Component Based Problem Solving Environment

A.J.G. Hey[1], J. Papay[2], A.J. Keane[3], and S.J. Cox[2]

[1] EPSRC, Polaris House, North Star Avenue, Swindon, SN2 1ET, UK
Tony.Hey@epsrc.ac.uk
http://www.epsrc.ac.uk
[2] Department of Electronics and Computer Science, University of Southampton, Southampton, SO17 1BJ, UK
{jp, sc}@ecs.soton.ac.uk
[3] Computational Engineering and Design Centre, University of Southampton, Southampton, SO17 1BJ, UK
ajk@soton.ac.uk

Abstract. The aim of the project described in this paper was to use modern software component technologies such as CORBA, Java and XML for the development of key modules which can be used for the rapid prototyping of application specific Problem Solving Environments (PSE). The software components developed in this project were a user interface, scheduler, monitor, various components for handling interrupts, synchronisation, task execution and software for photonic crystal simulations. The key requirements for the PSE were to provide support for distributed computation in a heterogeneous environment, a user friendly interface for graphical programming, intelligent resource management, object oriented design, a high level of portability and software maintainability, reuse of legacy code and application of middleware technologies in software design.

1 Introduction

A Problem Solving Environment (PSE) is an application-specific environment that provides the user with support for all stages of the problem solving process - from program design and development to compilation and performance optimization [1]. Such an environment also provides access to libraries and integrated toolsets, as well as support for visualization and collaboration. The main reasons for the development of PSEs are to simplify the usage of existing software modules, simplify problem specification, solution and to maximize the utilisation of distributed computing resources.

In this project we investigated in detail the design, implementation and use of several PSEs applied to large-scale computational problems. These PSEs were Promenvir [2], Toolshed [3], GasTurbnLab [4], Autobench [5] and SCIRUN [6]. The reason for analyzing these projects was to draw lessons and to evaluate the pros and cons of various design options. There were numerous lessons drawn which were related to portability of software components, Object Oriented Design, application of middleware, fault tolerance and performance modelling. Here we summarise the main ideas influencing the software design.

B. Monien and R. Feldmann (Eds.): Euro-Par 2002, LNCS 2400, pp. 105–112.

All software except the FEM solvers used for photonic crystal simulations were developed using the Java [7] programming language. This provided platform independence, portability, high level reliability and security. Moreover, Java proved to be more elegant and more suitable for rigorous object oriented design than the more complex C++ language.

During the project considerable time was spent on the evaluation of emerging middleware technologies and products. Several alternatives for the software architecture were evaluated by taking into account the complexity of the application, capability of handling complex data structures, capability of handling applications written in different languages and running on different platforms. After considering and testing various component technologies, we opted for CORBA based middleware [8]. The use of sockets and other mechanisms was also considered, but these options were later dropped because of the increase in complexity of design. CORBA provides a standard mechanism for defining the interfaces between components, a compiler enabling the use of interfaces in other programming languages, services such as the naming service, implementation repository, event service etc., a communication mechanism enabling objects to interact with each other, programming language and platform independence. CORBA also provides a portable and reliable platform for transparent remote object access and communication and clearly demonstrated the suitability of this technology for the design of large scale distributed systems.

Learning from the lessons of previous projects we decided to use a database rather than implementing complicated data structures such as linked lists, tables, stacks, etc., for storing and manipulating monitoring and scheduling information. The use of the database simplified the software architecture and significantly improved the flexibility of the design and its capacity for evolution.

During the PSE development, issues of fault tolerance and error recovery were given high priority. In this respect the PSE provides availability checking and lifetime control of remote objects. The robustness of the system was tested by simulating various scenarios during which individual servers were switched off.

Performance Engineering (PE) played an important role during the scheduler design. PE is concerned with the reliable prediction and estimation of resource requirements and performance of applications. Several performance models were developed, these models were used by the scheduler and performance estimator for execution time prediction and achieving better utilisation of the available computing resources.

The XML technology was extensively used in the project. The main advantage of XML is that it provides platform independent data exchange and a framework for the specification of data structures. XML was used for the design of user interfaces and as a configuration language for describing the interconnection of tasks, their resource requirements and performance models.

The selection of an appropriate middleware product is of key importance for the software design. Various CORBA based middleware products were tested for their performance, reliability, user support and suitability for distributed PSE

development. These products were the Sun ORB, Orbix [9] and Orbacus [10]. Initially the Sun ORB contained in the Java Development Kit (JDK) was used, however, at the time of evaluation it had limited functionality and its speed and reliability were unsatisfactory. Both Orbix and Orbacus offered the required functionality and complied with the CORBA 2.3 specification. Moreover Orbacus is a shareware product which can be an advantage for exploratory academic research, considering the risks associated with a fast moving middleware market. Orbacus includes C++ and Java implementations and provides several services: Naming, Event, Property, Time, Trading, Notification, and Logging. During the lifetime of the project several versions of JDK and Orbacus were released. As a result, components already developed had to be modified in order to maintain compatibility and to take advantage of the improved functionality of new releases. The final version of the PSE prototype was build by using JDK version 1.3 and Orbacus 4.0.5.

2 PSE Architecture

The PSE developed in this project is a distributed component based architecture (see Fig. 1). The reasons for opting for this design were the isolation of components, scalability of the system, flexibility of software deployment and updates, and resistance against the side effects of software modifications.

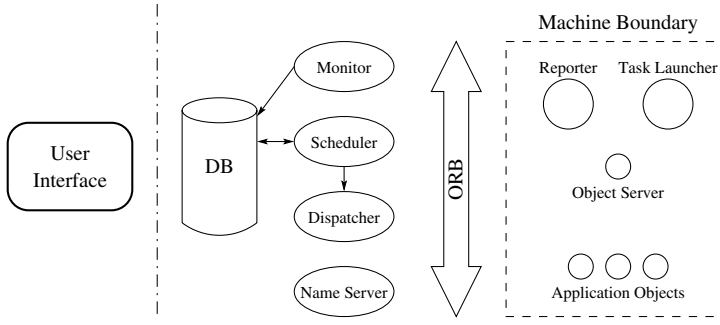

Fig. 1. Architecture of the prototype PSE

The user interface is a graphical Java front-end which provides numerous functions such as task-graph composition, parameter setup, lifetime control of distributed objects, monitoring the status of machines and tasks, computation steering and visualization. The user interface was developed in cooperation with Cardiff University [11]. The output of the graphical composer is a task-graph which describes the execution order, data dependencies and parallelism. This form of problem specification corresponds with the data flow model of computation in which individual tasks consist of programs and parameters which specify

their function, characterise their resource requirements and performance. The resource requirements are expressed in terms of memory size, communication and I/O traffic, and disk volume used by the application. An example of a task-graph representing a parallel version of photonic crystal simulation is given in Fig. 2. The user interface also contains a separate control panel which provides life time control of objects, computational steering, task and machine monitoring.

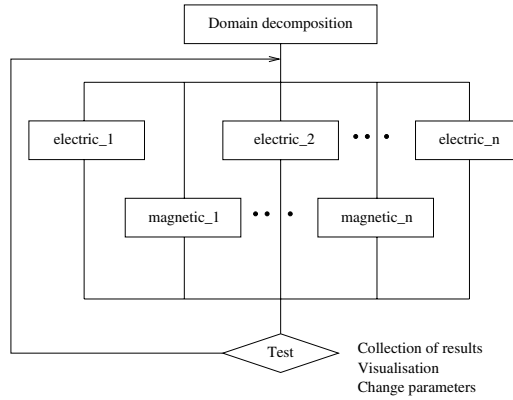

Fig. 2. Task-graph of parallel photonic-crystal simulation

The data layer is represented by the database which stores information related to the status of the distributed system. The scheduler provides task-to-machine mapping and generates a sequence of task executions represented by a schedule. The schedule is passed to the Dispatcher which forwards the tasks to individual Task Launcher objects and handles synchronization and sequencing. On each machine there are Reporter, Task Launcher and application specific objects. All these objects are instantiated by the Object Server. The Reporter provides performance information, i.e., the current amount of available memory, disk space and load information. The Task Launcher handles task execution, interrupts and delivers status information to the Monitor. An important part of the PSE architecture is the Name Server which contains a hierarchy of name and Internet Object Reference (IOR) tuples. This service enables components to be accessed by their name rather than address, this makes application development easier and more transparent. The Application objects represent application specific components described by their IDL specification.

The database is a critical part of the PSE architecture, since it stores all information related to the PSE environment. This information is constantly updated as the computation proceeds. For the database programming the JDBC 2.0 was used which allows database access programmatically, i.e., from the Java code rather than via embedding SQL commands in the code. This provides better portability because it bridges the variations of SQL dialects typical for different DBMS products. The key tables in the database are the Machine and Task ta-

bles. The Machine table contains the following fields: machine name, IP-address, number of processors, clock-speed, memory size, disk size, type of operating system, processor's flop rate, I/O rate, communication speed and current load. The Task table reflects the current status of tasks in the system and also contains information on their predicted resource requirements and measured resource usage in terms of flop-count, memory and disk sizes, I/O and communication volume and flop count.

The information collected by the Monitor is stored in a database and used by the Scheduler for task allocation and load balancing. The interactions of the Monitor with the other components of the system are represented in Fig. 3. On each computer there is an Object Server deployed which instantiates all local objects. The Reporter registers with the Name Server, which maintains a list of remote object addresses. The Monitor queries the Reporter objects at regular intervals and updates the Machine and Task tables in the database.

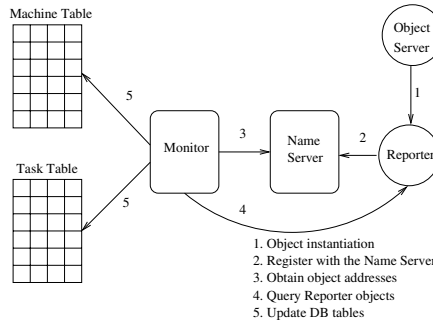

Fig. 3. Monitor's interaction with other modules

The task-to-machine mapping is performed by the scheduler which allows components of the task-graph to run on remote machines in a transparent manner. This operation is based on matching tasks resource requirements with computer parameters. The resource requirements are described by performance models. These were developed by using a characterization technique which involves statistical processing of measurements, identifying the key parameters governing the application's resource requirements and performance, and developing mathematical models. Fig. 4 gives an illustration of performance models of photonic crystal simulations in terms of flop-count, memory and disk usage.

These models are machine independent in that they characterize the resource requirements specific to the given application. The scheduling algorithm is based on the Cartesian product of Machine and Task tables. This operation generates all possible task-to-machine combinations and computes the predicted execution time for each of them. The scheduling algorithm performs the following operations for each task of the task-graph: checks the available memory, disk and load of individual machines, generates a table containing all possible task-to-machine

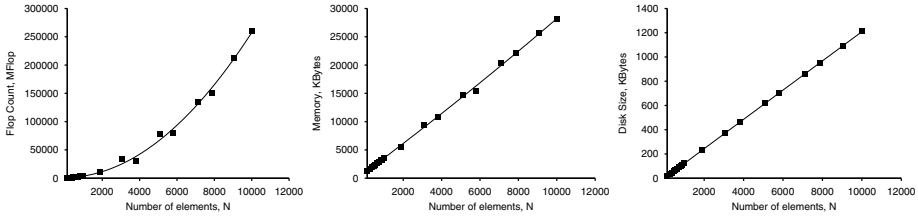

Fig. 4. Resource models for flop-count, memory and disk usage of photonic crystal simulations

mappings,computes the estimated execution time for each combination,generates a schedule by selecting those combinations which minimise the predicted execution time for the whole task-graph.

In the distributed PSE environment where a large number of independent threads are present synchronization is a key issue. In the design we developed a call-back object which is part of the Dispatcher for handling the synchronization between different levels of parallelism and task sequencing. The critical regions of the code in the call-back object were locked in order to prevent corruption of shared data by a simultaneous access of parallel tasks.

3 Simulation of Photonic Crystals

The components developed in the project were used for the construction of an application specific PSE protype. This prototype was used as a parallel computational test-bed for performing various numerical experiments and optimization studies on the example of photonic crystal simulations. The parallel version of the photonic crystal simulation task-graph is presented in Fig. 2. The PSE prototype proved to be an excellent test-bed for performing parametric studies involving changing the material properties and geometry parameters of photonic crystals. Photonic crystals with a band-gap properties have numerous applications for optical computing and for the development of efficient narrow band lasers [12]. These simulations allow optimization of the positioning of air rods in the crystal to achieve band gaps at various frequencies. The measurements of the simulations showed that the complexity of the computations (i.e. flop count) could be approximated by a second order polynomial (see Fig. 4). Substantial performance improvement was achieved by using various approximation methods which significantly increased the speed of computation while preserving the accuracy of results. The prototype PSE was run on a cluster of 13 workstations running the Windows and Linux operating systems. For visualization the Matlab package was used. Fig. 5 gives an illustration of the results by presenting a unit cell of a crystal with an air rod and the density of states for the transverse electric and magnetic components of radiation. Gaps in the density of states indicate frequencies of photons which cannot propagate through the medium.

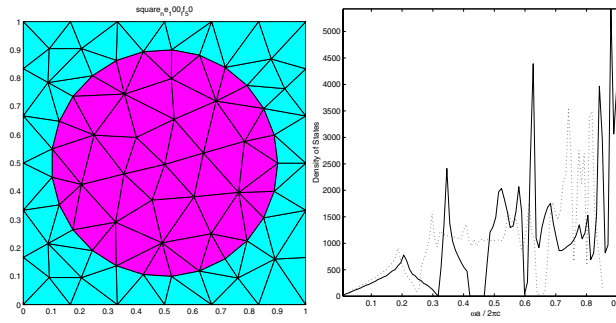

Fig. 5. Unit cell of computation and the distribution of electric and magnetic components of radiation in a photonic crystal

The benefits of the presented PSE can be summarized as follows: this environment integrates all stages of the problem solving process beginning from the task specification to the visualization of results, provides the user with a flexible graphical programming interface which enables to construct various task-graphs, the components developed in the project can be used for the construction of application specific PSEs as it was demonstrated on the example of photonic crystal simulations, the scheduler contains a performance prediction component which provides performance estimation and enables to optimize load balancing on the computing cluster.

4 Conclusions

In this paper the results achieved in a PSE project have been described. The aim of the project was to develop general purpose components such as a user interface, scheduler, monitor, and other components which can be used for rapid prototyping of application specific Problem Solving Environments. The key requirements for this PSE were to provide support for distributed computation in a heterogeneous environment, a user friendly interface for graphical programming, intelligent resource management, object oriented software design, a high level of portability and software maintainability. We conclude the paper with the following remarks:

Although the use of component based middleware technology promises simple distributed application development the reality is that it is not simple to program such systems. The main reason is that the CORBA based middleware technology is still in evolution and numerous features are not mature enough to provide a stable platform for the development of distributed systems. It must be said that the learning curve for CORBA is considerable and a large amount of new knowledge has to be absorbed and utilised in practice. Although programming using CORBA is not a trivial task, nevertheless, this technology is much more suitable for the development of distributed systems than the traditional techniques based on sockets and Remote Procedure Calls.

The utilization of the performance potential offered by a distributed environment is a challenging task. These issues are closely related to scheduling, performance engineering, load balance, ordering of events, dealing with a possible failure of components, etc. These requirements highlight the need for performance models that are relatively simple and easy to use yet are sufficiently accurate in their predictions to be useful as input to a scheduler [3]. At present there is no generally accepted methodology for characterising the performance of applications. We therefore suggest that applications, in addition to specifying interfaces need to incorporate some form of information about the governing parameters determining performance and resource usage. Only with the availability of such performance information will the construction of truly intelligent schedulers become possible. Although the computer science community has been researching performance for a long time, we believe that such research needs to become more systematic and scientific. A common approach to representing performance data together with a methodology that allows independent verification and validation of performance results would be a good start in this direction.

Acknowledgements

This work was sponsored by an EPSRC grant No. GR/M17259/01. We would like to thank to Jacek Generowicz, Ben Hiett, David Lancaster and Tony Scurr for the helpful discussions.

References

1. E.Gallopoulos, E.Houstis and J.R.Rice. Problem Solving Environments for Computational Science, IEEE Comput. Sci. Eng., 1, pp.11-23, 1994.
2. Jacek Marczyk, Principles of Simulation-Based Computer-Aided Engineering FIM Publications, Barcelona, 1999, p.174.
3. N. Floros, K.Meacham, J. Papay, M. Surridge. Predictive Resource Management for Unitary Meta-Applications, Future Generation Computer Systems, 15, 1999, pp.723-734.
4. http://www.cs.purdue.edu/research/cse/gasturbn
5. http://wwwvis.informatik.uni-stuttgart.de/eng/research/proj/autobench/
6. C.Johnson, S.Parker and D.Weinstein. Large Scale Computational Science Applications Using the SCIRun Problem Solving Environment, in Supercomputer 2000.
7. Pat Niemeyer, Jonathan Knudsen, Learning Java 3d revised edition, pp.728, O'Reilly UK, ISBN: 1565927184.
8. CORBA specification by the Object Management Group, http://www.omg.org
9. http://www.iona.com
10. http://www.ooc.com
11. M. S. Shields, O. F. Rana, David W. Walker, Li Maozhen, David Golby. A Java/CORBA based Visual Program Composition Environment for PSEs, Concurrency: Practice and Experience. Volume 12, Issue 8, 2000, pp.687-704.
12. J.M.Generowicz, B.P.Hiett, D.H.Beckett, G.J.Parker, S.J.Cox. Modelling 3 Dimensional Photonic Crystals using Vector Finite Elements. Photonics 2000 (EPSRC, UMIST, Manchester, 4-5 July 2000).