Parallel and Distributed Computing using Pervasive Web and Object Technologies

Geoffrey C. Fox, Wojtek Furmanski

mailto:gcf@npac.syr.edu , furm@npac.syr.edu

NPAC, Syracuse University
111 College Place, Syracuse, NY 13244-4100

World Wide Web: http://www.npac.syr.edu/users/gcf/HPcc/HPcc.html

Abstract

We review the growing power and capability of commodity computing and communication technologies largely driven by commercial distributed information systems. These systems are built from CORBA, Microsoft's COM, Javabeans, and less sophisticated web and networked approaches. One can abstract these to a three-tier model with largely independent clients connected to a distributed network of servers. The latter host various services including object and relational databases and, of course, parallel and sequential computing. High performance can be obtained by combining concurrency at the middle-server tier with optimized parallel back-end services. The resultant system combines the needed performance for large-scale HPCC applications with the rich functionality of commodity systems. Further, the architecture with distinct interface, server and specialized service implementation layers, naturally allows advances in each area to be easily incorporated. We show that this approach can be applied to both metacomputing and to provide improved parallel programming environments. We describe exploitation issues within a CORBA context and illustrate how performance can be obtained within a commodity architecture. Examples are given from collaborative systems, support of multidisciplinary interactions, proposed visual HPCC ComponentWare, distributed simulation, and the use of Java in high-performance computing.

1. INTRODUCTION

We believe that industry and the loosely organized worldwide collection of (freeware) programmers is developing a remarkable new software environment of unprecedented quality and functionality. We call this DcciS—Distributed commodity computing and information System. We believe that this can benefit HPCC in several ways and allow the development of both more powerful parallel programming environments and new distributed metacomputing systems. In the second section, we define what we mean by commodity technologies and explain the different ways that they can be used in HPCC. In the third and critical section, we define an emerging architecture of DcciS in terms of a conventional three-tier commercial computing model. The next section describes the expected steps in the CORBA model for establishing HPcc as a community framework and CORBA facility. In this and related papers [1], we discuss several examples and the critical research issue: can high performance systems—called HPcc or High Performance Commodity Computing—be built on top of DcciS. Examples include integration of collaboration into HPcc; the natural synergy of distribution simulation and the HLA standard with our architecture [2]; and the step from object to visual component based programming [3] in parallel and distributed computing. Finally we discuss the use of DcciS to build parallel programming environments and HPJava—the use of Java on parallel or sequential machines for high performance computing [4,5].

2. COMMODITY TECHNOLOGIES AND THEIR USE IN HPCC

The last three years have seen an unprecedented level of innovation and progress in commodity technologies driven largely by the new capabilities and business opportunities of the evolving worldwide network. The web is not just a document access system supported by the somewhat limited HTTP protocol [6]. Rather, it is the distributed object technology which can build general multi-tiered enterprise intranet and internet applications. CORBA is turning from a sleepy heavyweight standards initiative to a major competitive development activity that battles with COM and Javabeans to be the core distributed object technology.

There are many driving forces and many aspects to DcciS but we suggest that the three critical technology areas are the web, distributed objects, and databases. These are being linked and we see them subsumed in the next generation of "object-web" technologies, which is illustrated by the recent Netscape and Microsoft Version 4 browsers. Databases are older technologies, but their linkage to the web and distributed objects is transforming their use and making them more widely applicable.

In each commodity technology area, we have impressive and rapidly improving software artifacts. As examples, we have at the lower level the collection of standards and tools such as HTML, HTTP, MIME, IIOP, CGI, Java, JavaScript, Javabeans, CORBA, COM, ActiveX, VRML, new powerful object brokers (ORB's), dynamic Java servers, and clients including applets and servlets [6,7]. At a higher level collaboration, security, commerce, multimedia and other applications/services are rapidly developing using standard interfaces or frameworks and facilities. This emphasizes that equally and perhaps more importantly than raw technologies, we have a set of open interfaces enabling distributed modular software development. These interfaces are at both low and high levels and the latter generate a very powerful software environment in which large preexisting components can be quickly integrated into new applications. We believe that there are significant incentives to build HPCC environments in a way that naturally inherits all the commodity capabilities so that HPCC applications can also benefit from the impressive productivity of commodity systems. NPAC's HPcc activity is designed to demonstrate that this is possible and useful so that one can achieve simultaneously both high performance and the functionality of commodity systems.

Note that commodity technologies can be used in several ways. This article concentrates on exploiting the natural architecture of commodity systems but more simply, one could just use a few of them as "point solutions". This we can term a "tactical implication" of the set of the emerging commodity technologies and illustrate below with some examples:

- Perhaps VRML or Java3D are important for scientific visualization
- Web (including Java applets) front-ends provide convenient customizable interoperable user interfaces to HPCC facilities [8,9,10]
- Perhaps the public key security and digital signature infrastructure being developed for electronic commerce, could enable more powerful approaches to secure HPCC systems
- Perhaps Java will become a common scientific programming language and so effort now devoted to Fortran and C++ tools needs to be extended or shifted to Java
- One can harness the immense compute power of web clients to perform large-scale computation as illustrated by the Javelin system from UCSB [11]. This is very important but different from the Web Server approach to computing described in this paper.
- The universal adoption of JDBC (Java Database Connectivity) and the growing convenience of web-linked databases could imply a growing importance of systems that link large-scale commercial databases with HPCC computing resources
- Javabeans, RMI, COM, CORBA, IIOP form the basis of the emerging "object web" which analogously to the previous bullet could encourage a growing use of modern object technology
- Emerging collaboration and other distributed information systems could allow new distributed work paradigms that could change the traditional teaming models in favor of those, for instance, implied by the new NSF Partnerships in Advanced Computation

However, probably more important is the strategic implication of DcciS, which implies certain critical characteristics of the overall architecture for a high-performance parallel or distributed computing system. First, we note that we have seen over the last 30 years many other major broad-based hardware and software developments—such as IBM business systems, UNIX, Macintosh/PC desktops, video games—but these have not had profound impact on HPCC software. However, we suggest the DcciS is different for it gives us a world-wide/enterprise-wide distributing computing environment. Previous software revolutions could help individual components of a HPCC software system, but DcciS can in principle be the backbone of a complete HPCC software system—whether it be for some global distributed application, an enterprise cluster or a tightly coupled large scale parallel computer. In a nutshell, we suggest that "all we need to do" is to add "high performance" (as measured by bandwidth and latency) to the emerging commercial concurrent DcciS systems. This "all we need to do" may be very hard, but by using DcciS as a basis we inherit a multi-billion dollar investment, and what in many respects is the most powerful productive software environment ever built. Thus, we should look carefully into the design of any HPCC system to see how it can leverage this commercial environment.

3. THREE TIER HIGH-PERFORMANCE COMMODITY COMPUTING

We start with a common modern industry view of commodity computing with the three tiers shown in Figure 1 [12]. Here, we have customizable client and middle tier systems accessing "traditional" back end services such as 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 on figure, both these two layers can use web technology such as Java and Javabeans, distributed objects with CORBA and standard interfaces such as JDBC (Java Database Connectivity). There are, of course, no rigid solutions and one can get "traditional" client server solutions by collapsing two of the layers together. For instance, with database access, one gets a two-tier solution by either incorporating custom code into the "thick" client or in analogy to Oracle's PL/SQL, compile the customized database access code for better performance, and incorporate the compiled code with the back-end server. The latter like the general three-tier solution, supports "thin" clients such as the currently popular network computer.

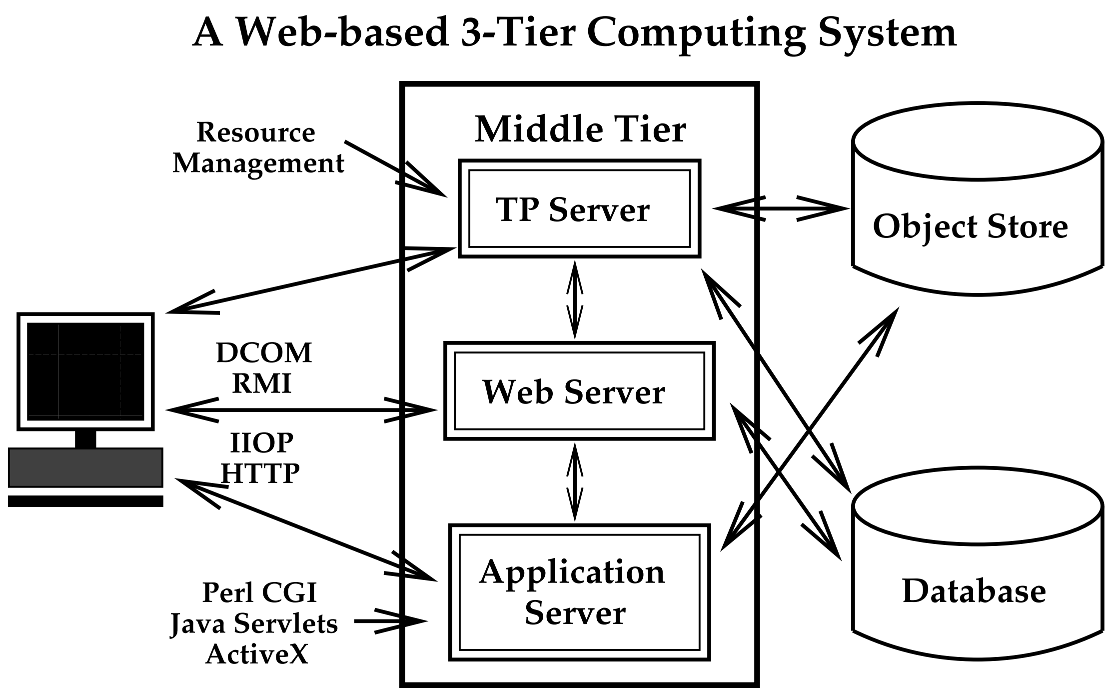

Figure 1: Industry 3-tier View of Enterprise Computing

The commercial architecture is evolving rapidly and is exploring several approaches that co-exist in today's (and any realistic future) distributed information system. The most powerful solutions involve distributed objects. There are three important commercial object systems—CORBA, COM and Javabeans. These have similar approaches, and it is not clear if the future holds a single such approach or a set of interoperable standards. CORBA is a distributed object standard managed by the OMG (Object Management Group) comprised of 700 companies. COM is Microsoft's distributed object technology initially aimed at Window machines. Javabeans (augmented with RMI and other Java 1.1 features) is the "pure Java" solution—cross platform but unlike CORBA, not cross-language! Legion is an example of a major HPCC focused distributed object approach; currently it is not built on top of one of the three major commercial standards. The HLA/RTI standard for distributed simulations in the forces modeling community is another important domain specific distributed object system. It appears to be moving to integration with CORBA standards. Although a distributed object approach is attractive, most network services today are provided in a more ad-hoc fashion. In particular, today's web uses a "distributed service" architecture with HTTP middle-tier servers invoking via the CGI mechanism, C and Perl programs linking to databases, simulations or other custom services. There is a trend toward the use of Java servers with the servlet mechanism for the services. This is certainly object based, but does not necessarily implement the standards implied by CORBA, COM or Javabeans. However, this illustrates an important evolution as the web absorbs object technology with the evolution:

HTTP --> Java Sockets --> IIOP or RMI

(Low Level network standard) (High level network standard)

Perl CGI Script --> Java Program --> Javabean distributed object.

As an example, consider the evolution of networked databases. Originally these were client-server with a proprietary network access protocol. Web linked databases produced a three-tier distributed service model with an HTTP server using a CGI program (running Perl for instance) to access the database at the backend. Today we can build databases as distributed objects with a middle-tier Javabean using JDBC to access the backend database. Thus, a conventional database is naturally evolving to the concept of managed persistent objects.

Today, as shown in Figure 2, we see a mixture of distributed service and distributed object architectures. CORBA, COM, Javabean, HTTP Server + CGI, Java Server and servlets, databases with specialized network accesses, and other services co-exist in the heterogeneous environment with common themes but disparate implementations. We believe that there will be significant convergence as a more uniform architecture is in everyone's best interest.

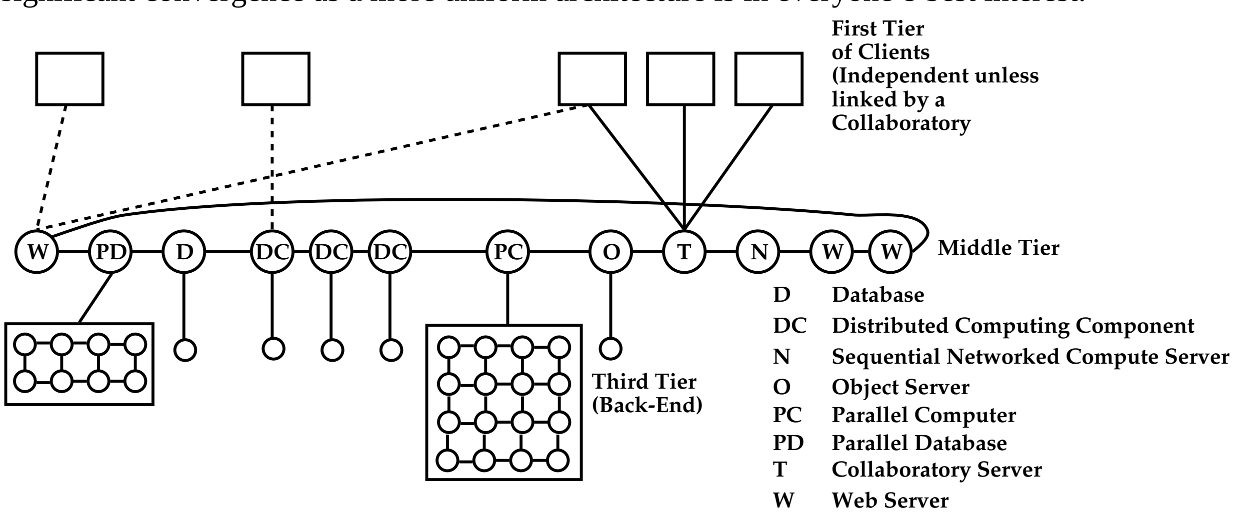

Figure 2: Today's Heterogeneous Interoperating Hybrid Server Architecture. HPcc involves adding to this system, high performance in the third tier.

We also believe that the resultant architecture will be integrated with the web so that the latter will exhibit a distributed object architecture. More generally, the emergence of IIOP (Internet

Inter-ORB Protocol), CORBA2, and the realization that CORBA is naturally synergistic with Java is starting a new wave of "Object Web" developments that could have profound importance. Java is not only a good language to build brokers, but also Java objects are the natural inhabitants of object databases. The resultant architecture gives a small object broker (a so-called ORBlet) in each browser as in Netscape's current plans. Most of our remarks are valid for all these approaches to a distributed set of services. Our ideas are, however, easiest to understand if one assumes an underlying architecture that is a CORBA or Javabean distributed object model integrated with the web.

We wish to use this service/object evolving three-tier commodity architecture as the basis of our HPcc environment. We need to naturally incorporate (essentially) all services of the commodity web, and to use its protocols and standards wherever possible. We insist on adopting the architecture of commodity distribution systems as complex HPCC problems require the rich range of services offered by the broader community systems. Perhaps we could "port" commodity services to a custom HPCC system, but this would require continued upkeep with each new upgrade of the commodity service. By adopting the architecture of the commodity systems, we make it easier to track their rapid evolution and expect it will give high functionality HPCC systems, which will naturally track the evolving Web/distributed object worlds. This requires us to enhance certain services to get higher performance and to incorporate new capabilities such as high-end visualization (e.g. CAVE's) or massively parallel systems where needed. This is the essential research challenge for HPcc for we must not only enhance performance where needed, but do it in a way that is preserved as we evolve the basic commodity systems. We certainly have not demonstrated clearly that this is possible, but we have a simple strategy that we will elaborate in [1] and Section 5. Thus, we exploit the three-tier structure and keep HPCC enhancements in the third tier, which is inevitably the home of specialized services in the object-web architecture. This strategy isolates HPCC issues from the control or interface issues in the middle layer. If successful, we will build an HPcc environment that offers the evolving functionality of commodity systems without significant re-engineering as advances in hardware and software lead to new and better commodity products.

Returning to Figure 2, we see that it elaborates Figure 1 in two natural ways. Firstly, the middle tier is promoted to a distributed network of servers; in the "purest" model these are CORBA/ COM/ Javabean object-web servers (as shown in Figure 3), but obviously any protocol compatible server is possible. This middle-tier layer includes not only networked servers with many different capabilities (increasing functionality) but also multiple servers to increase performance on a given service. The use of high functionality but modest performance communication protocols and interfaces at the middle tier limits the performance levels that can be reached in this fashion. However, this first step gives a modest performance scaling, parallel (implemented if necessary, in terms of multiple servers) HPcc system that includes all commodity services such as databases, object services, transaction processing and collaboratories. The next step is only applied to those services with insufficient performance. Naively we "just" replace an existing back-end (third tier) implementation of a commodity service by its natural HPCC high-performance version. Sequential or socket-based messaging distributed simulations are replaced by MPI (or equivalent) implementations, as in Section 7, on low-latency high-bandwidth dedicated parallel machines. These could be specialized architectures or "just" clusters of workstations. Note that with the right high-performance software and network connectivity, workstations can be used at tier three just as the popular "LAN" consolidation" use of parallel machines like the IBM SP-2, corresponds to using parallel computers in the middle tier. Further, a "middle tier" compute or database server could, of course, deliver its services using the same or different machine from the server. These caveats illustrate that, as with many concepts, there will be times when the relatively clean architecture of Figure 2 will become confused and in particular, the physical realization does not necessarily reflect the logical architecture shown in the figure.

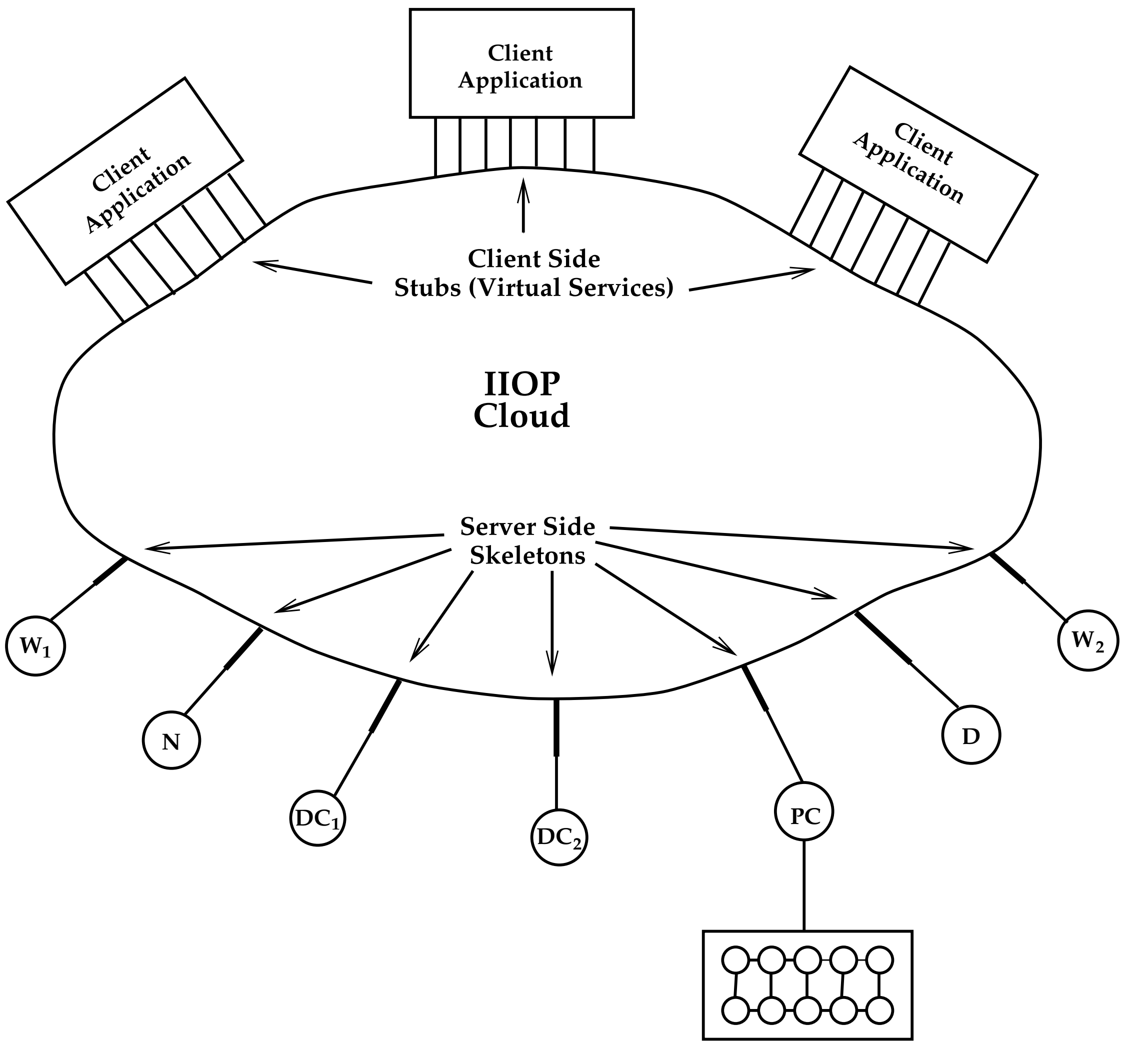

Figure 3: Pure CORBA architecture for the heterogeneous DcciS services of Figure 2. There is a similar Java version of this using RMI and JDBC with of course the linked application being restricted to Java code. CORBA and the analogous COM solution are cross-language solutions.

4. HPcc AS A CORBA FACILITY

CORBA is defined as a suite of software layers with the architecture illustrated in Figure 4.

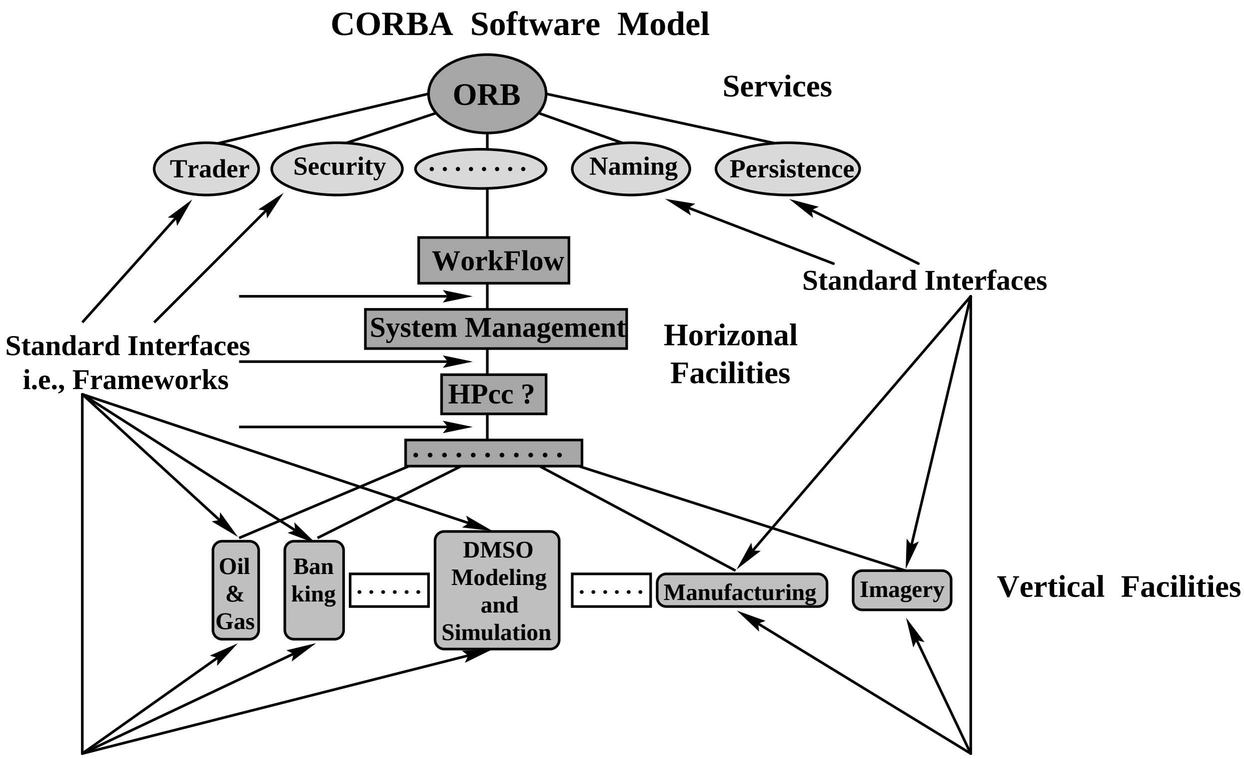

Figure 4: Software Layers in CORBA

We see (currently 15) basic services, such as naming and persistence layered below a set of general capabilities or horizontal facilities in the ComponentWare jargon. We suggest that HPcc is naturally placed here as it is designed to provide high performance to essentially any application area. The latter are seen as vertical or specialized facilities in Figure 4 that provide domain-specific distributed object support. Here, we see mainstream commercial applications such as manufacturing, banking and mapping. The vertical and horizontal facilities are associated with frameworks that are the universal interfaces with which they are linked together and to user programs (or objects).

Note that CORBA currently supports only relatively simple computing models including "embarrassingly parallel" as in transaction processing or dataflow as in the CORBA workflow facility. The modeling and simulation community appears likely to evolve their HLA standard to a new vertical CORBA facility. HPcc, therefore, fills a gap and is defined as the HPCC (here we are using capital C's) CORBA horizontal facility. In the following paragraph, we point out that this allows us to define a commercialization strategy for high-performance computing technologies.

Academia and Industry should now experiment with the design and implementations of HPcc as a general framework for providing high-performance CORBA services. Then one or more industry-led groups proposes HPcc specifications as a new horizontal facility. A process similar to the MPI or HPF forum activities leads to consensus and the definition of an HPcc facility standard. This then allows industry (and academia) to compete in the implementation of HPcc within an agreed interoperable standard. HPcc can either include or be accompanied by another CORBA facility—namely, that involved in user interfaces to (scientific) computers. This would include interfaces necessary for performance tools and resource managers, as well as file systems, compilation, debugging and visualization [8]. Such a seamless interface was the subject of the *Birds of the Feather* session at SC97 and a recent workshop in England [13].

The remarks we make above for CORBA have analogies in the Java and COM object models. In particular, CORBA facilities are logically equivalent to the different frameworks being developed for Java. HPcc would become a Java framework for high-performance computing and used in 100% Java solutions—something that is quite realistic as we show in Section 7.2, that Java is a potentially excellent language for scientific computing. Of course, not all code will be written in Java and it is unlikely to be wise to convert existing Fortran and C++ to Java. However, putting Java (or more generally CORBA) wrappers around existing code seems to us a good way of preserving old codes. This 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 HPcc can be used in general computing solutions. In particular, the CORBA seamless interfaces discussed above are likely to be built in a 100% Java environment anyway, as the natural implementation is a client side [9,10] Java Applet linking to a Java Web server acting as the proxy for the high-performance computer. Finally, we note that the discussion in Section 7.2 of HPJava—the collection of libraries, compilers, coding practices and environments to support high-performance scientific computing naturally form a Java framework for Scientific Computing. Alternatively, high-performance scientific computing is another CORBA facility that is more specialized than HPcc and, hence, becomes perhaps a vertical facility.

5. TYPICAL APPLICATIONS OF HPcc

5.1 Overview

The essential idea behind HPcc is to build systems that preserve the basic tier-2 commodity server layer "untouched". We add to this at tier 1, customized user interfaces and client side analysis and visualization systems. The specialized services such as massively parallel systems are in tier 3 with suitable customized links to the tier-2 servers. As explained today's commodity tier-2 architecture is very confusing but a clean model for it is given by CORBA and it is quite likely that this will become the standard.

CORBA offers a powerful organization framework whose utility can already be seen in HPCC applications like image processing and data analysis. The latter is illustrated by the Nile project [14] where the "embarrassingly parallel" high-energy physics data analysis certainly requires large scale computing. As there are no significant communication needs between the basic computational steps, this can be efficiently implemented within today's modest performance commodity frameworks.

Application of HPcc to general high performance applications requires one to address latency and bandwidth of communication between components of the system. One strategy for achieving this is illustrated in the following two subsections. We need to retain the high functionality of tier-2 services and achieve this by leaving the "overall" protocol (control) processing at tier 2 while enabling high performance data transfer at tier 3. Our technical approach to this is described in [1] and we control message passing using Java events. These, in the current Java 1.1 AWT (abstract windowing toolkit) release, do separate event preparation, notification and processing in terms of separate event sources, listeners, and observers (or sinks for data). With tier 2 as a set of CORBA servers, we call this strategy HP-CORBA as it integrates high performance into CORBA.

In Section 5.2, we contrast our multi-tier messaging model with other HPCC approaches for a simple multi-disciplinary application. In sec. 5.3, we describe a particular HPCC programming environment WebFlow developed at NPAC.

5.2 HPcc for Multidisciplinary Applications

Here we illustrate the commodity technology strategy with a simple multidisciplinary application involving the linkage of two modules A and B—say CFD, and structures applications respectively. Let us assume both are individually parallel, but we need to link them. One could view the linkage sequentially as in Figure 5(a), but often one needs higher performance and one would "escape" totally into a layer which linked decomposed components of A and B with high-performance MPI (Nexus or PVMPI). Here, we view MPI as the "machine language" of the higher-level commodity communication model given by approaches such as WebFlow described in Section 5.3.

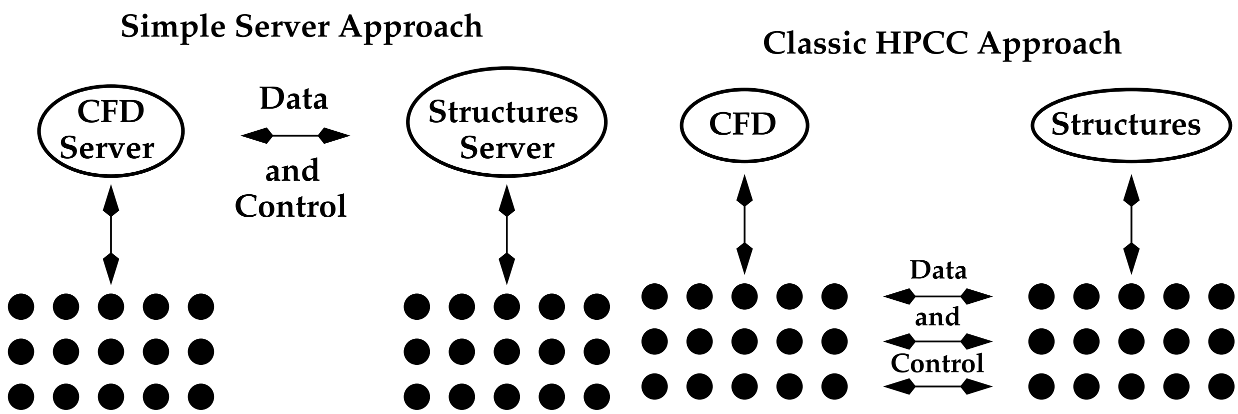

Figure 5(a): Simple sequential server approach to Linking Two Modules, and 5(b): Full HPCC approach to Linking Two Modules

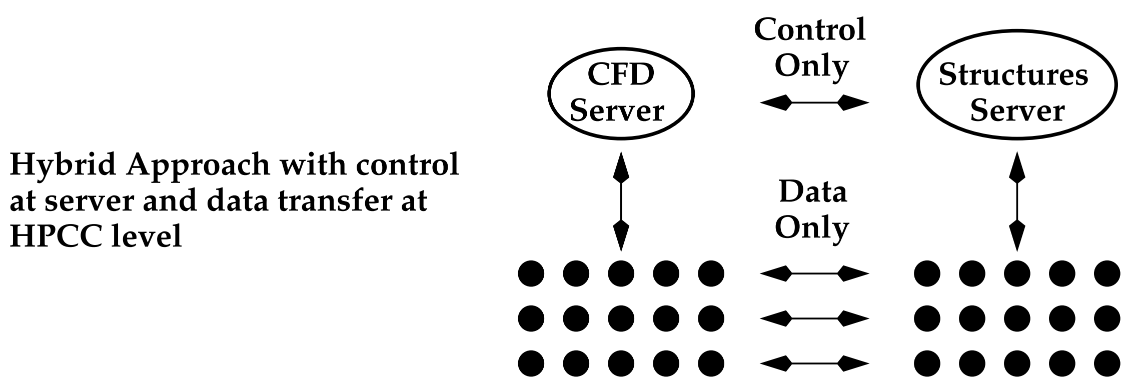

Figure 5(c): Hybrid approach to Linking Two Modules

There is the "pure" HPCC approach of Figure 5(b), which replaces all commodity web communication with HPCC technology. However, as described in Section 5.1, we note that there is a middle ground between the implementations of Figures 5(a) and 5(b) where one keeps control (initialization, etc.) at the server level and "only" invokes the high-performance back end for the actual data transmission. This is shown in Figure 5(c) and appears to obtain the advantages of both commodity and HPCC approaches, for we have the functionality of the Web and where necessary the performance of HPCC software. As we wish to preserve the commodity architecture as the baseline, this strategy implies that one can confine HPCC software development to providing high-performance data transmission with all of the complex control and service provision capability inherited naturally from the Web.

5.3 WebFlow HPcc Programming Environment

NPAC has developed WebFlow [3,15] as a web-based visual programming environment illustrated in Figure 6.

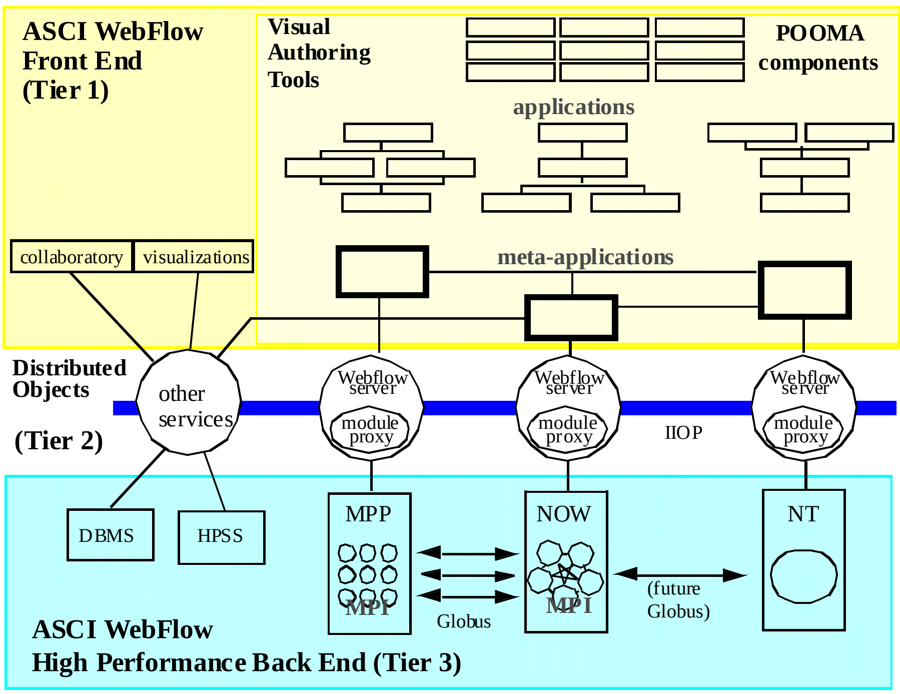

Figure 6: Integration of WebFlow [3] with Globus [16] in a 3-tier HPcc computing environment.

WebFlow supports the dataflow visual programming model popularized by such commercial systems as AVS [17] and Khoros [18] and research projects like CODE [19] and HeNCE [20]. The current version of WebFlow supports a simple two-tier computing model with a mesh of Java Servers at the second tier. These use servlets as compute modules and link them as in Figure 5(a) at the tier-2 level. WebFlow is one of only the few available metacomputing programming environments and can naturally link heterogeneous platforms including UNIX, Windows NT and Windows 95 systems as these all can run Java servers. Using the approach of Figures 2 and 9, one can include parallel computers in the current set up if the tier-2 servlet uses Java's native method interface and acts as a "host" to a standard HPCC simulation at tier 3. However this is not a complete metacomputing system as it does not allow high performance links between different tier-3 components as is illustrated in Figure 5(c) above. Figure 6 shows how one can use Globus (or its messaging subsystem Nexus) to provide the high-performance tier-3 links. Note how the hybrid approach of Figure 6 does realize a combination of high functionality commodity with specialized tier-3 services. Currently, Globus provides tier-3 high-performance services between UNIX systems while WebFlow provides universal but modest performance tier-2 linkage including databases and Windows machines. As Globus or other tier-3 capabilities are extended (for instance to Windows NT), one can replace more and more of the tier-2 links by their high performance versions. However HPcc always gives you a complete system and Globus is “only” needed to improve performance and it is not required to implement the system. Further by building our programming environment on top of tier-2 servers, we both track improvements of commodity systems and allow a uniform programming environment independent of the implementation details at tier 3. As described in Section 7, one can apply these concepts to either metacomputing (high performance distributed computing) or parallel processing. The above argument shows that HPcc can provide a uniform machine independent parallel processing environmentr.

6. COMMODITY SERVICES IN HPcc

We have already stressed that a key feature of HPcc is its support of the natural inclusion into the environment of commodity services such as databases, web servers and object brokers. Here, we give some further examples of commodity services that illustrate the power of the HPcc approach.

6.1. Distributed Collaboration Mechanisms

The current Java Server model for the middle tier naturally allows one to integrate collaboration into the computing model, and our approach allows one to "re-use" collaboration systems built for the general Web market. Thus, one can without any special HPCC development, address areas such as computational steering and collaborative design, which require people to be integrated with the computational infrastructure. In Figure 7, we define collaborative systems as integrating client side capabilities together. In steering, these are people with analysis and visualization software. In engineering design, one would also link design (such as CATIA or AutoCAD) and planning tools. In both cases, one would need the base collaboration tools such as white-boards, chat rooms and audio-video conferencing.

If we are correct in viewing collaboration as sharing of services between clients, the three-tier model naturally separates HPCC and collaboration, and allows us to integrate into the HPCC environment the very best commodity technology, which is likely to come from larger fields such as business or (distance) education. Currently, commodity collaboration systems are built on top of the Web and although emerging facilities, such as Work Flow, imply approaches to collaboration are not yet defined from a general CORBA point of view. We assume that collaboration is sufficiently important that it will emerge as a CORBA capability to manage the sharing and replication of objects. Note that CORBA is a server-server model and "clients" are viewed as servers (i.e., run Orb's) by outside systems. This makes the object-sharing view of collaboration natural whether the application runs on "client" (e.g., shared Microsoft Word document) or on back-end tier, as in the case of a shared parallel computer simulation.

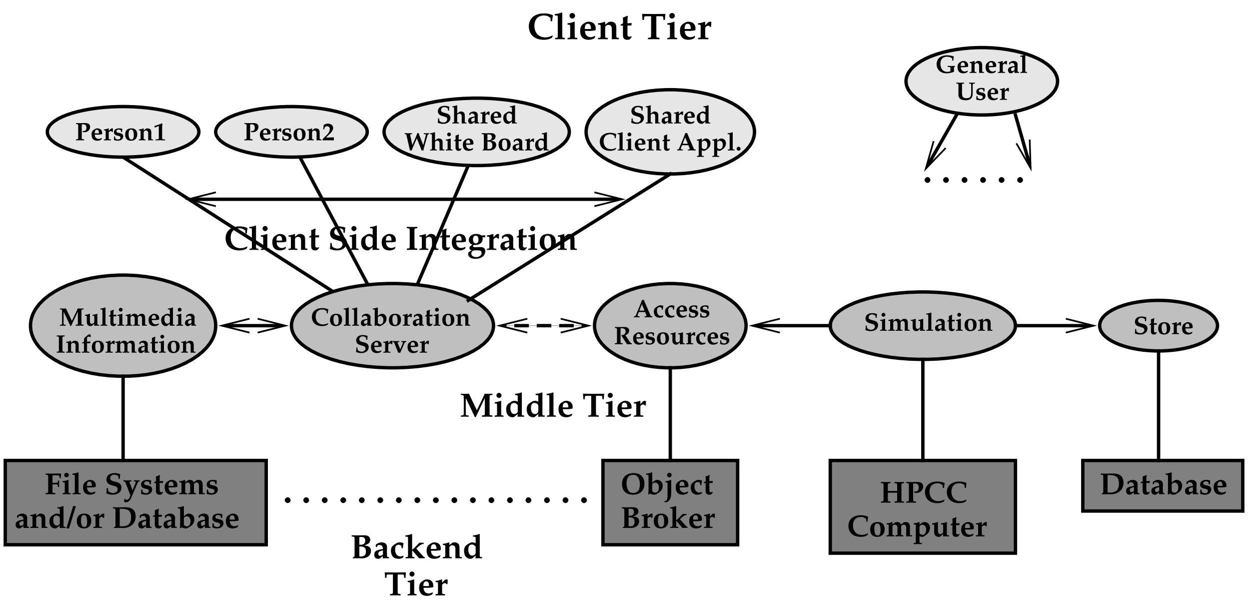

Figure 7: Collaboration in today's Java Web Server implementation of the three-tier computing model. Typical clients (on top right) are independent, but Java collaboration systems link multiple clients through object (service) sharing.

NPAC has developed two systems TANGO [21,22] and WebFlow described in Section 5.3, which illustrate the differences between collaborative and computational sharing. Both NPAC systems use Java servers for their middle-tier level. TANGO, which can be considered here as equivalent to NCSA's Habanero [23], provides collaboration services as in Figure 7. Client-side applications are replicated using an event distribution model, and to put a new application into TANGO, one must be able to define both its absolute state and changes therein. Using Java object serialization or similar mechanisms, this state is maintained identically in the linked applications. On the other hand, WebFlow can be thought of as a Web version of AVS or Khoros and as in Figure 5(a) integrates program modules together using a dataflow paradigm. Now the module developer defines data input and output interfaces, and builds methods to handle data I/O. However, there would typically be no need to replicate the state of module in a WebFlow application.

6.2. Object Web and Distributed Simulation

The integration of HPCC with distributed objects provides an opportunity to link the classic HPCC ideas with those of DoD's distributed simulation DIS or Forces Modeling FMS community. The latter do not make extensive use of the Web these days, but they have a commitment to distributed objects with their HLA (High Level Architecture) and RTI (Runtime Infrastructure) initiatives. The Naval postgraduate school at Monterey [24,25] has built some interesting web-linked DIS prototypes. However, distributed simulation is traditionally built with distributed event-driven simulators managing C++ or equivalent objects. We suggest that the object web (and parallel and distributed ComponentWare described in Section 6.3) is a natural convergence point for HPCC and DIS. This would provide a common framework for time-stepped, real-time and event-driven simulations. Further, it will allow one to more easily build systems that integrate these concepts as is needed in many major DoD projects. As a simple example, note that an event driven overall battle scenario could have a component that was a time stepped weather simulation.

6.3. HPCC ComponentWare

HPCC does not have a good reputation for the quality and productivity of its programming environments. Indeed, one of the difficulties with adoption of parallel systems is the rapid improvement in performance of workstations and recently PC's with much better development environments. Parallel machines do have a clear performance advantage, but this for many users, this is more than counterbalanced by the greater programming difficulties. We can give two reasons for the lower quality of HPCC software. Firstly, parallelism is intrinsically hard to find and express. Secondly, the PC and workstation markets are substantially larger than HPCC and so can support a greater investment in attractive software tools such as the well-known PC visual programming environments. The DcciS revolution offers an opportunity for HPCC to produce programming environments that are both more attractive than current systems, and further could be much more competitive than previous HPCC programming environments with those being developed by the PC and workstation world. Here, we can also give two reasons. Firstly, the commodity community must face some difficult issues as they move to a distributed environment that has challenges where in some cases the HPCC community has substantial expertise. Secondly, as already described, we claim that HPCC can leverage the huge software investment of these larger markets.

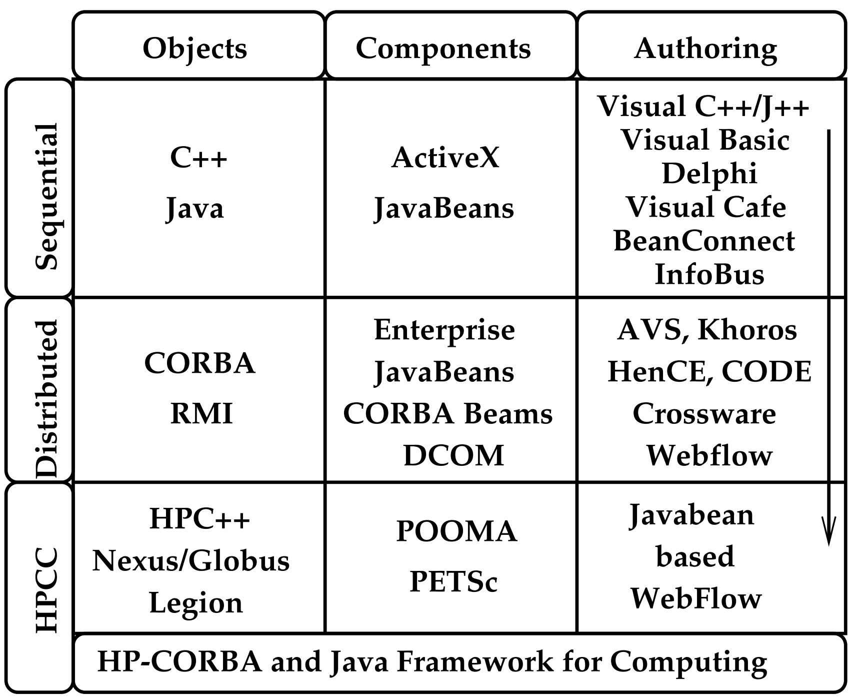

Figure 8: System Complexity (vertical axis) versus User Interface (horizontal axis) tracking of some technologies

In Figure 8, we sketch the state of object technologies for three levels of system complexity—sequential, distributed and parallel and three levels of user (programming) interface—language, components and visual. Industry starts at the top left and moves down and across the first two rows. Much of the current commercial activity is in visual programming for sequential machines (top right box) and distributed components (middle box). Crossware (from Netscape) represents an initial talking point for distributed visual programming. Note that HPCC already has experience in parallel and distributed visual interfaces (CODE [19] and HeNCE [20] as well as AVS [17] and Khoros [18]). We suggest that one can merge this experience with Industry's Object Web deployment and develop attractive visual HPCC programming environments. Currently, NPAC's WebFlow, described in Section 5.3, uses a Java graph editor to compose systems built out of modules. This could become a prototype HPCC ComponentWare system if it is extended with the modules becoming Javabeans and integration of web with CORBA at tier 2. Note the linkage of modules would incorporate the generalized communication model of Figure 5. Returning to Figure 1, we note that as industry moves to distributed systems, they are implicitly taking the sequential client-side PC environments and using them in the much richer server (middle-tier) environment that traditionally had more closed proprietary systems.

7. COMMODITY TECHNOLOGIES FOR PARALLEL COMPUTING ENVIRONMENTS

7.1. High Performance Commodity Communication

Most of the discussion in this paper has been devoted to the use of commodity technologies for computational grids or the field that is sometimes termed HPDC (High Performance Distributed Computing). However, as we started to explore in the last subsection, we believe that one can also use commodity technologies to build parallel computing environments that combine both high functionality and high performance. As usual, the functionality comes from inheriting commodity services, and the challenge is to combine these with high performance.

First compare the two views of a parallel computer in Figures 9 and 10.

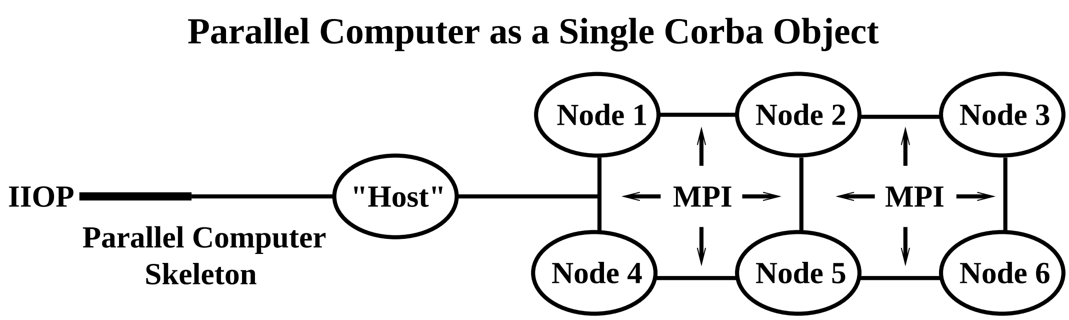

Figure 9: A Parallel Computer viewed as a single CORBA object in a classic "Host-node computing model.

In the above figure, we see a simple multi-tier view with commodity protocols (HTTP, RMI, COM or the IIOP pictured) used to access the parallel computer as a single entity. This entity (object) delivers high performance in an obvious way by running classic HPCC technologies (such as HPF, PVM or the pictured MPI) in the third tier. This has been successfully implemented by many groups [13] to provide web interfaces to parallel computing systems [10]. Although this provides important commodity services to the user based on Java and JavaScript client interfaces, the approach only addresses the parallel computer as a single object and essentially implies the "host-node" model of parallel programming. Note that in Figures 9 and 10, we draw various nodes and the host as separate entities. These represent logically distinct functions, but the physical implementation need not reflect the distinct services. In particular, two or more capabilities can be implemented on the same sequential or SMP system.

In Figure 9, we are not exploiting the distributed computing support of commodity technologies for parallel programming. However, in Figure 10, we view the parallel computer as a distributed computer with a particularly fast network and integrated architecture.

Each Node of a Parallel Computer as a Corba Object

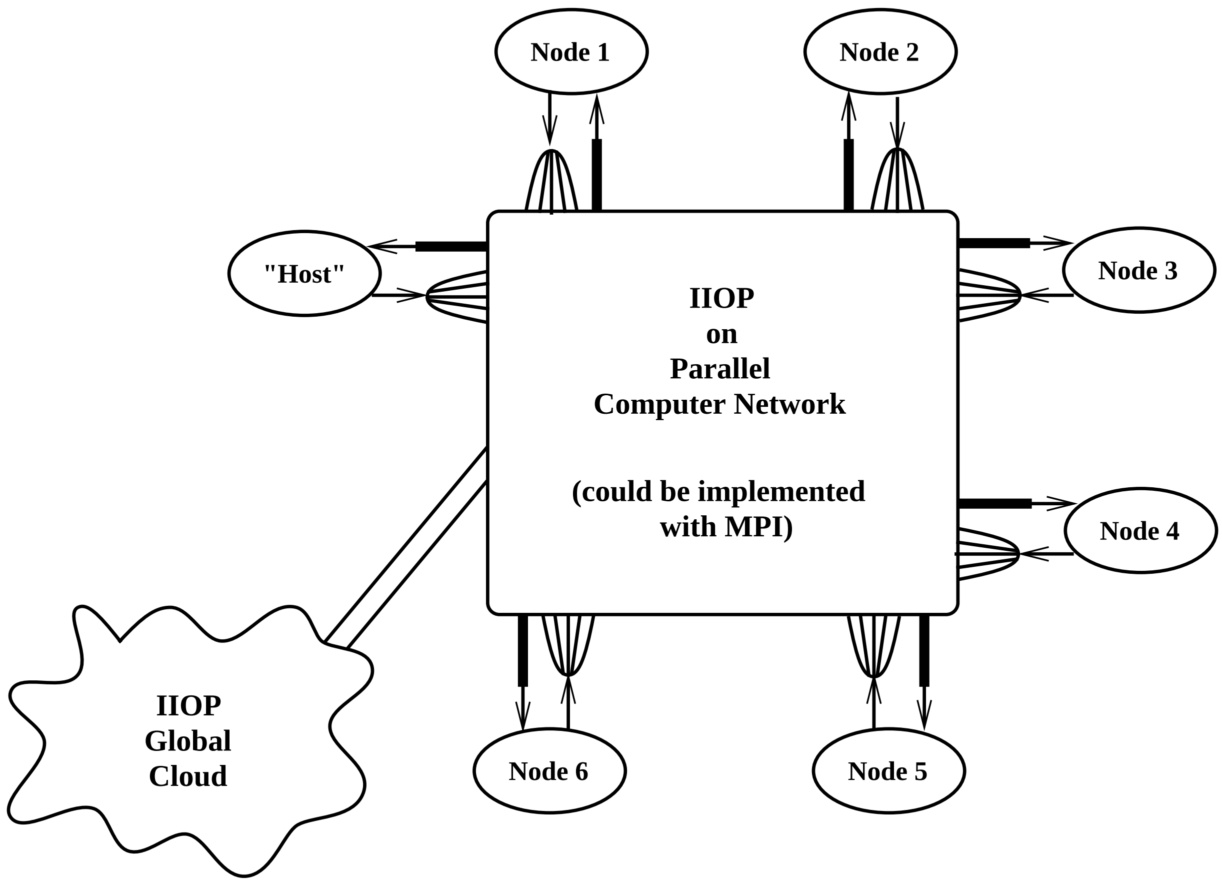

Figure 10: Each node of a parallel computer instantiated as a CORBA object.

In Figure 10, each node of the parallel computer runs a CORBA ORB (or perhaps more precisely a stripped down ORBlet), Web Server or equivalent commodity server. Now commodity protocols can operate both internally and externally to the parallel machine. This allows a particularly powerful environment where one can uniformly address the full range of commodity and high-performance services. Further, tools such as the visual environment of Section 6.3 can now be applied to parallel as well as distributed computing. Obviously, one should be concerned that this flexibility has been accompanied by a reduction in communication performance from that of Figure 9. Indeed, most messaging protocols such as RMI, IIOP and HTTP have unacceptable performance for most parallel computing applications. However, we can use the ideas of Section 5 to obtain good performance with a suitable binding of MPI and PVM to the commodity protocols. In Figure 3, we redraw Figure 2 in a fashion to demonstrate the analogy to Figure 10.

In Figure 11, we extend the previous figure to show 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.

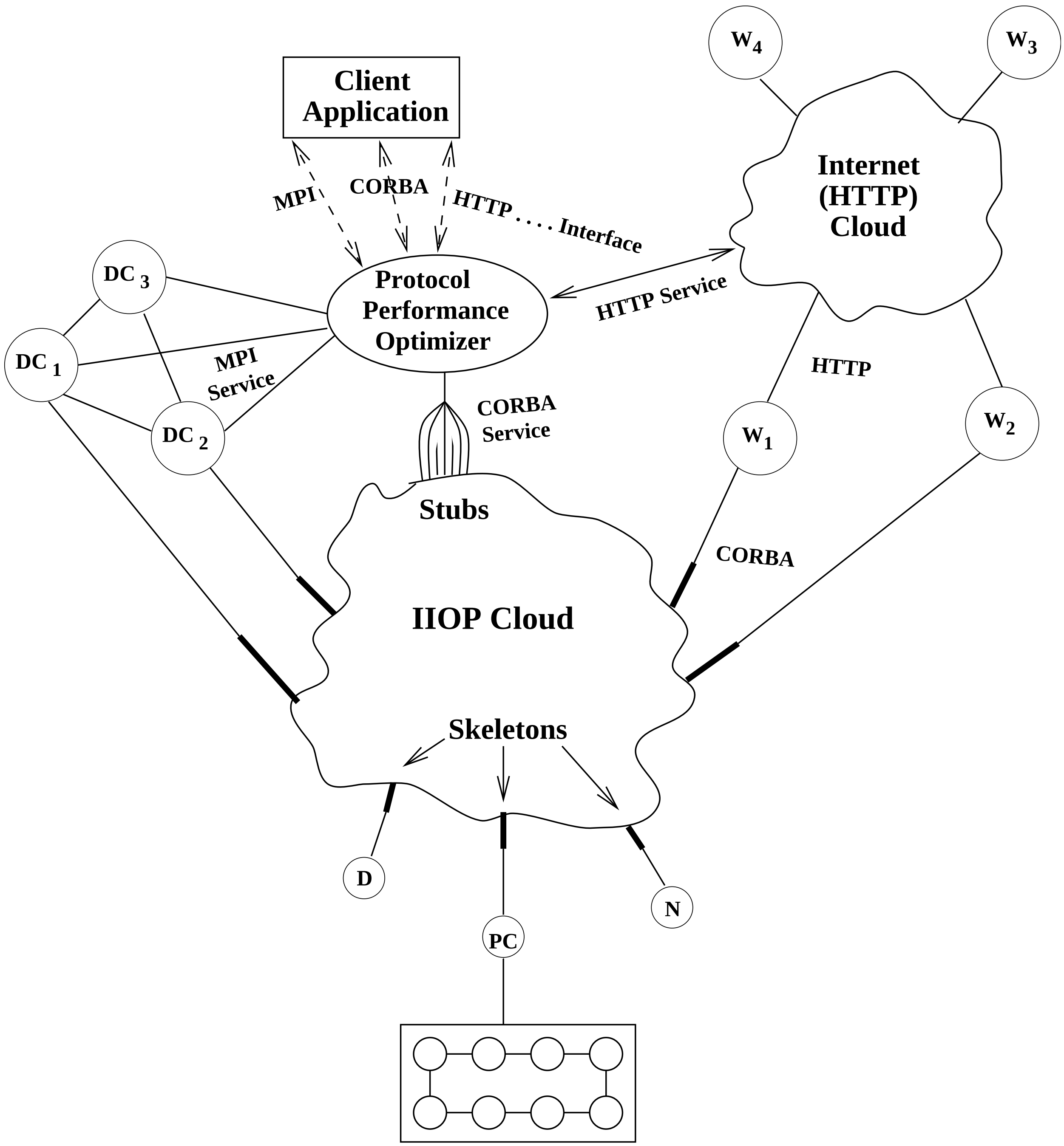

Figure 11: A message optimization bridge allows MPI (or equivalently Nexus, Globus or PVM) and commodity technologies to coexist with a seamless user interface.

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 supports HTTP CORBA or MPI. Of course, one can substitute equivalent commodity (RMI, COM) or HPCC technologies (PVM, PVMPI, Nexus, Globus) in the above discussion.

Note that there are two ways of linking MPI and CORBA. Firstly, there is the MPI function call that actually calls a CORBA stub; secondly, a CORBA invocation can be trapped and replaced by an optimized MPI implementation. We are investigating this at NPAC in the context of a Java MPI linkage, which raises already 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. This preference can be set using the mechanism of Section 5. As stressed by Gannon [26], there are important research issues in efficient object serialization needed for a high-performance implementation of Figure 11.

7.2. HPJava—Java as a High Performance Computing Language

We have, in fact, discussed many critical uses of Java in both client interfaces and tier-two servers to high-performance systems. Here, we discuss its direct use as a scientific and engineering programming language [4,5,27,28,29]. We first discuss Java's possible role as the basic programming language for science and engineering—taking the role now played by Fortran 77, Fortran 90, and C++. In our three-tier architecture, this is the use of Java in tier-three engineering and science applications or in the CORBA HPCC vertical facility. We now discuss this controversial area. One of Java's important advantages over other languages is that it will be learnt 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. Java is a very social language as one naturally gets Web pages from one's introductory Java exercises that can be shared with one's peers. We have found this as a helpful feature for introductory courses. Of course, the Web is the only real exposure to computers for many children, and the only languages to which they are typically exposed to today are Java, JavaScript, and Perl. We find it difficult to believe that entering college students, fresh from their Java classes, will be willing to accept Fortran, which will appear quite primitive in contrast. C++ as a more complicated systems-building language may well be a natural progression, but although quite heavily used, C++ has limitations as a language for simulation. In particular, it is hard for C++ to achieve good performance on even sequential and parallel code, and we expect compiled Java not to have these problems.

In fact, let us now discuss performance, which is a key issue for Java. As discussed in the two workshops on Java for Science and Engineering computation [27], there seems little reason why native Java compilers, as opposed to current portable JavaVM interpreters or Just in Time compilers (JIT), cannot obtain comparable performance to C or Fortran compilers. Difficulties in compiling Java include its rich exception framework that could restrict compiler optimizations. Users would need to avoid complex exception handlers in performance critical portions of a code. Other important issues with Java include the lack of any operator overloading that could allow efficient elegant handling of Fortran constructs like COMPLEX. There is much debate on Java's rule (philosophy?) that code not only run everywhere but give the same value on all machines. This inhibits optimization on machines, such as the Intel Pentium, that include multiply 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 and their absence, of course, allows much more optimization for both sequential and parallel codes. Optimistically, we can say that Java shares the object-oriented features of C++ and the performance features of Fortran. One interesting area is the expected performance of Java interpreters (using just in time techniques) and compilers on the Java bytecodes (Virtual Machine). Here, we find today perhaps a factor of

3–10 lower performance from a PC JIT compiler compared to C compiled code and this can be expected to improve to become "only" a factor of two performance hit. As described above, with some restrictions on programming style, we expect Java language or VM compilers to be competitive with the best Fortran and C compilers. Note that we can also expect a set of high-performance ‘‘native class” libraries to be produced that can be downloaded and accessed by applets to improve performance in the usual areas one builds scientific libraries.

In order to discuss parallel Java, we consider four forms of parallelism seen in applications.

1. Data Parallelism

Here, we refer to natural large-scale parallelism found from parallel updates of grid-points, particles and other basic components in scientific computations. Such parallelism is supported in Fortran by either high-level data parallel HPF or at a lower-level Fortran plus message passing (MPI). Java does not have any built-in parallelism of this type, but at least the lack of pointers means that natural parallelism is less likely to get 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++). This can be done using threads on shared memory machines as pioneered at Indiana, while in distributed memory machines, one must use message passing. Here, the situation is clearly satisfactory for Java as the language naturally supports inter-program communication, and the standard capabilities of high-performance message passing are being implemented for Java. In Section 7.1, we pointed out that there is an interesting relationship between the Java-CORBA and Java MPI linkage. At NPAC, we are focussing on SPMD rather than the HPF style approach to data parallel Java. This corresponds to providing Java with a powerful runtime to support data parallel applications [30,31].

2. Modest Grain Size Functional Parallelism

Here, we are thinking of the type of parallelism used when computation and I/O operation are overlapped as 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

This is quite natural for C++ or Java where the latter can use the applet mechanism to portably represent objects. We have already discussed this in Sections 6.1 and 6.2 for shared objects and the large class of military simulations that use large scale object based models.

4. Metaproblems

This is the parallelism in applications that are made up of several different sub-problems that themselves may be sequential or parallel. We have already discussed in the earlier sections, the power of Java in this case for overall coarse grain software integration. This is use of Java in CORBA and web servers in tier two, and is explicitly discussed in Section 5 to link application components together.

In summary, Java directly addresses three of the four forms of parallelism described above. In these areas, it seems superior to other languages. Java needs to be augmented to fully support data parallelism but so do Fortran and C++!

Interpreted Environments

Java and Web technology suggest new programming environments that integrate compiled and interpreted or scripting languages. In Figure 12, we show a system built at NPAC [32] that uses an interpreted Web client interacting dynamically with compiled code through a typical tier-two server. This happens to be designed for HPF back-end, but the architecture is independent of the back-end language. The Java or JavaScript front-end holds proxy objects produced by an HPF front-end operating on the back-end 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. Note that for compiled (parallel) Java, use of objects (as opposed to simple types in the language) has probably unacceptable overhead. However, they are well used in interpreted front-ends where object references are translated into efficient compiled code. We see such hybrid architectures as quite attractive and warranting further research.

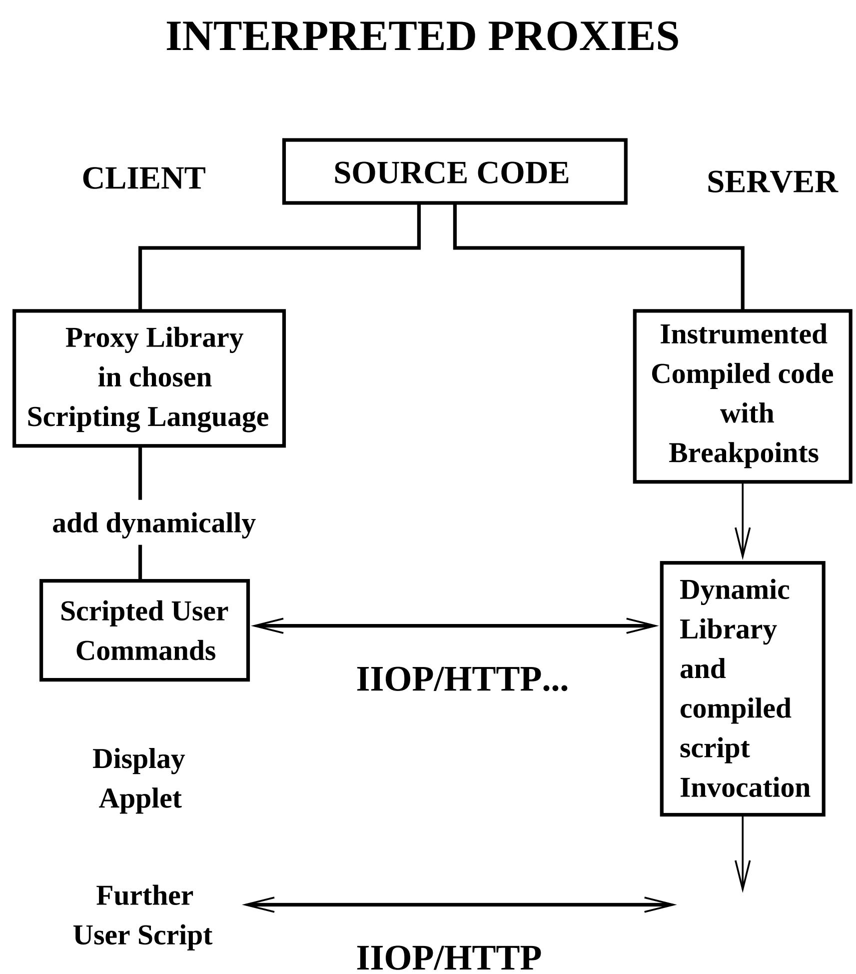

Figure 12: An architecture for an interpreted Java front-end communicating with a tier-two server controlling dynamically a HPCC backend.

In summary, we see that Java has no obvious major disadvantages and some clear advantages compared to C++ and especially Fortran as a basic language for large-scale simulation and modeling. Obviously, we should not and cannot port all our codes to Java. Rather, we can start using Java or more precisely Javabeans for wrappers and user interfaces. 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 Web technology in all aspects of their work.

8. BACKGROUND REFERENCES ON COMMODITY TECHNOLOGIES

The base material on CORBA can be found on the OMG Web site [omg]. This includes OMG Formal Documentation [omg2], Recently Adopted Specifications and The Technical Committee Work in Progress [omgtc], which offers up-to-date on-line information on the individual RFPs and their adoption process. One such RFP of a particular relevance for this Chapter, CORBA Components [rfp], has been recently posted by OMG in response to the Position Paper [inos] by IBM, Netscape, Oracle and SunSoft, with mid November '97 as the first submission deadline. It is expected that this activity will result in JavaBeans based ComponentWare model for CORBA. Primary source of information on JavaBeans is the JavaSoft Web site [beans]. See also the recent O'Reilly book by Robert Englander [beans:97]. A good recent reference on Microsoft COM (Component Object Model) is Microsoft's Press book by Dale Rogerson [com]. CORBA/COM integration model is specified in the Core CORBA 2.1 document [iiop], Chapters 14 (Interworking), 15 (COM/CORBA mapping) and 16 (OLE Automation/CORBA mapping). A good overview of CORBA/Java integration and the Object Web concepts can be found in the recent book Robert Orfali and Dan Harkey [orfali:97]. The two currently most popular commercial Java ORBs are: OrbixWeb by IONA [iona] and VisiBroker for Java by Visigenic [visi]. The first public domain ORBs became recently available such as JacORB [jacorb] by University of Berlin, omniBroker by Olivetti and Oracle Labs [omni] or Electra by Olsen & Associates [electra]. These public domain ORB's facilitate several ongoing research projects on using CORBA for reliable distributed or/and high performance computing which we list below.

Nile, a National Challenge Computing Project [14] develops 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. NILE is CORBA based and it uses the Electra ORB.

Douglas Schmidt, Washington University, conducts research on high performance implementations of CORBA [schmidt], geared towards real-time image processing and telemedicine applications on workstation clusters over ATM. His high performance ORB—TAO [tao]—based on optimized version of public domain IIOP implementation from SunSoft outperforms commercial ORB's by factor 2-3. Steve Vinoski, IONA and Douglas Schmidt address current R&D topics on the use of CORBA for distributed computing in their C++ Report column [vinoski]. Richard Muntz, UCLA, explores the use of CORBA for building large-scale object based data mining systems. His OASIS (Open Architecture Scientific Information System) [mesrobian:96] environment for scientific data analysis allows 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.

NPAC is developing a public domain Java based IIOP and HTTP server, JWORB [jworb], with the alpha release planned by mid '98 [jworb]. New anticipated developments in CORBA based distributed computing include emergent CORBA facilities in specialized areas such as Workflow [wfmd] or Distributed Simulations [dmso].

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

[beans:97] "*Developing JavaBeans*" by Robert Englander, O'Reilly & Associates, June '97, ISBN: 1-56592-289-1.

[com] "*Inside COM - Microsoft's Component Object Model*" by Dale Rogerson, Microsoft Press, 1997, ISBN: 1-57231-349-8.

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

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

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

[inos] "*CORBA Component Imperatives*" - a position paper by IBM, Netscape, Oracle and SunSoft, http://www.omg.org/news/610pos.htm

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

[jacorb] JacORB by Freie Universität Berlin, http://www.inf.fu-berlin.de/~brose/jacorb/

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

[mesrobian:96] 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, La., February, 1996. See also http://techinfo.jpl.nasa.gov/JPLTRS/SISN/ISSUE36/MUNTZ.htm

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

[omg2] OMG Formal Documentation, http://www.omg.org/library/specindx.htm

[omgtc] OMG TC Work in Progress, http://www.omg.org/library/schedule.htm

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

[orfali:97] "*Client/Server Programming with Java and CORBA*" by Robert Orfali and Dan Harkey, Wiley, Feb'97, ISBN: 0-471-16351-1

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

[schmidt] "*Research on High Performance and Real-Time CORBA*" by Douglas Schmidt,http://www.cs.wustl.edu/~schmidt/CORBA-research-overview.html

[tao] "Real-time CORBA with TAO (The ACE ORB)" by Douglas Schmidt, http://www.cs.wustl.edu/~schmidt/TAO.html

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

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

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

REFERENCES

1. G. C. Fox and W. Furmanski, chapter in book "*Computational Grids: The Future of High-Performance Distributed Computing*", to be published by Morgan-Kaufmann (1998) and edited by Carl Kesselman and Ian Foster.

2. David Bernholdt, Geoffrey Fox and Wojtek Furmanski, *"Towards High Performance Object Web Based FMS"*, White Paper for ARL MSRC PET Program, Sept. 97. See http://osprey7.npac.syr.edu:1998/iwt98/projects/webhla/

3. 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).

4. Geoffrey Fox and Wojtek Furmanski, "*Petaops and Exaops: Supercomputing on the Web*", IEEE Internet Computing, 1(2), 38-46 (1997); http://www.npac.syr.edu/users/gcf/petastuff/petaweb

5. 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(6), 4135-426(1997). Web version in ref. [27].

6. Geoffrey Fox, *"Introduction to Web Technologies and Their Applications"*, Syracuse report SCCS-790. http://www.npac.syr.edu/techreports/html/0750/abs-0790.html. This is an elementary overview of "classic" Web Technologies.

7. Geoffrey Fox, Wojtek Furmanski, and Shrideep Pallickara**,** *Building Distributed Systems on the Pragmatic Object Web***",** to be published, http://www.npac.syr.edu/users/shrideep/book/

8. 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/. This is the beginning of a compilation of input for defining interfaces for a horizontal CORBA HPcc facility or more precisely it is aimed at one aspect – a seamless interface for users to computing systems.

9. K.Dincer, G. Fox, *"Using Java and JavaScript in the Virtual Programming Laboratory: a web-based parallel programming environment"*, Concurrency Practice and Experience 9,521-534(97). http://www.npac.syr.edu/users/dincer/papers/vpl/

10. WebSubmit supercomputer job submission system from NIST, http://www.boulder.nist.gov/websubmit/index.html

11. B.O. Christiansen, P.Cappello, M.F. Ionescu, M.O. Neary, K.E. Schauser, and D. Wu, "Javelin: Internet-based parallel computing using Java", http://www.cs.ucsb.edu/~schauser/papers/97-javelin.ps, Concurrency: Practice and Experience, Vol. 9, 1139-1160(97)

12. Article in August 1997 Byte on *3-tier commercial computing model*. http://www.byte.com/art/9708/sec5/art1.htm

13. Jim Almond at ECMWF in Reading, England "*Resource for seamless computing*", http://www.ecmwf.int/html/seamless/

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

15. Geoffrey Fox, Wojtek Furmanski and Tomasz Haupt , "*ASCI WebFlow: High-Level Programming Environment and Visual Authoring Toolkit for High Performance Distributed Computing*" , unpublished study.

16. See the Globus site at http://www.globus.org for the collection of protocols, software, testbeds and documents such as: "*The Globus Project: A Status Report*." by I. Foster and C. Kesselman. To appear in Heterogeneous Computing Workshop, 1998. This describes the current status of the Globus System as of early 1998 http://www.globus.org/globus/papers/gram.ps.Z

17. For AVS product information, see: Advanced Visual Systems Home Page, http://www.avs.com/. For an example of AVS used for coarse-grain software integration see: Cheng, G., Faigle. C., Fox, G.C., Furmanski, W., Li, B. and Mills, K., "*Exploring AVS for HPDC Software Integration: Case Studies Towards Parallel Support for GIS*", in Proceedings of the 2nd Annual International AVS Conference The Magic of Science: AVS'93, Lake Buena Vista, Florida, May 1993. http://www.npac.syr.edu/PROJECTS/PUB/wojtek/hpsin/doc/avs93.ps

18. For Khoros product information, see: Khoral Research Inc. Home Page, http://www.khoral.com/. For the history of the Khoros system, see: http://www.khoral.com/khoros/khoros2/history.html

19. For CODE, see James C. Browne's page at http://www.cs.utexas.edu/users/browne/ and documents listed there such as: P. Newton and J.C. Browne, "*The CODE 2.0 Graphical Parallel Programming Language*", Proc. ACM Int. Conf. on Supercomputing, July, 1992. J. C. Browne, S. I. Hyder, J. Dongarra, K. Moore, P. Newton, "*Visual Programming and Debugging for Parallel Computing*," IEEE Parallel and Distributed Technology, Spring 1995, Volume 3, Number 1, 1995

20. HeNCE is described in "HeNCE (Heterogeneous Network Computing Environment)" at http://www.netlib.org/hence/ and the documents listed there, e.g.: "*HeNCE: A Users' Guide Version 2.0*" by Adam Beguelin, Jack Dongarra, G. A. Geist, Robert Manchek, Keith Moore and Peter Newton, Vaidy Sunderam

21. 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(97). See http://trurl.npac.syr.edu/tango/

22. Tango Collaboration System, http://trurl.npac.syr.edu/tango/

23. Habanero Collaboration System, http://www.ncsa.uiuc.edu/SDG/Software/Habanero/

24. "*Distributed Interactive Simulation DIS-Java-VRML Working Group*" by Don Brutzman, Navy Postgraduate School, Monterey CA http://www.stl.nps.navy.mil/dis-java-vrml/

25. Daniel Dias, Geoffrey Fox, Wojtek Furmanski, Vishal Mehra, Balaji Natarajan, H.Timucin Ozdemir, Shrideep Pallickara and Zeynep Ozdemir, "*Exploring JSDA, CORBA and HLA based MuTech's for Scalable Televirtual (TVR) Environments*", to be presented at Workshop on Object-Oriented Technologies during the VRML98, Monterey, CA, Feb 16-19, 1998. http://king.npac.syr.edu:2006/src/hasan/FOILS/VR98/paperVR98.ps. See also preliminary NPAC WebHLA pages at: http://osprey7.npac.syr.edu:1998/iwt98/projects/webhla

26. D. Gannon in book "*Computational Grids: The Future of High-Performance Distributed Computing*", to be published by Morgan-Kaufmann (1998) and edited by Carl Kesselman and Ian Foster.

27. *Compilation of References to use of Java in Computational Science and Engineering including proceeding of Syracuse (December 96) and Las Vegas (June 97) meetings*. http://www.npac.syr.edu/projects/javaforcse. This includes a summary of discussion of June 21,97 meeting at Las Vegas on *Issues impacting use of Java in Scientific computation*: http://www.npac.syr.edu/projects/javaforcse/june21summary.html

28. Compilation of *NPAC Activities in Web-based HPcc*, http://www.npac.syr.edu/projects/webspace/webbasedhpcc/

29. B. Carpenter et al., "HPJava Resource at NPAC", (includes Java-MPI binding), http://www.npac.syr.edu/projects/pcrc/HPJava/

30. Bryan Carpenter, Guansong Zhang, Geoffrey Fox, Xinying Li and Yuhong Wen, "HPJava: Data Parallel Extensions to Java", http://www.npac.syr.edu/projects/pcrc/doc, Proceedings of Java '98 San Jose Feb28-March 1 1998 and to be published in Concurrency:Practice and Experience,

31. Bryan Carpenter and Geoffrey Fox, Donald Leskiw, Xinying Li, Yuhong Wen and Guansong Zhang, "Language Bindings for a Data-parallel Runtime", in proceedings of Third International Workshop on High-Level Parallel Programming Models and Supportive Environments (1998), http://www.npac.syr.edu/projects/pcrc/doc

32. E. Akarsu, G. Fox, T. Haupt, "*DARP: Java-based Data Analysis and Rapid Prototyping Environment for Distributed High Performance Computations*", proceedings of Java '98 San Jose Feb28-March 1 1998 and to be published in Concurrency:Practice and Experience, http://www.npac.syr.edu/projects/hpfi/

9. GLOSSARY

Applets

An application interface where referencing (perhaps by a mouse click) a remote application as a hyperlink to a server causes it to be downloaded and run on the client. Typically applets are written in Java.

CGI (Common-Gateway-Interface)

A method for communication between a browser and a server for processing user input through a script and generating output that is sent back to the user. This script is often written in PERL but any interpreted or compiled code is supported.

CORBA (Common Object Request Broker Architecture)

An approach to cross-platform cross-language distributed object developed by a broad industry group, the OMG. CORBA specifies basic services (such as naming, trading, persistence) the protocol IIOP used by communicating ORBS, and is developing higher level facilities which are object architectures for specialized domains such as banking (Figure 7).

COM (Common Object Model)

Microsoft's windows object model, which is being extended to distributed systems and multi-tiered architectures. ActiveX controls are an important class of COM object, which implement the component model of software. The distributed version of COM used to be called DCOM.

ComponentWare

An approach to software engineering with software modules developed as objects with particular design frameworks (rules for naming and module architecture) and with visual editors both to interface to properties of each module and also to link modules together.

DIS (Distributed Interactive Simulation)

Original framework developed in such projects as SIMNET, to support FMS and IMT applications. HLA and RTI are superceding DIS.

HLA (High Level Architecture)

Current object architecture for FMS and IMT applications. HLA (and RTI) are expected to become a CORBA facility.

HPcc (High Performance commodity computing)

NPAC project to develop a commodity computing based high performance computing software environment. Note that we have dropped "communications" referred to in the classic HPCC acronym. This is not because it is unimportant but rather because a commodity approach to high performance networking is already being adopted. We focus on high level services such as programming, data access and visualization that we abstract to the rather wishy-washy "computing" in the HPcc acronym.

HPCC (High Performance Computing and Communication)

Originally a formal federal initiative but even after this ended in 1996, this term is used to describe the field devoted to solving large-scale problems with powerful computers and networks.

HTML (Hypertext Markup Language)

Syntax for describing documents to be displayed on the World Wide Web

HTTP (Hyper Text Transport Mechanism)

A stateless transport protocol allowing control information and data to be transmitted between web clients and servers.

Hyperlink

The user-level mechanism (remote address specified in a HTML or VRML object) by which remote services are accessed by Web Clients or Web Servers.

IIOP (Internet Inter Orb Protocol)

A stateful protocol allowing CORBA ORB's to communicate with each other, and transfer both the request for a desired service and the returned result.

JDBC (Java Data Base Connection)

A set of interfaces (Java methods and constants) in the Java 1.1 enterprise framework, defining a uniform access to relational databases. JDBC calls from a client or server Java program link to a particular "driver" that converts these universal database access calls (establish a connection, SQL query, etc.) to particular syntax needed to access essentially any significant database.

Java

An object-oriented programming language from Sun, suitable for Web development due to the built-in portable support for mobility, networking, multithreading and graphical user interfaces. Java can either be interpreted from the JavaVM intermediate form or compiled to a native machine model.

Javabean

Part of the Java 1.1 enhancements defining design frameworks (particular naming conventions) and inter Javabean communication mechanisms for Java components with standard (Bean box) or customized visual interfaces (property editors). Enterprise Javabeans are Javabeans enhanced for server side operation with capabilities such as multi user support. Javabeans are Java's component technology and in this sense are more analogous to ActiveX than either COM or CORBA. However Javabeans augmented with RMI can be used to build a "pure Java" distributed object model.

JavaVM

A virtual machine or abstract computer that is defined by the bytecodes downloaded in an Applet. This is target of javac compiler that compiles Java code for Web use.

Object Web

The evolving systems software middleware infrastructure gotten by merging CORBA with Java. Correspondingly merging CORBA with Javabeans gives Object Web ComponentWare. This is expected to compete with the COM/ActiveX architecture from Microsoft.

OMG (Object Management Group)

OMG is the organization of over 700 companies that is developing CORBA through a process of call for proposals and development of consensus standards.

ORB (Object Request Broker)

Used in both clients and servers in CORBA to enable the remote access to objects. ORB's are available from many vendors and communicate via the IIOP protocol.

RMI (Remote Method Invocation)

A somewhat controversial part of Java 1.1 in the enterprise framework which specifies the remote access to Java objects with a generalization of the UNIX RPC (Remote Procedure Call).

RTI (Run Time Infrastructure)

Run time defined to support HLA compliant simulations including "federated" interoperable simulations.

Servlet

A component of the Java Web Server (formerly known as *Jeeves*) that facilitates creation of executable programs on the server side of an HTTP connection. Servlets can be viewed as the tier-2 server equivalent of tier-1 Java Applets. They represent an interesting alternative to libraries for dynamic building of programs from components.

VRML 1.0 (Virtual Reality Modeling Language)

A standard network protocol for 3D-scene description based on a subset of the ASCII Open Inventor format from Silicon Graphics.

VRML 2.0

Extends VRML 1.0 towards interactive 3D worlds by adding Sensor nodes that detect user events, routes that connect user events with the scene graph nodes, and Script nodes that describe interactive behavior of the scene graph objects using a scripting language such as Java.

Web Client

Originally web clients displayed HTML and related pages but now they support Java Applets that can be programmed to give web clients the necessary capabilities to support general enterprise computing. The support of signed applets in recent browsers has removed crude security restrictions, which handicapped previous use of applets.

Web Servers

Originally Web Servers supported HTTP requests for information - basically HTML pages but included the invocation of general server side programs using the very simple but arcane CGI - Common Gateway Interface. A new generation of Java servers have enhanced capabilities including server side Java program enhancements (Servlets) and support of stateful permanent communication channels.