HPcc as High Performance Commodity Computing on top of integrated Java, CORBA, COM and Web standards

G.C. Fox, W. Furmanski, T. Haupt, E. Akarsu and H. Ozdemir

Northeast Parallel Architectures Center, Syracuse University, Syracuse NY, USA
gcf@npac.syr.edu, furm@npac.syr.edu
http://www.npac.syr.edu

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 rapidly advancing Web 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 illustrate how performance can be obtained within a commodity architecture and we propose a middleware integration approach based on JWORB (Java Web Object Broker) multi-protocol server technology. Examples are given from collaborative systems, support of multidisciplinary interactions, proposed visual HPCC ComponentWare, quantum Monte Carlo and distributed interactive simulations.

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 3 tier commercial computing model, augmented by distributed object and component technologies of Java, CORBA, COM and the Web. This is followed in sections four and five by more detailed discussion of the HPcc core technologies and high-level services.

In this and related papers [5], we discuss several examples to address the following 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; and the step from object to visual component based programming in high performance distributed computing. Our claim, based on early experiments and prototypes is that HPcc is feasible but we need to exploit fully the synergies between several currently competing commodity technologies. We refer to our approach towards HPcc, based on integrating several popular distributed object frameworks as Pragmatic Object Web and we describe a specific integration metodology based on multi-protocol middleware server, JWORB (Java Web Object Request Broker).

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. 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, JavaBeans and new W3C object initiatives 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" [1] 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 clients and servers including applets and servlets, and new W3C technologies towards the Web Object Model (WOM) such as XML, DOM and RDF.

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,Java3D or DirectX are important for scientific visualization;
- Web (including Java applets and ActiveX controls) front-ends provide convenient customizable interoperable user interfaces to HPCC facilities;
- 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;
- The universal adoption of JDBC (Java Database Connectivity), rapid advances in the Microsoft's OLEDB/ADO transparent persistence standards 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, COM, CORBA and WOM 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 which 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

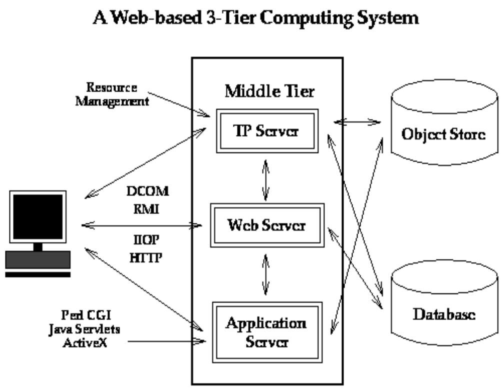

Figure 1: *Industry 3-tier view of enterprise Computing*

We start with a common modern industry view of commodity computing with the three tiers shown in fig 1. 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 3-tier solution, supports "thin" clients such as the currently popular network computer.

The commercial architecture is evolving rapidly and is exploring several approaches which co-exist in today's (and any realistic future) distributed information system. The most powerful solutions involve distributed objects. Currently, we are observing three important commercial object systems - CORBA, COM and JavaBeans, as well as the ongoing efforts by the W3C, referred by some as WOM (Web Object Model), to define pure Web object/component standards. 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! Finally, WOM is an emergent Web model that uses new standards such as XML, RDF and DOM to specify respectively the dynamic Web object instances, classes and methods.

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 [2] 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 from low- to high-level network standards:

- from HTTP to Java Sockets to IIOP or RMI
- from Perl CGI Script to Java Program to JavaBean distributed object

As an example consider the evolution of networked databases. Originally these were client-server with a proprietary network access protocol. In the next step, 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 fig. 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 coexist 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.

We also believe that the resultant architecture will be integrated with the web so that the latter will exhibit distributed object architecture shown in fig. 3.

More generally the emergence of IIOP (Internet Inter-ORB Protocol), CORBA2, rapid advances with the Microsoft's COM, DCOM, and COM+, and the realization that both CORBA and COM are 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 in fig. 3 shows a small object broker (a so-called ORBlet) in each browser as in

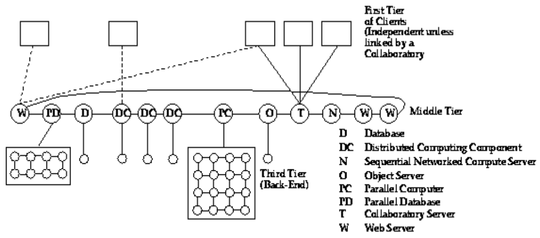

Figure 2: *Today's Heterogeneous Interoperating Hybrid Server Architecture. HPcc involves adding to this system, high performance in the third tier.*

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 which is a CORBA or Javabean distributed object model integrated with the web.

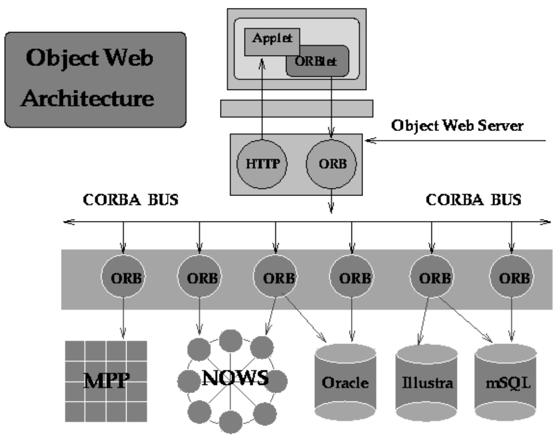

Figure 3: *Integration of Object Technologies (CORBA) and the Web*

We wish to use this service/object evolving 3-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 ref. [5] and sec. 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 fig. 2, we see that it elaborates fig. 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 in fig. 5, 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 an 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 which 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 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 fig 2 will become confused. In particular the physical realization does not necessarily reflect the logical architecture shown in fig 2.

4 Core Technologies for High Performance Commodity Systems

4.1 .ultidisciplinary Application

We can 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 fig. 4, 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 (or PVMPI). Here we view MPI as the "machine language" of the higher-level commodity communication model given by approaches such as WebFlow from NPAC.

There is the "pure" HPCC approach of fig. 5, which replaces all commodity web communication with HPCC technology. However there is a middle ground between the implementations of figs. 4 and 5 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 fig. 6 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 com-

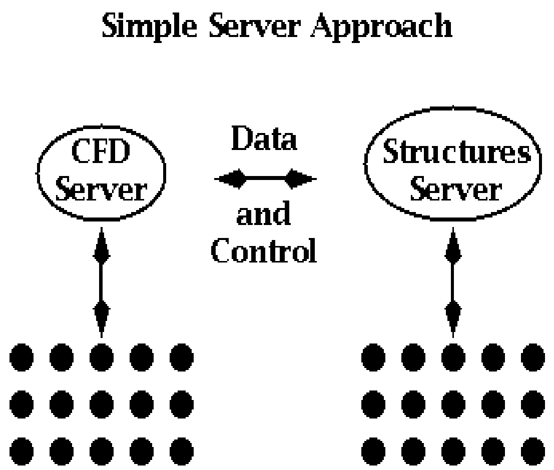

Figure 4: *Simple sequential server approach to Linking Two Modules*

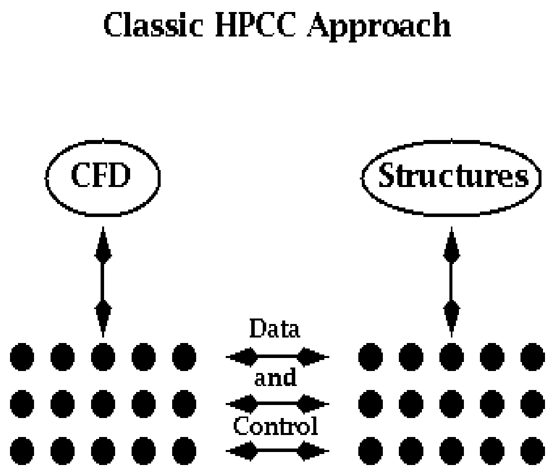

Figure 5: *Full HPCC approach to Linking Two Modules*

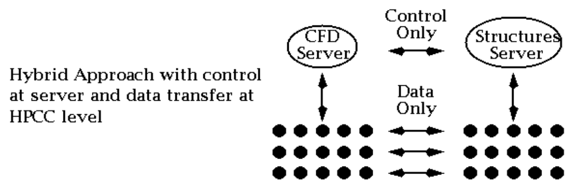

Figure 6: *Hybrid approach to Linking Two Modules*

plex control and service provision capability inherited naturally from the Web.

4.2 .avaBean Communication .odel

We note that JavaBeans (which are one natural basis of implementing program modules in the HPcc approach) provide a rich communication mechanism, which supports the separation of control (handshake) and implementation. As shown below in fig. 7, Javabeans use the JDK 1.1 AWT event model with listener objects and a registration/callback mechanism.

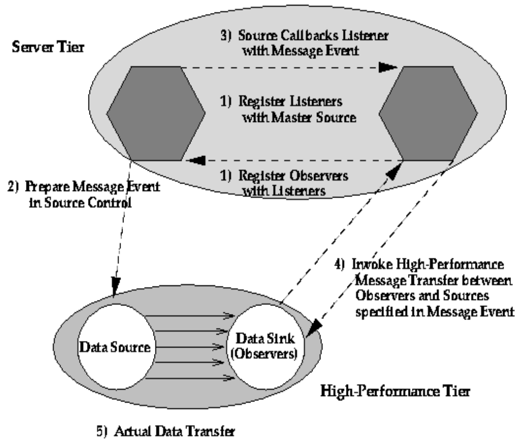

Figure 7: *JDK 1.1 Event Model used by (inter alia) Javabeans*

JavaBeans communicate indirectly with one or more "listener objects" acting as a bridge between the source and sink of data. In the model described above, this allows a neat implementation of separated control and explicit communication with listeners (a.k.a. sink control) and source control objects residing in middle tier. These control objects decide if high performance is necessary or possible and invoke the specialized HPCC layer. This approach can be used to advantage in "run-time compilation" and resource management with execution schedules and control logic in the middle tier and libraries such as MPI, PCRC and CHAOS implementing the determined data movement in the high performance (third) tier. Parallel I/O and "high-performance" CORBA can also use this architecture. In general, this listener model of communication provides a virtualization of communication that allows a separation of control and data transfer that is largely hidden from the user and the rest of the system. Note that current Internet security systems (such as SSL and SET) use high functionality public keys in the control level but the higher performance secret key cryptography in bulk data transfer. This is another illustration of the proposed hybrid multi-tier communication mechanism.

4.3 .W.RB based .iddleware

Enterprise JavaBeans that control, mediate and optimize HPcc communication as described above need to be maintained and managed in a suitable middleware container. Within our integrative approach of Pragmatic Object Web, a CORBA based environonment for the middleware management with IIOP based control protocol provides us with the best encapsulation model for EJB components. Such middleware ORBs need to be further integrated with the Web server based middleware to assure smooth Web browser interfaces and backward compatibility with CGI and servlet models. This leads us to the concept of JWORB (Java Web Object Request Broker) [6] - a multi-protocol Java network server that integrates several core services (so far dispersed over various middleware nodes as in Fig. 2) within a single uniform middleware management framework.

An early JWORB prototype has been recently developed at NPAC. The base server has HTTP and IIOP protocol support as illustrated in fig. 8. It can serve documents as an HTTP Server and it handles the IIOP connections as an Object Request Broker. As an HTTP server, JWORB supports base Web page services, Servlet (Java Servlet API) and CGI 1.1 mechanisms. In its CORBA capacity, JWORB is currently offering the base remote method invocation services via CDR based IIOP and we are now implementing higher level support such as the Interface Repository, Portable Object Adapter and selected Common Object Services.

During the startup/bootstrap phase, the core JWORB server checks its configuration files to detect which protocols are supported and it loads the necessary protocol classes (Definition, Tester, Mediator, Configuration) for each protocol. Definition Interface provides the necessary Tester, Configuration and Mediator objects. Tester object inpects the current network package and it decides how to interpret this particular message format. Configuration object is responsible for the configuration parameters of a particular protocol. Mediator object serves the connection. New protocols can be added simply by implementing the four classes de-

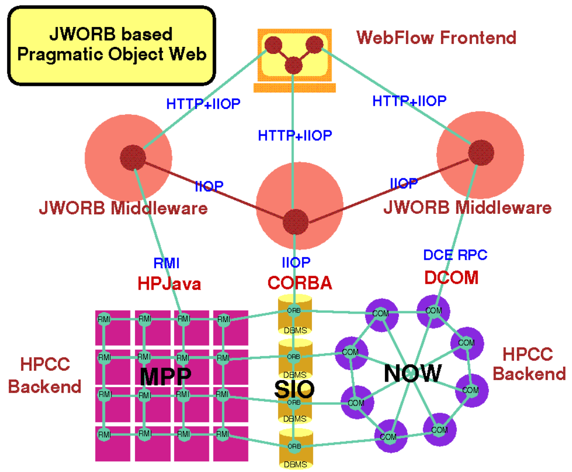

Figure 8: *Overall architecture of the JWORB based Pragmatic Object Web middleware*

scribed above and by registering a new protocol with the JWORB server.

After JWORB accepts a connection, it asks each protocol handler object whether it can recognize this protocol or not. If JWORB finds a handler which can serve the connection, is delegates further processing of the connection stream to this protocol handler. Current algorithm looks at each protocol according to their order in the configuration file. This process can be optimized with randomized or prediction based algorithm. At present, only HTTP and IIOP messaging is supported and the current protocol is simply detected based on the magic anchor string value (GIOP for IIOP and POST, GET, HEAD etc. for HTTP). We are currently working on further extending JWORB by DCE RPC protocol and XML co-processor so that it can also act as DCOM and WOM/WebBroker server.

5 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.

5.1 Distributed Collaboration .echanisms

The current Java Server model for the middle tier naturally allows one to integrate collaboration into the computing model and our approach allow 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 fig. 9, 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.

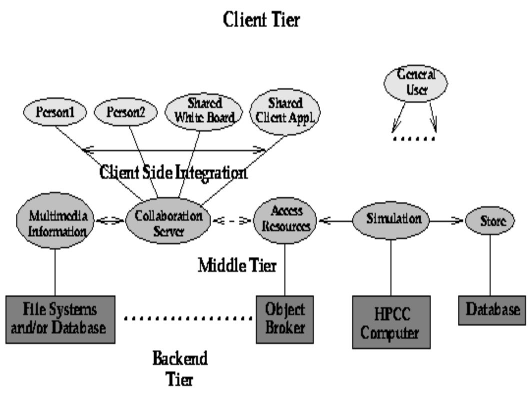

Figure 9: *Collaboration in today's Java Web Server implementation of the 3 tier computing model. Typical clients (on top right) are independent but Java collaboration systems link multiple clients through object (service) sharing*

If we are correct in viewing collaboration (see Tango [10,11] and Habanero [12]) as sharing of services between clients, the 3 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 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 application runs on "client" (e.g. shared Microsoft Word document) or on back-end tier as in case of a shared parallel computer simulation.

5.2 .bject 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 CORBA with their HLA (High Level Architecture) and RTI (Runtime Infrastructure) initiatives. 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 sec. 5.3) is a natural convergence point for HPCC and DIS/FMS. 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 exemplified by the FMS and IMT DoD computational activities which are part of the HPCC Modernization program.

HLA is a distributed object technology with the object model defined by the Object Model Template (OMT) specification and including the Federation Object Model (FOM) and the Simulation Object Model (SOM) components. HLA FOM objects interact by exchanging HLA interaction objects via the common Run-Time Infrastructure (RTI) acting as a software bus similar to CORBA. Current HLA/RTI follows a custom object specification but DMSO's longer term plans include transferring HLA to industry via OMG CORBA Facility for Interactive Modeling and Simulation.

At NPAC, we are anticipating these developments are we are building a prototype RTI implementation in terms of Java/CORBA objects using the JWORB middleware [7]. RTI is given by some 150 communication and/or utility calls,

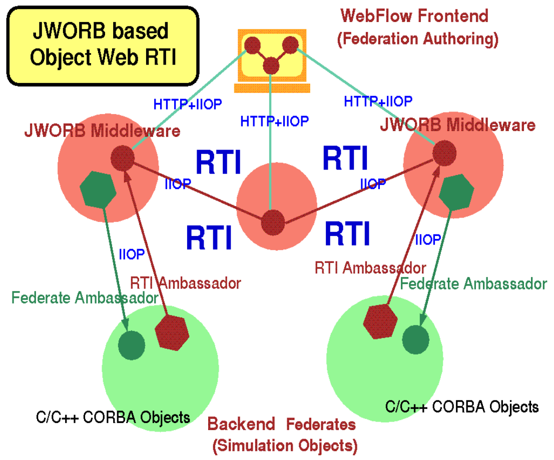

Figure 10: *Overall architecture of the Object Web RTI - a JWORB based RTI prototype recently developed at NPAC*

packaged as 6 main managment services: Federation Management, Object Management, Declaration Managmeent, Ownership Management, Time Management, Data Distribution Management, and one general purpose utility service. Our design shown in fig. 12 is based on 9 CORBA interfaces, including 6 Managers, 2 Ambassadors and RTIKernel. Since each Manager is mapped to an independent CORBA object, we can easily provide support for distributed management by simply placing individual managers on different hosts.

The communication between simulation objects and the RTI bus is done through the RTIambassador interface. The communication between RTI bus and the simulation objects is done by their FederateAmbassador interfaces. Simulation developer writes/extends FederateAmbassador objects and uses RTIambassador object obtained from the RTI bus.

RTIKernel object knows handles of all manager objects and it creates RTIambassador object upon the federate request. Simulation obtains the RTIambassador object from the RTIKernel and from now on all interactions with the RTI bus are handled through the RTIambassador object. RTI bus calls back (asynchronously) the FederateAmbassador object provided by the simulation and the federate receives this way the interactions/attribute updates coming from the RTI bus.

Although coming from the DoD computing domain, RTI follows generic design patterns and is applicable to a much broader range of distributed applications, including modeling and simulation but also collaboration, on-line gaming or visual authoring. From the HPCC perspective, RTI can be viewed as a high level object based extension of the low level messaging libraries such as PVM or MPI. Since it supports shared objects management and publish/subscribe based multicast channels, RTI can also be viewed as an advanced collaboratory framework, capable of handling both the multi-user and the multi-agent/multi-module distributed systems.

5.3 Visual H.CC 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, which 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.

In fig. 11, 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

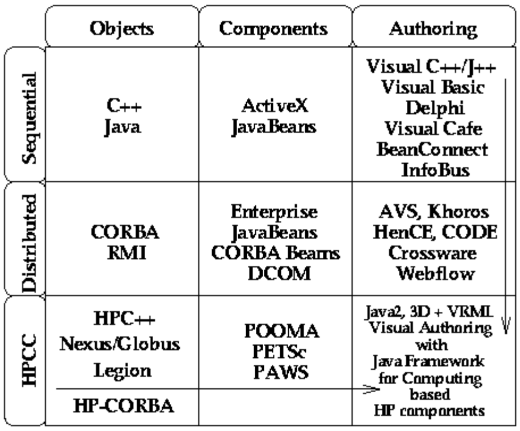

Figure 11: *System Complexity (vertical axis) versus User Interface (horizontal axis) tracking of some technologies*

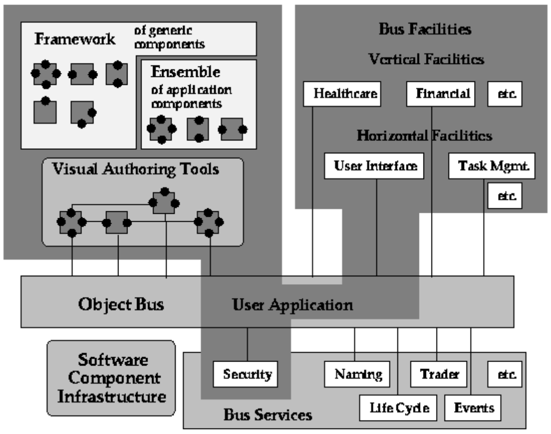

Figure 12: *Visual Authoring with Software Bus Components*

that HPCC already has experience in parallel and distributed visual interfaces (CODE and HenCE as well as AVS and Khoros). We suggest that one can merge this experience with Industry's Object Web deployment and develop attractive visual HPCC programming environments as shown in fig. 12.

Currently NPAC's WebFlow system [9][12] 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 the integration with CORBA. Note the linkage of modules would incorporate the generalized communication model of fig. 7, using a mesh of JWORB servers to manage a recourse pool of distributedHPcc components. An early version of such JWORB based WebFlow environment, illustrated in Fig. 13 is in fact operational at NPAC and we are currently building the Object Web management layer including the Entperprise JavaBeans based encapsulation and communication support discussed in the previous section.

Returning to fig. 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 which traditionally had more closed proprietary systems.

We will then generate an environment such as fig. 12 including object broker services, and a set of horizontal (generic) and vertical (specialized application) frameworks. We do not have yet much experience with an environment such as fig. 12, but suggest that HPCC could benefit from its early deployment without the usual multi-year lag behind the larger industry efforts for PC's. Further the diagram implies a set of standardization activities (establish frameworks) and new models for services and libraries that could be explored in prototype activities.

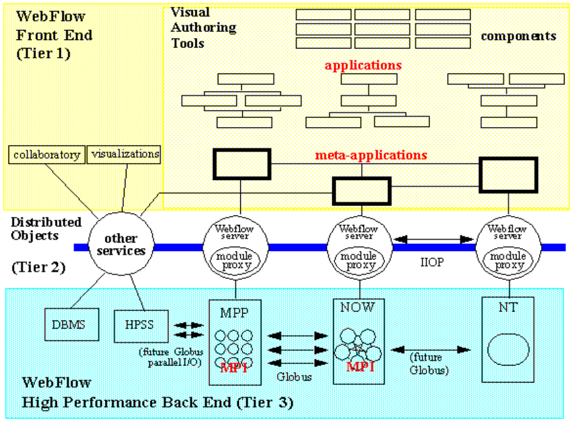

Figure 13: *Top level view of the WebFlow environment with JWORB middleware over Globus metacomputing or NT cluster backend*

5.4 Early User Communities

In parallel with refining the individual layers towards production quality HPcc environment, we start testing our existing prototypes such as WebFlow, JWORB and WebHLA for the selected application domains.

Within the NPAC participation in the NCSA Alliance, we are working with Lubos Mitas in the Condensed Matter Physics Laboratory at NCSA on adapting WebFlow for Quantum Monte Carlo simulations [13]. This application is illustrated in figs. 14 and 15 and it can be characterized as follows. A chain of high performance applications (both commercial packages such as GAUSSIAN or GAMESS or custom developed) is run repeatedly for different data sets. Each application can be run on several different (multiprocessor) platforms, and consequently, input and output files must be moved between machines. Output files are visually inspected by the researcher; if necessary applications are rerun with modified input parameters. The output file of one application in the chain is the input of the next one, after a suitable format conversion.

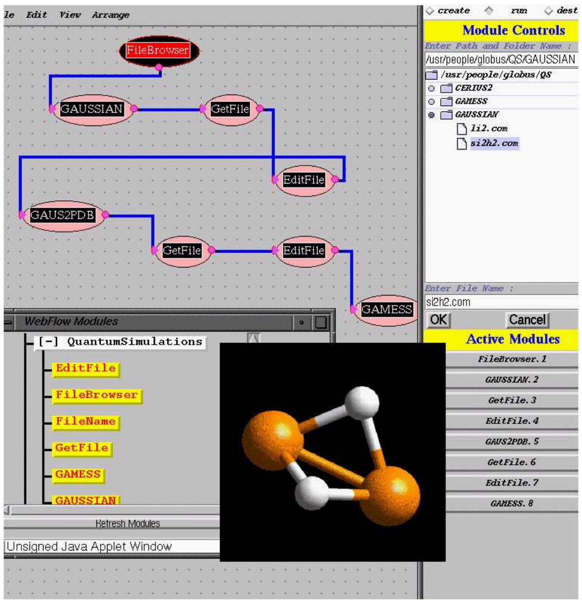

Figure 14: *Screendump of an example WebFlow session: running Quantum Simulations on a virtual metacomputer. Module GAUSSIAN is executed on Convex Exemplar at NCSA, module GAMESS is executed on SGI Origin2000, data format conversion module is executed on Sun SuperSparc workstation at NPAC, Syracuse, and file manipulation modules (FileBrowser, EditFile, GetFile) are run on the researcher's desktop*

The high performance part of the backend tier in implemented using the GLOBUS toolkit [14]. In particular, we use MDS (metacomputing directory services) to identify resources, GRAM (globus resource allocation manager) to allocate resources including mutual, SSL based authentication, and GASS (global access to secondary storage) for a high performance data transfer. The high performance part of the backend is augmented with a commodity DBMS (servicing Permanent Object Manager) and LDAP-based custom directory service to maintain geographically distributed data files generated by the Quantum Simulation project. The diagram illustrating the WebFlow implementation of the Quantum Simulation is shown in Fig. 15.

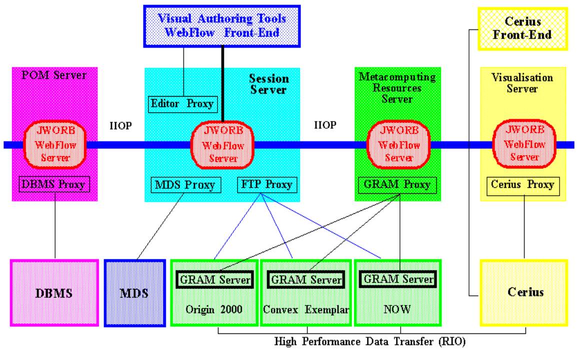

Figure 15: *WebFlow implementation of the Quantum Simulations problem*

Another large application domain we are currently adressing is DoD Modeling Simulation, approached from the perspective of FMS and IMT thrusts within the DoD Modernization Program. We already described the core effort on building Object Web RTI on top of JWORB. This is associated with a set of more application- or component-specific efforts such as: a) building distance training space for some mature FMS technologies such as SPEEDES; b) parallelizing and CORBA-wrapping some selected computationally intense simulation modules such as CMS (Comprehensive Mine Simulator at Ft. Belvoir, VA); c) adapting WebFlow to support visual HLA simulation authoring. We refer to such Pragmatic Object Web based interactive simulation environment as WebHLA [8] and we believe that it will soon offer a powerful modeling and simulation framework, capable to address the new challenges of DoD computing in the areas of Simulation Based Design, Testing, Evaluation and Acquisition.

References

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

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

[3] 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

[4] 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).

[5] Geoffrey Fox and Wojtek Furmanski, "Use of Commodity Technologies in a Computational Grid", chapter in book to be published by Morgan-Kaufmann and edited by Carl Kesselman and Ian Foster.

[6] Geofrey C. Fox, Wojtek Furmanski and Hasan T. Ozdemir, "JWORB - Java Web Object Request Broker for Commodity Software based Visual Dataflow Metacomputing Programming Environment", NPAC Technical Report, Feb 98, http://tapetus.npac.syr.edu/ iwt98/pm/documents/

[7] G.C.Fox, W. Furmanski and H. T. Ozdemir, "Java/CORBA based Real-Time Infrastructure to Integrate Event-Driven Simulations, Collaboration and Distributed Object/Componentware Computing", In Proceedings of Parallel and Distributed Processing Technologies and Applications PDPTA '98, Las Vegas, Nevada, July 13-16, 1998, http://tapetus.npac.syr.edu/ iwt98/pm/documents/

[8] David Bernholdt, Geoffrey Fox and Wojtek Furmanski, B. Natarajan, H. T. Ozdemir, Z. Odcikin Ozdemir and T. Pulikal, "WebHLA - An Interactive Programming and Training Environment for High Performance Modeling and Simulation", In Proceedings of the DoD HPC 98 Users Group Conference, Rice University, Houston, TX, June 1-5 1998, http://tapetus.npac.syr.edu/ iwt98/pm/documents/

[9] 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), http://tapetus.npac.syr.edu/ iwt98/pm/documents/

[10] 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). http://trurl.npac.syr.edu/tango

[11] Tango Collaboration System, http://trurl.npac.syr.edu/tango

[12] Habanero Collaboration System, http://www.ncsa.uiuc.edu/ SDG/Software/Habanero

[13] Erol Akarsu, Geoffrey Fox, Wojtek Furmanski, Tomasz Haupt, "WebFlow - High-Level Programming Environment and Visual Authoring Toolkit for High Performance Distributed Computing", paper submitted for Supercomputing 98, http://www.npac.syr.edu/ users/haupt/ALLIANCE/sc98.html

[14] Ian Foster and Carl Kessleman, Globus Metacomputing Toolkit, http://www.globus.org