999, pp. 291–299.

[23] Applab is part of a project, OpenBSA, to provide a freely available implementation of the Biomolecular Sequence Analysis specification. It is being developed at the European Bioinformatics Institute, http://bach.ebi.ac.uk/openBSA/.
[24] OMG, *Draft Adopted Specification for Biomolecular Sequence Analysis*, lifesci/99-12-01 available at: http://www.omg.org/.
[25] ASCI, *Constructing the ASCI Computational Grid*, available at: http://www.sandia.gov/supercomp/sc99/drm-sc99.pdf.
[26] T. Haupt, P. Bangalore and G. Henley, *A Computational Web Portal for the Distributed Marine Environment Forecast System*, Available at: http://www.computingportals.org/.
[27] S. Newhouse, A. Mayer and J. Darlington, A Software Architecture for HPC Grid Applications, Springer Verlag, *Proc. EuroPar'00 LNCS* **1900** (2000), 686–689.
[28] 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*, The CCA Forum web site is at: http://www.acl.lanl.gov/cca-forum/.