High-Performance Commodity Computing

In this chapter, we consider the role of commodity, off-the-self software technologies and components in the construction of computational grids. We take the position that computational grids can and should build on emerging commodity network computing technologies, such as CORBA, Microsoft's COM, JavaBeans, and less sophisticated Web and networked approaches. These technologies are being used to construct *three-tier* architectures, in which middle-tier *application servers* mediate between sophisticated backend services and potentially simple frontends. The decomposition of application functionality into separate presentation, application, and backend service tiers results in a distributed computing architecture that, we argue, can be extended transparently to incorporate grid-enabled second- and/or third-tier services. Consequently, the three-tier architecture being deployed for commodity network applications is well suited to serve as an architecture for computational grids, combining high performance with the rich functionality of commodity systems. The distinct interface, server, and specialized service implementation layers enable technological advances to be incorporated in an incremental fashion.

This commodity approach to grid architecture should be contrasted with more specialized grid architectures such as Globus (Chapter **??**) and Legion (Chapter **??**). Clearly, these latter technologies can be used to implement lower-tier services. This would seem to be consistent with the Globus design philosophy, given that it defines a service-oriented approach to metacomputing. The integration of Legion into the commodity architecture is more problematic, in that Legion defines a model for the programmer that the three-tier model will not expose. That said, Legion could be encapsulated into backend services as well.

The rest of this chapter proceeds as follows. We first define what we mean by commodity technologies and explain the different ways that they can be used in high-performance computing. Then, we discuss an emerging distributed commodity computing and information system in terms of a conventional three-tier commercial computing model. We describe how this model can be used as a CORBA facility, and we give various examples of how commodity technologies can be used effectively for computational grids. Finally, we discuss how commodity technologies such as Java can be used to build parallel programming environments that combine high functionality and high performance.

1 Commodity Technologies

The past few years have seen an unprecedented level of innovation and progress in commodity technologies. Three areas have been critical in this development: the Web, distributed objects, and databases. Each area has developed impressive and rapidly improving software artifacts. Examples at the lower level include Hypertext Markup Language (HTML), Hypertext Transfer Protocol (HTTP), MIME, Internet Inter-ORB Protocol (IIOP), Common Gateway Interface (CGI), Java, JavaScript, JavaBeans, Common Object Request Broker Architecture (CORBA), Common Object Model (COM), ActiveX, Virtual Reality Markup Language (VRML), object broker ORBs, and dynamic Java servers and clients. Examples at the higher level include collaboration, security, commerce, and multimedia technologies. Perhaps more important than these raw technologies is a set of open interfaces that enable large components to be quickly integrated into new applications.

We believe that computational grid environments can and should be constructed that incorporate these commodity capabilities in such a way as to achieve both high performance and high functionality.

One approach to this goal would be to use just a few of the emerging commodity technologies as "point solutions." For example:

- VRML or Java3D could be used for scientific visualization.
- Web (including Java applets) frontends could provide convenient customizable interoperable user interfaces to high-performance facilities [22].
- The public key security and digital signature infrastructure being developed for electronic commerce could enable more powerful approaches to secure high-performance systems.
- Java could become a common scientific programming language.
- The universal adoption of Java DataBase Connectivity and the growing convenience of Web-linked databases could result in the growing importance of systems that link large-scale commercial databases with high-performance computing resources.
- The emerging "object Web" (linking the Web, distributed objects, and databases) could encourage a growing use of modern object technology.
- Emerging collaboration and other distributed information systems could encourage new distributed work paradigms in place of traditional approaches to collaboration [30].

Our focus, however, is not on such point solutions but on exploiting the overall architecture of commodity systems for high-performance parallel or distributed computing. One might immediately raise the objection that over the past thirty years, many other major broad-based hardware and software developments have occurred—-such as IBM business systems, UNIX, Macintosh and PC desktops, and video games—without any profound impact on high-performance computing software. We believe, however, that the emerging distributed commodity computing and information system (DcciS) is different:

it gives us a worldwide, enterprise-wide distributing computing environment. Whereas previous software revolutions could help *individual components* of a high-performance computing system, DcciS can, in principle, be the backbone of a complete high-performance computing software system—whether it be for some global distributed application, an enterprise cluster, or a tightly coupled large-scale parallel computer.

To achieve this goal, we must add high performance to the emerging DcciS environment. This task may be extremely difficult; but, by using DcciS as a basis, we inherit a multi-billion dollar investment and what is, in many respects, the most powerful productive software environment ever built.

2 The Three-Tier Architecture

Within commodity network computing, the *three-tier* architecture has become pervasive. As shown in Figure 1, the top level of this model provides a customizable client tier, consisting of components such as graphical user interfaces, application programs, and collaboration tools. The middle tier, often referred to as an *application server*, consists of high-level *agents* that can provide application functionality as well as a range of high-level services such as load-balancing, integration of legacy systems, translation services, metering, and monitoring. An important aspect of the middle tier is that it both defines interfaces and provides a control function, coordinating requests across one or more lower-tier servers. The bottom tier provides *backend services*, such as traditional relational and object databases. A set of standard interfaces allows a rich set of custom applications to be built with appropriate client and middleware software. As indicated in Figure 1, these layers can use Web technology such as Java and JavaBeans, distributed objects with CORBA, and standard interfaces such as Java DataBase Connectivity (JDBC).

2.1 Implementing Three-Tier Architectures

To date, two basic technologies have been used to construct commodity three-tier networked computing systems: distributed object systems and distributed service systems. *Distributed object-based systems* = are build on object-oriented technologies such as CORBA, COM, and JavaBeans (when combined with remote method invocation, or RMI). These technologies are discussed in depth in Chapter **??**. Object-oriented systems are ideal for defining middle-tier services. They provide well-defined interfaces and a clean encapsulation of backend services. Through the use of mechanisms such as inheritance, these systems can be easily extended, allowing specialized services to be created.

In spite of the advantages of the distributed object approach, many network services today are provided through a *distributed service* architecture. The most notable example is the *World Wide Web*, in which the middle-tier service is pro-

PICTURE SHOWING 3-TIER MODEL

Figure 1: Industry three-tier view of enterprise = computing

vided by a distributed collection of HTTP servers. Linkage to backend services (databases, simulations, and other custom services) is provided via c-stom programs called via CGI scripts.

The uses of Web technology in *networked databases* provides a good example of how the distributed services architecture is used. Originally, remote access to these databases was provided via a two-tier client-server architecture. In these architectures, sophisticated clients would submit SQL queries to remote databases using proprietary network access protocols to connect the client to the server. The three-tier version of this system might use Web-based forms implemented on a standard *thin client* (i.e., a Web browser) with middle-tier application functionality implemented via CGI scripts in the HTTP server. These scripts then access backend databases by using vendor-specific methods. This scenerio becomes even more attractive with the introduction of Java and JDBC. Using this interface, the middle-tier HTTP service can communicate transparently to a wide range of vendor databases.

Currently, a mixture of distributed service and distributed object architectures is deployed, using CORBA, COM, JavaBean, HTTP servers and CGI scripts, Java servers, databases with specialized network protocols, and other services. These all coexist in a heterogeneous environment with common themes but disparate implementations. We believe that in the near future, there will be a significant convergence of network computing approaches that combines both the distributed server and distributed object approaches [7, 8]. Indeed, we already see a blurring of the distinction between Web and distributed object servers, with Java playing a central role in this process.

On the Web side, we are seeing a trend toward the use of extensible Java-based Web servers. Rather than implementing middle-tier services using CGI scripts, written in a variety of langauges, these servers can be customized on the fly through the use of Java "servlets." Alternatively, CORBA ORBs already exist whose functionality can be = implemented by using Java.

We also believe that these advances will lead to an integrated architecture in which Web-based services (browsers, Java, JavaBeans) = and protocols (HTTP, RMI) are used to construct the top layer, distributed object services and protocols (such as CORBA and IIOP) are used to interface to the lower tier, and the middle tier consists of a distributed set of extensible servers that can process both Web-based and distributed object protocols. The exact technologies used are not critical; however, for the sake of discussion, we will consider a middle tier based on an integrated Java and CORBA server. We believe that the resulting "object-Web" three-tier networked computing architecture will have profound importance and will be the most appropriate way to implement a range of computational grid environments.

2.2 Exploiting the Three-Tier Structure

We believe that the evolving service/object three-tier commodity architecture can and should form the basis for high-performance computational grids. These grids can incorporate (essentially) all of the services of the three-tier architecture outlined above, using its protocols and standards wherever possible, but using specialized techniques to achieve the grid goal of dependable performance. One might achieve this goal by simply porting commodity services to high-performance computing systems. Alternatively, one could continue to use the commodity architecture on current platforms while enhancing specific services to ensure higher performance and to incorporate new capabilities such as high-end visualization (e.g., immersive visualization systems: Chapter **??**) or massively parallel endsystems (Chapter **??**). The advantage of this approach is that it facilitates tracking the rapid evolution of commodity systems, avoiding the need for continued upkeep with each new upgrade of the commodity service. This results in a high-performance commodity computing environment that offers the evolving functionality of commodity systems without requiring significant re-engineering as advances in hardware and software lead to new and better commodity products.

The above discussion indicates that the essential research challenge for high-performance commodity computing is to

> enhance the performance of selected components within a commodity framework in such a way that the performance improvement is preserved through the evolution of the basic commodity technologies.

We believe that the key to achieving this goal is to exploit the three-tier structure by keeping high-performance computing enhancements in the third layer—which is, inevitability, the home of specialized services. This strategy isolates high-

HYBRID SERVER ARCHITECTURE

Figure 2: Today's heterogeneous interoperating hybrid server architecture. High-performance commodity computing involves adding high performance in the third tier.

performance computing issues from the control or interface issues in the middle layer.

Let us briefly consider how this strategy works. Figure 2 shows a hybrid three-tier architecture in which the middle tier is implemented as a distributed network of servers. In general, these servers can be CORBA, COM, or Java-based object-Web servers: any server capable of understanding one of the basic protocols is possible. The middle layer not only includes networked servers with many different capabilities but can = also contain multiple instantiations of the same server to increase performance. The use of high-functionality but modest-performance communication protocols and interfaces at the middle layer limits the performance levels that can be reached. Nevertheless, this first step gives a modest-performance, scalable, parallel (implemented, if necessary, in terms of multiple servers) system that includes all commodity services (such as databases, object services, transaction processing, and collaboratories).

The next step is applied only to those services whose lack of performance constitutes a bottleneck. In this case, an existing backend (third layer) implementation of a commodity service is replaced by its natural high-performance version. For example, sequential databases are replaced by parallel database machines, and sequential or socket-based messaging distributed simulations are replaced by message-passing implementations on low-latency, high-bandwidth dedicated parallel machines (specialized architectures or clusters of workstations).

Note that with the right high-performance software and network connectivity, clusters of workstations (Chapter **??**) could be used at the third layer. Alternatively, collections of middle-tier services could be run on a single paral-

PICTURE OF OBJECT-WEB

Figure 3: Integration of Object Technologies (CORBA) and the Web=3D

lel computer. These various possibilities underscore the fact that the relatively clean architecture of Figure 3 can become confused. In particular, the physical realization may not reflect the logical architecture shown in Figure 2.

3 A High-Performance Facility for CORBA

As discussed above, we envision the middle tier of the network computing architecture providing the interface to high-performance computing capabilities. In the CORBA based strawman that we have presented, this means that high-performance computing components must be integrated into the CORBA architecture.

CORBA is defined in terms of a set of *facilities*, where each facility defines an established, standardized high-level service. Facilities are split those that are required by most applications (called horizontal facilities) and those facilities that are defined to promote interoperability within specific application domains. As CORBA evolves, it is expected that some vertical facilities will migrate to become horizontal facilities.

We believe that high-performance computing can be integrated into the CORBA model by creating a new facility which defines how CORBA objects interact with one another in a high-performance environment. CORBA currently supports only relatively simple computing models, including the embarrassingly parallel activities of transaction processing or dataflow. High-performance commodity computing therefore would fill a gap by providing CORBA's high-performance computing facility.

This new facility allows us to define a commercialization strategy for high-performance computing technologies. Specifically, academia and industry should experiment with high-performance commodity computing as a general framework for providing high-performance CORBA services. Then, one or more industry-led groups should propose high-performance commodity computing specifications, following a process similar to the MPI or HPF forum activities. Such specifications could include another CORBA facility, namely, that involved in user interfaces to (scientific) computers. This facility could comprise interfaces necessary for performance tools and resource managers, file systems, compilation, debugging, and visualization.

We note that we focus here on the use of CORBA, analogies exist in the Java and COM object models. In particular, in Section 6, we discuss how wrappers might be used to provide a Java framework for high-performance computing.

4 High-Performance Communication

Communication performance is a critical aspect of the performance of many high-performance systems. Indeed, a distinguishing feature between distributed and high-performance parallel computation is the bandwidth and latency of communication. In this section, we present an example of how high-performance communication mechanisms can be integrated into a commodity three-tier computing system.

The example we consider is a multidisciplinary simulation. As discussed in Chapter **??**, multidisciplinary applications typically involve the linkage of two or more modules—say, computational fluid dynamics and structures applications—into=20 a single simulation. We can assume that simulation components are individually parallel.

As an initial approach, one could view the linkage between components sequentially, with the middle tier coordinating the movement of data from one simulation component to the other at every step in the simulation. If higher performance is required, one may need to link the components directly, using a high-performance communication mechanism such as the Message Passing Interface (MPI) [**?**]. The connections between modules can be set up by a middle-tier service (such as WebFlow or JavaBeans); then, two third-tier modules can communicate to one another without intervention of the middle-tier service. Control flow returns to the middle tier when the simulation is complete.

A third possibility is to keep the control function within the middle tier, setting up high-performance connections between the two modules and initiating data transfer at every step in the simulation. Unlike the first scenario we considered, the actual transfer of data between modules would take place using the high-performance interface, not through the middle tier. This alternative preserves the advantages of both approaches, using the commodity protocols and services of the three-tier architecture for all user-visible control functions while exploiting the performance of high-performance software only where necessary.

A key element of this example is the structure of the middle-tier service. One approach would be to use JavaBeans (see Chapter **??**) as the vehicle for integrating the individual simulation components. In the JavaBeans model, there is a separation of control (handshake) and implementation. This separation makes it possible to create JavaBeans "lister objects" that reside in the middle tier and act as a bridge between a source and sink of data. The lister object can decide whether high performance is necessary or possible and invoke the specialized high-performance layer. As discussed above, this approach can be used to advantage in runtime compilation and resource management, with execution schedules and control logic in the middle tier and high-performance communication libraries implementing the determined data movement. This approach can also be used to provide parallel I/O and high-performance CORBA.

5 High-Performance Commodity Services

A key feature of high-performance commodity computing is its support for databases, Web servers, and object brokers (see Section 1). In this section, we use the additional example of collaboration services to illustrate the power of the commodity computing approach.

Traditionally, a *collaborative system* is one in which specific capabilities are integrated across two or more clients. Examples of such systems include whiteboards, visualization, and shared control. With the introduction of the flexible Java-based three-tier architecture, support for collaboration can also be integrated into the computing model by providing collaboration services in the middle tier.

Building grid applications on the three-tier architecture provides a well-defined separation between high-performance computing (bottom tier) and collaboration (top and middle tier). Consequently, we can *reuse* collaboration systems built for the general Web market to address areas that require people to be integrated with the computational infrastructure, such as computational steering and collaborative design.

This configuration enables the best commodity technology (e.g., from business or distance education) to be integrated into the high-performance computing environment. Currently, commodity collaboration systems are built on top of the Web and are not yet defined from a general CORBA point of view. Nevertheless, facilities such as WorkFlow [20] are being developed, and we assume that collaboration will emerge as a CORBA capability to manage the sharing and replication of objects.

We note that CORBA is a server-server model in which clients are viewed as servers (i.e., run ORBs) by outside systems. This makes the object-sharing view of collaboration natural, whether an application runs on the "client" (e.g., a shared Microsoft Word document) or the backend tier (e.g., a shared parallel computer simulation).

Two systems, TANGO [30] and WebFlow [29], can be used to illustrate the differences between collaborative and computational sharing. Both systems use

COLLABORATORY PICTURE

Figure 4: Collaboration in today's Java Web Server implementation of the three-tier computing model. Typical clients (top right) are = independent, but Java collaboration systems link multiple clients through object (service) sharing.

Java servers for their middle tier. TANGO provides collaboration services as in Figure 4. Client-side applications are replicated by using an event distribution model. To put a new application into TANGO, one must be able to define both its absolute state and changes therein. By using Java object serialization or similar mechanisms, this state is maintained identically in the linked applications. On the other hand, WebFlow integrates program modules by using a dataflow paradigm. With this system, the module developer defines data input and output interfaces and builds methods to handle data I/O. Typically, there is no need to replicate the state of a module in a WebFlow application.

6 Commodity Parallel Computing

Most of the discussion in this chapter has focused on the use of commodity technologies for computational grids, a field sometimes termed high-performance distributed computing. We believe, however, that commodity technologies can also be used to build parallel computing environments that combine high functionality and high performance. In this section, we first compare alternative views of high-performance distributed parallel computers. Then, we discuss Java as a scientific and engineering programming language.

6.1 High-Performance Commodity Communication

Consider first two views of a parallel computer. In both, various nodes and the host are depicted as separate entities. These represent logically distinct functions, but the physical implementation need not reflect the distinct services. In

PICTURE OF SIMPLE MULTI-TIER VIEW

Figure 5: A parallel computer viewed as a single CORBA object in a classic host-node computing model. Logically, the host is in the middle tier and the nodes in the lower tier. The physical architecture could differ from the logical architecture.

particular, two or more capabilities can be implemented on the same sequential or shared-memory multiprocessor system.

Figure 5 presents a simple multitier view with commodity protocols (HTTP, RMI, COM, or the IIOP pictured) to access the parallel computer as a single entity. This entity (object) delivers high performance by running classic HPCC technologies (such as HPF, PVM, or the pictured MPI) in the third tier. This approach has been successfully implemented by many groups [27] to provide parallel computing systems with important commodity services based on Java and JavaScript client interfaces. Nevertheless, the approach addresses the parallel computer only as a single object and is, in effect, the "host-node" model of parallel programming [31]; the distributed computing support of commodity technologies for parallel programming is not exploited.

Figure 6 depicts the parallel computer as a distributed system with a fast network and integrated architecture. Each node of the parallel computer runs a CORBA ORB (or, perhaps more precisely, a stripped-down ORBlet), Web server, or equivalent commodity server. In this model, commodity protocols can operate both internally and externally to the parallel machine. The result is a powerful environment where one can uniformly address the full range of commodity and high-performance services. Other tools can now be applied to parallel as well as distributed computing.

Obviously, one should be concerned that the flexibility of this second parallel computer is accompanied by a reduction in communication performance. Indeed, most commodity messaging protocols (e.g., RMI, IIOP, and HTTP) have unacceptable performance for most parallel computing applications. How-

PICTURE OF DISTRIBUTED SYSTEM

Figure 6: Each node of a parallel computer instantiated as a CORBA object. The "host" is logically a separate CORBA object but could be instantiated on the same computer as one or more of the nodes. Via a protocol bridge, one could address objects using CORBA with local parallel computing nodes invoking MPI and remote accesses using CORBA where its functionality (access to many services) is valuable.

ever, good performance can be obtained by using a suitable binding of MPI or other high-speed communication library to the commodity protocols.

In Figure 7, we illustrate such an approach to high performance, which uses a separation between messaging interface and implementation. The bridge shown in this figure allows a given invocation syntax to support several messaging services with different performance-functionality tradeoffs. In principle, each service can be accessed by any applicable protocol. For instance, a Web server or database can be accessed by HTTP or CORBA; a network server or distributed computing resource can support HTTP, CORBA, or MPI.

Note that MPI and CORBA can be linked in one of two ways: (1) the MPI function call can call a CORBA stub, or (2) a CORBA invocation can be trapped and replaced by an optimized MPI implementation. Current investigations of a Java MPI linkage have raised questions about extending MPI to handle more general object data types. One could, for instance, extend the MPI communicator field to indicate a preferred protocol implementation, as is done in Nexus [33]. Other research issues focus on efficient object serialization needed for a high-performance implementation of the concept in Figure 7.

6.2 Java and High-Performance Computing

We have thus far discussed many critical uses of Java in both client interfaces and middle-tier servers to high-performance systems. Here, we focus on the

PICTURE OF BRIDGE

Figure 7: A message optimization bridge allows MPI (or equivalently Globus or PVM) and commodity technologies to coexist with a seamless user interface.

direct use of Java as a scientific and engineering programming language [26], taking the role currently played by Fortran 77, Fortran 90, and C++. (In our three-tier architecture, this is the use of Java in lower-tier engineering and science applications or in a CORBA vertical facility designed to support high-performance computing.)

User Base. One of Java's important advantages over other languages is that it will be learned and used by a broad group of users. Java is already being adopted in many entry-level college programming courses and will surely be attractive for teaching in middle or high schools. We believe that entering college students, fresh from their Java classes, will reject Fortran as quite primitive in contrast. C++, as a more complicated systems-building language, may well be a natural progression; but although it is quite heavily used, C++ has limitations as a language for simulation. In particular, it is hard for C++ to achieve good performance even on sequential code. We expect that Java will not have these problems.

Performance. Performance is arguably a critical issue for Java. However, there seems little reason why native Java compilers, as opposed to current portable JavaVM interpreters or "just in time" (JIT) compilers, cannot obtain performance comparable with that of C or Fortran compilers. One difficulty in compiling Java is its rich exception framework, which could restrict compiler optimizations: users would need to avoid complex exception handlers in performance-critical portions of a code. Another important issue with Java is the lack of any operator overloading, which could allow efficient elegant handling of Fortran constructs like COMPLEX. Much debate centers on Java's rule that code not only must run everywhere but must give the same value on all machines. This rule inhibits optimization on machines such as the Intel Pentium that include multiple add instructions with intermediate results stored to higher precision than final values of individual floating-point operations.

An important feature of Java is the lack of pointers. Their absence allows much more optimization for both sequential and parallel codes. Optimistically, one can say that Java shares the object-oriented features of C++ and the performance features of Fortran. An interesting area is the expected performance of Java interpreters (using JIT techniques) and compilers on the Java bytecodes (virtual machine). Currently, a PC just in time compiler shows a factor of 3–10 lower performance than C-compiled code, and this can be expected to decrease to a factor of 2. Hence, with some restrictions on programming style, we expect Java language or VM compilers to be competitive with the best Fortran and C compilers. We also expect a set of high-performance "native-class" libraries to be produced, which can be downloaded and accessed by applets to improve performance in the usual areas one builds scientific libraries.

Parallelism. To discuss parallel Java, we consider four forms of parallelism seen in applications.

1. *Data Parallelism.* By data parallelism we mean large-scale = parallelism found from parallel updates of grid points, particles, and other basic components in scientific computations (see Chapter **??**). Such parallelism is supported in Fortran by either high-level data parallel HPF or, at a lower level, Fortran plus message passing. Java has no built-in parallelism of this type, but the lack of pointers means that natural parallelism is less likely to be obscured. There seems no reason why Java cannot be extended to high-level data-parallel form (HPJava) in a similar way to Fortran (HPF) or C++ (HPC++). Such an extension can be done by using threads on shared-memory machines, while in distributed-memory machines, message passing may be used.

2. *Modest-Grain Functional Parallelism.* Functional parallelism refers to the Functional parallelism arises when... type of parallelism used when computation and I/O operations are overlapped, a situation exploited extensively by Web browsers. This parallelism is built into the Java language with threads but has to be added explicitly with libraries for Fortran and C++.

3. *Object Parallelism.* Object parallelism is quite natural for C++ = or Java. Java can use the applet mechanism to represent objects portably.

4. *Metaproblem Parallelism.* Metaproblem parallelism occurs in applications that are made up of several different subproblems, which themselves may be sequential or parallel.

PICTURE OF JAVA FRONT-END

Figure 8: An architecture for an interpreted Java frontend communicating with a middle-tier server controlling dynamically an HPCC backend

Interpreted Environments. Java and Web technology suggest new programming environments that integrate compiled and interpreted or scripting languages. In Figure 8, we show a system that uses an interpreted Web client interacting dynamically with compiled code through a typical middle-tier server [32]. This system uses an HPF = backend, but the architecture is independent of the backend language. The Java or JavaScript frontend holds proxy objects produced by an HPF frontend operating on the backend code. These proxy objects can be manipulated with interpreted Java or JavaScript commands to request additional processing, visualization, and other interactive computational steering and analysis. We note that for compiled (parallel) Java, the use of objects (as opposed to simple types in the language) probably has unacceptable overhead. However, such objects are appropriate for interpreted frontends, where object references are translated into efficient compiled code. We believe such hybrid architectures are attractive and warrant further research.

Evaluation. In summary, we see that Java has no obvious major disadvantages and some clear advantages compared with C++ and especially Fortran as a basic language for large-scale simulation and modeling. Obviously, we cannot and should not port all our codes to Java. Putting Java (or, more generally, CORBA) wrappers around existing code does, however, seem a good way of preserving old codes. Java wrappers can both document their capability (through the CORBA trader and JavaBean Information services) and allow definition of methods that allow such codes to be naturally incorporated into larger systems. In this way a Java framework for high-performance commodity computing can be used in general computing solutions. As compilers get better, we expect users will find it more and more attractive to use Java for new applications.

Thus, we can expect to see a growing adoption by computational scientists of commodity technology in all aspects of their work.

7 Related Work

The Nile project [14] is developing a CORBA-based distributed-computing solution for the CLEO high-energy physics experiment using a self-managing, fault-tolerant, heterogeneous system of hundreds of commodity workstations, with access to a distributed database in excess of about 100 terabytes. These resources are spread across the United States and Canada at 24 collaborating institutions.

TAO [16] is a high-performance ORB being developed by Douglas Schmidt of Washington University. Schmidt conducts research on high-performance implementations of CORBA [15], geared toward real-time image processing and telemedicine applications on workstation clusters over ATM. TAO, which is based on an optimized version of public domain IIOP implementation from SunSoft, outperforms commercial ORBs by a factor of 2 to 3.

The OASIS (Open Architecture Scientific Information System) [18] environment, being developed by Richard Muntz of UCLA for scientific data analysis, allows one to store, retrieve, analyze, and interpret selected datasets from a large collection of scientific information scattered across heterogeneous computational environments of earth science projects such as EOSDIS. Muntz is exploring the use of CORBA for building large-scale object-based data-mining systems. Several groups are also exploring specialized facilities for CORBA-based distributed computing. Examples include the Workflow Management Coalition [20] and Distributed Simulations [21].

8 Summary

We have described the three-tier architecture employed in commodity computing and also reviewed a number of the commodity technologies that are used in its implementation. We have argued that the resulting separation of concerns among interface, control, and implementation may well make the integration of high-performance capabilities quite natural. We have also sketched a path by with this integration may be achieved. We believe that while=20 significant challenges must be overcome before commodity technologies can guarantee the performance required for computational grids, there is much to be gained from structuring approaches to grid architectures in terms of this framework.

9 Further Reading

The three-tier architecture is reviewed in an article in *Byte* [23].

[We need a good set of 5 review articles and books.]

References

[1] Object Management Group, http://www.omg.org .

[2] CORBA Component Model RFP, http://www.omg.org/library/schedule/CORBA Component Model RFP.htm.

[3] IBM, Netscape, Oracle, and SunSoft, CORBA Component Imperatives, http://www.omg.org/news/610pos.htm.

[4] Dale Rogerson, *Inside COM - Microsoft's Component Object Model*, Microsoft Press, 1997.

[5] JavaBeans, http://www.javasoft.com/beans/

[6] Robert Englander, *Developing JavaBeans,* O'Reilly & Associates, 1997.

[7] CORBA 2.0/IIOP Specification, http://www.omg.org/corba/c2indx.htm.

[8] Robert Orfali and Dan Harkey, *Client/Server Programming with Java and CORBA,* Wiley, 1997.

[9] OrbixWeb for Java from IONA, http://www.iona.com.

[10] VisiBroker for Java from Visigenic, http://www.visigenic.com.

[11] JacORB by Freie Universit=3DE4t Berlin, http://www.inf.fu-berlin.de/ brose/jacorb/.

[12] omniORB2 by Olivetti and Oracle Research Laboratory http://www.orl.co.uk/omniORB/omniORB.html.

[13] The Electra Object Request Broker, http://www.olsen.ch/∼maffeis/electra.html.

[14] Nile: National Challenge Computing Project http://www.nile.utexas.edu/.

[15] Douglas Schmidt, Research on High Performance and Real-Time CORBA, http://www.cs.wustl.edu/∼schmidt/corba-research-overview.html.

[16] Douglas Schmidt, *Real-time CORBA with TAO (The ACE ORB),* http://www.cs.wustl.edu/∼schmidt/TAO.html.

[17] Steve Vinoski, *Object Interconnections,* column in C++ Report, http://www.iona.com/hyplan/vinoski/.

[18] E. Mesrobian, R. Muntz, E. Shek, S. Nittel, M. LaRouche, and M. Krieger, *OASIS: An Open Architecture Scientific Information = System,* 6th International Workshop on Research Issues in Data Engineering, New Orleans, February, 1996. See also http://techinfo.jpl.nasa.gov/JPLTRS/SISN/ISSUE36/MUNTZ.htm.

[19] WORB - Web Object Request Broker, http://osprey7.npac.syr.edu:1998/iwt98/projects/worb.

[20] Workflow Mangement Coalition, http://www.aiai.ed.ac.uk/project/wfmc/.

[21] DoD Modeling and Simulation Office (DMSO), High Level Architecture and Run-Time Infrastructure, http://www.dmso.mil/hla.

[22] Geoffrey Fox, "Introduction to Web Technologies and Their Applications," Syracuse report SCCS-790. http://www.npac.syr.edu/techreports/html/0750/abs-0790.html.

[23] Three-tier commercial computing model, *Byte*, August 1997, http://www.byte.com/art/9708/sec5/art1.htm.

[24] Mark Baker Portsmouth, Collection of links relevant to HPcc Horizontal CORBA facility and seamless interfaces to HPCC computers, http://www.sis.port.ac.uk/ mab/Computing-FrameWork/.

[25] Compilation of references to use of Java in computational science and engineering (including proceedings of Syracuse and Las Vegas meetings), http://www.npac.syr.edu/projects/javaforcse.

[26] Geoffrey Fox and Wojtek Furmanski, Java for parallel computing and as a general language for scientific and engineering simulation and modeling, *Concurrency: Practice and Experience,* 9: 415–426, 1997.

[27] Jim Almond, Resource for seamless computing, http://www.ecmwf.int/html/seamless/.

[28] David Bernholdt, Geoffrey Fox, and Wojtek Furmanski, Towards High Performance Object Web Based FMS, whitepaper for ARL MSRC PET Program, Sept. 1997.

[29] D. Bhatia, V. Burzevski, M. Camuseva, G. Fox, W. Furmanski, and G. Premchandran, WebFlow – A visual programming paradigm for Web/Java based coarse grain distributed computing, *Concurrency: Practice = and Experience,* 9: 555–578, 1997.

[30] L. Beca, G. Cheng, G. Fox, T. Jurga, K. Olszewski, M. Podgorny, P. Sokolowski, and K. Walczak, Java enabling collaborative education, health care and computing, *Concurrency: Practice and Experience,* 9: 521–534, 1997. See alsohttp://trurl.npac.syr.edu/tango/.

[31] K. Dincer and G. Fox, Using Java and JavaScript in the Virtual Programming Laboratory: A web-based parallel programming environment, *Concurrency: Practice and Experience,* 9: 521–534, 1997; see = also http://www.npac.syr.edu/users/dincer/papers/vpl/.

[32] E. Akarsu and T. Haupt, Integrated Environment for HPF Compiler and Interpreter, http://www.npac.syr.edu/projects/hpfi/.

[33] I. Foster, J. Geisler, C. Kesselman, and S. Tuecke, Managing multiple communication methods in high-performance networked computing systems, *J. Parallel and Distributed Computing*, 40:35–48, 1997.