# Distributed Systems on the Pragmatic Object Web - Computing with Java and CORBA

G.C. Fox, W. Furmanski and T. Haupt

Northeast Parallel Architectures Center, Syracuse University, Syracuse NY, USA
gcf@npac.syr.edu, furm@npac.syr.edu and haupt@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. We illsutrate our approach on a set of prototype applications in areas such as collaborative systems, support of multidisciplinary interactions, WebFlow based visual metacomputing, WebFlow over Globus, 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 tech-

nologies 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

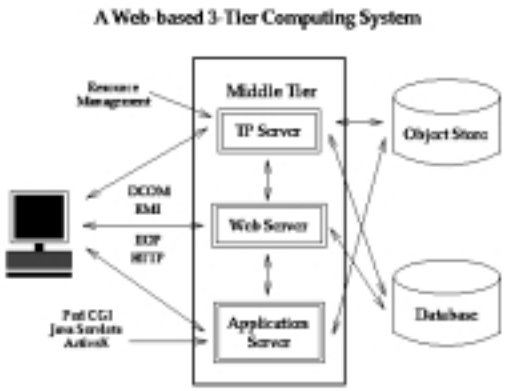


Figure 1: *Industry 3-tier view of enterprise Computing*

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. Actually the "thin client" is favored in consumer markets bue to cost and in corporations due to the greater ease of managing (centralized) server compared to (chaotic distributed) client systems.

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 four major commercial standards discussed above. 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 objects as in the new JINI architecture from Sun Microsystems.

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-¿CORBA3, 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 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/JINI distributed object model integrated with the web.

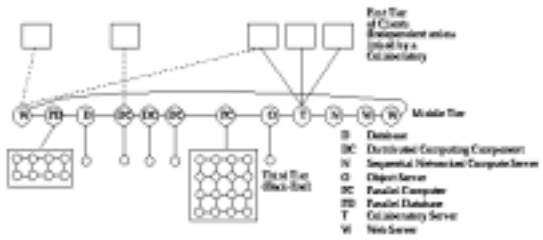

Figure 2: *Today's Heterogeneous Interoperating Hybrid Server Architecture. HPcc involves adding to this system, high performance in the third tier.*

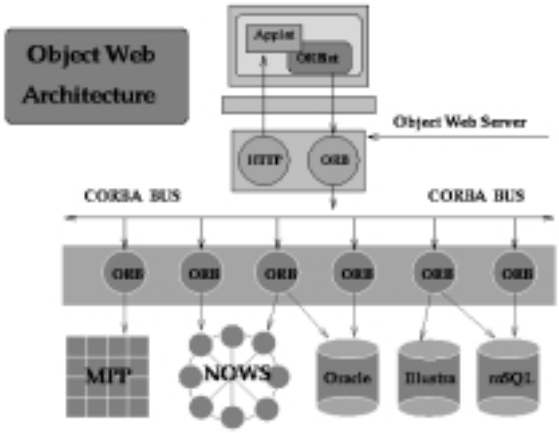


Figure 3: *Integration of Object Technologies (CORBA) and the Web*

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.

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 reengineering 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. 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 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 Multidisciplinary 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 complex control and service provision capability inherited naturally from the Web.

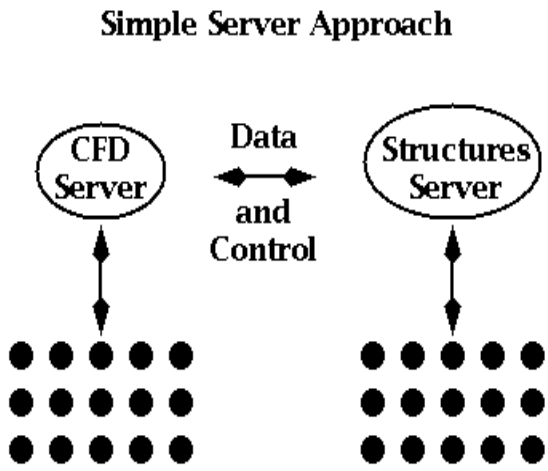


Figure 4: *Simple sequential server approach to Linking Two Modules*

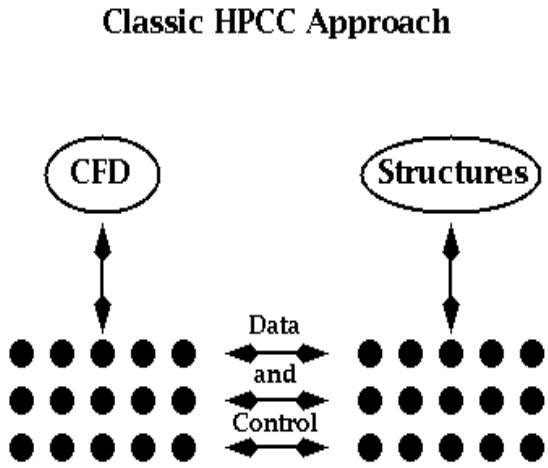


Figure 5: *Full HPCC approach to Linking Two Modules*

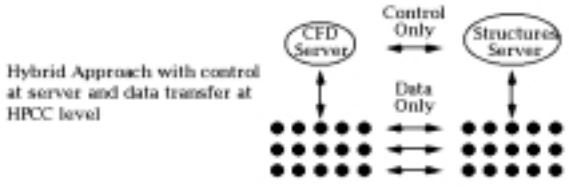


Figure 6: *Hybrid approach to Linking Two Modules*

## 4.2 JavaBean Communication Model

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.

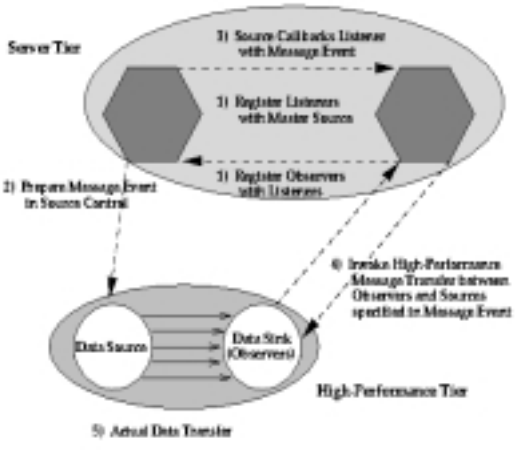


Figure 7: *JDK 1.1 Event Model used by (inter alia) Javabeans*

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 function-

ality 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 JWORB based Middleware

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 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 (Common Data Representation) 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 described above and by registering a new protocol with the JWORB server.

After JWORB accepts a connection, it asks each

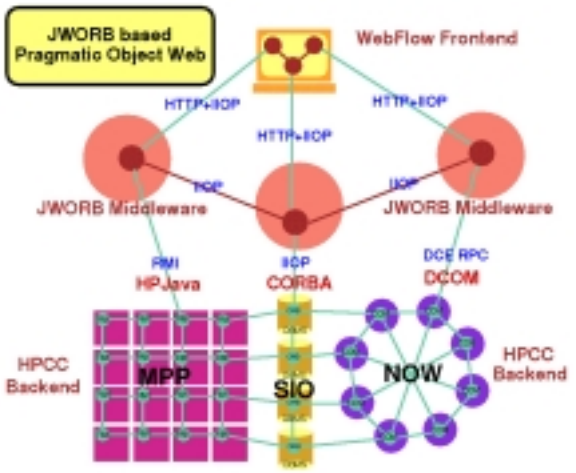


Figure 8: *Overall architecture of the JWORB based Pragmatic Object Web middleware*

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.

We tested the performance of the IIOP channel by echoing an array of integers and structures that contains only one integer value. We performed 100 trials for each array size and we got an average of these measurements. In these tests, client and server objects were running on two different machines. Since we only finished the server side support, we used JacORB on the client side to conduct the necessary tests for the current JWORB.

The timing results presented in Figs. 9-11 indicate that that JWORB performance is reasonable when compared with other ORBs even though we haven't invested yet much time into optimizing the IIOP communication channel. The ping value for various ORBs is the range of 3-5 msecs which is consistent with the timing values reported in the Orfali and Harkey book [1]. However, more study is

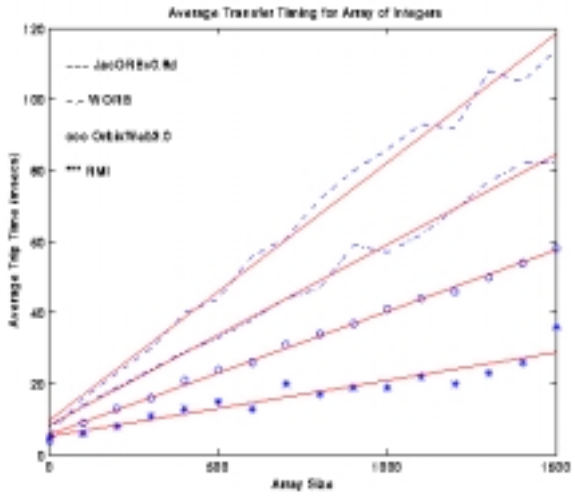


Figure 9: *IIOP communication performance for variable size integer array transfer by four Java ORBs: JacORB, JWORB, OrbixWeb and RMI. As seen, initial JWORB performance is reasonable and further optimizations are under way. RMI appears to be faster here than all IIOP based models.*

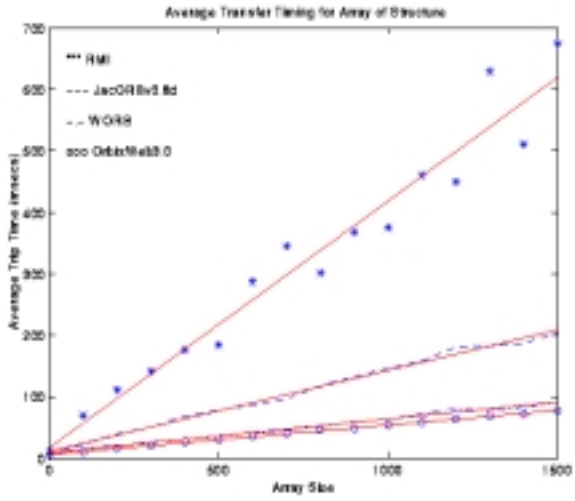


Figure 10: *IIOP communication performance for transferring a variable size array of structures by four Java ORBs: JacORB, JWORB, OrbixWeb and RMI. Poor RMI performance is due to the object serialization overhead, absent in the IIOP/CDR protocol.*

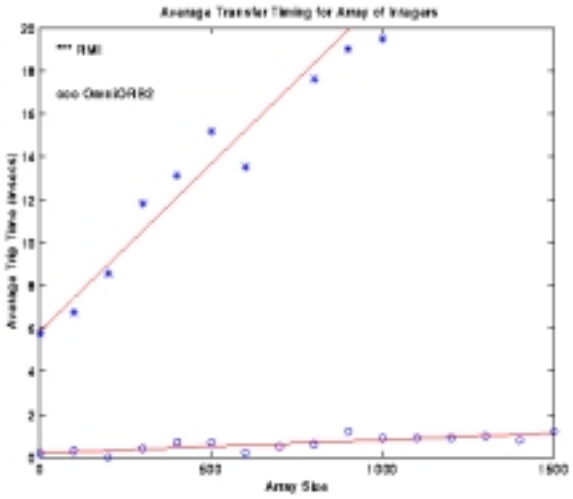


Figure 11: *Initial performance comparison of a C++ ORB (omniORB) with the fastest (for integer arrays) Java ORB (RMI). As seen, C++ outperforms Java when passing data between distributed objects by a factor of 20.*

needed to understand detailed differences between the slopes for various ORBs. One reason for the differences is related to the use of Java object serialization by RMI. In consequence, each structure transfer is associated with creating a separate object and RMI performs poorly for arrays of structure. JacORB uses object serialization also for arrays of primitive types and hence its performance is poor on both figures.

We are currently doing a more detailed performance analysis of various ORBs, including C/C++ ORBs such as omniORB2 or TAO that is performance optimized for real time applications. We will also compare the communication channels of various ORBs with the true high performance channels of PVM, MPI and Nexus. It should be noted that our WebFlow metacomputing is based on Globus/Nexus [14] backend (see next Section) and the associated high performance remote I/O communication channels wrapped in terms of C/C++ ORBs (such as omniORB2). However the middleware Java based ORB channels will be used mainly for control, steering, coordination, synchronization, load balancing and other distributed system services. This control layer does not require high bandwidth and it will benefit from the high functionality and quality of service offered by the CORBA model.

Initial performance comparison of a C++ ORB (omniORB2) and a Java ORB (RMI) indicates that

C++ outperforms Java by a factor of 20 in the IIOP protocol handling software. The important point here is that both high functionality Java ORB such as JWORB and high performance C++ ORB such as omniORB2 conform to the common IIOP standard and they can naturally cooperate when building large scale 3-tier metacomputing applications.

So far, we have got the base IIOP engine of the JWORB server operational and we are now working on implementing the client side support, Interface Repository, Naming Service, Event Service and Portable Object Adapter.

# 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 Mechanisms

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.

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 CORBA facilities such as Work

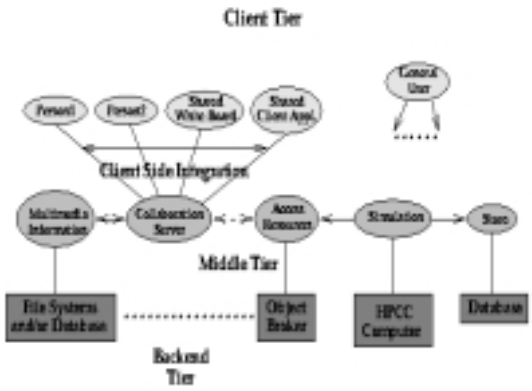


Figure 12: *Collaboration in today's Java Web Server implementation of the 3 tier computing model. Typical clients (on top right) are independent but Java collaboration systems link multiple clients through object (service) sharing*

Flow imply approaches to collaboration, they 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 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 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 frame-

work 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.

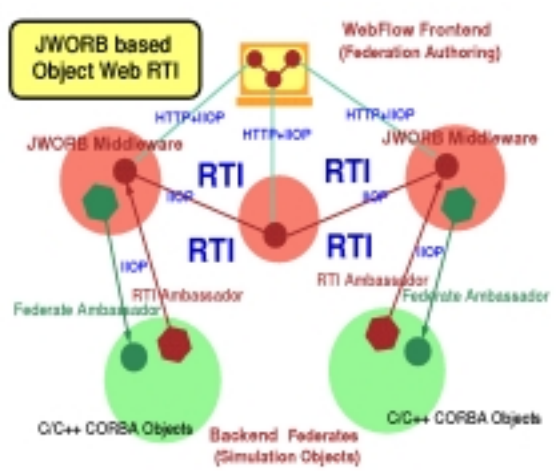


Figure 13: *Overall architecture of the Object Web RTI - a JWORB based RTI prototype recently developed at NPAC*

At NPAC, we are anticipating these developments and 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, 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. 13 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.

Federation Manager object is responsible for the life cycle of the Federation Execution. Each execution creates a different FederationExecutive and this object keeps track of all federates that joined this Federation.

Object Manager is responsible for creating and registering objects/interactions related to simulation. Federates register the simulated object instances with the Object Manager. Whenever a new registration/destroy occurs, the corresponding event is broadcast to all federates in this federation execution.

Declaration Manager is responsible for the subscribe/publish services for each object and its attributes. For each object class, a special object class record is defined which keeps track of all the instances of this class created by federates in this federation execution. This object also keeps a seperate broadcasting queue for each attribute of the target object so that each federate can selectively subscribe, publish and update suitable subsets of the object attributes.

Each attribute is currently owned by only one federate who is authorized for updating this attribute value. All such value changes are reflected via RTI in all other federates. Ownership Management offers services for transfering,maintaining and querying the attribute ownership information.

Individual federatesa can follow different time management frameworks ranging from time-stepped/real-time to event-driven/logical time models. Time Management service offers mechanisms for the federation-wide synchronization of the

local clocks, advanced and managed by the individual federates.

Data Distribution Management offers advanced publish/subscribe based communication services via routing spaces or multi-dimensional hypercube regions in the attribute value space.

In parallel with the first pass prototoype implementation, we are also addressing the issues of more organized software engineering in terms of Common CORBA Services. For example, we intend to use the CORBA Naming Service to provide uniform mapping between the HLA object names and handles, and we plan to use CORBA Event and Notification Services to support all RTI broadcast/multicast mechanisms. This approach will assure quality of service, scalability and fault-tolerance in the RTI domain by simply inheriting and reusing these features, already present in the CORBA model.

### 5.3 Commodity Cluster Management

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 and providing advanced services such as time management or distributed event driven simulation kernels.

From the Pragmatic Object Web, defined as a merger of CORBA, Java, COM and WOM, we need some uniform cohesive force that could combine various competing commodity standards towards a cooperative whole. At the core middleware level, this is realized by our multi-protocol JWORB server, but we also need some uniform framework to integrate higher level services coming from various commodity frameworks.

In our emergent WebHLA environment, we view HLA/RTI as a potential candidate for such a uniform high level service framework. In fact, the WebHLA application domains discussed in [15] can be viewed as various attempts at extending RTI beyond the original Modeling and Simulation domain towards collaborative training metacomputing resource management, or commodity cluster management.

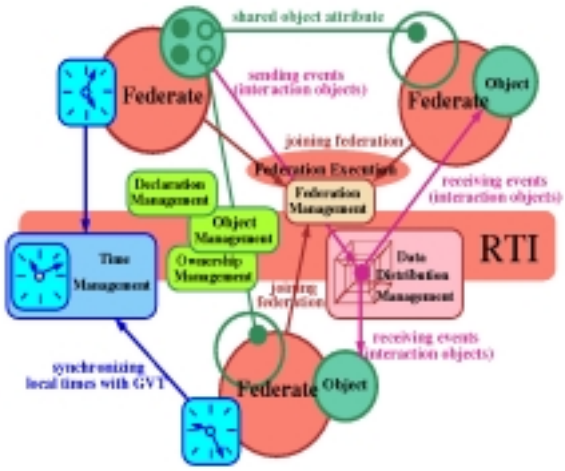


Figure 14: *Distributed object based architecture of DMSO RTI - originally constructed for the Modeling and Simulation domain but naturally extensible for other distributed computing management services such as cluster, metacomputing or collaboration management discussed in the text.*

Indeed, as illustrated in Fig. 14, RTI can be viewed as a high level abstraction of a distributed operating system with machines / nodes represented as federates, clusters as federations, with time management responsible for job scheduling, ownership management linked with security and so on. We are currently starting a project with Sandia National Laboratories which will explore RTI as such a high level operating and control framework for the Sandia's new growable commodity cluster technology called CPlant and developed within the DoE ASCI Defense Program.

### 5.4 Visual 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, 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.

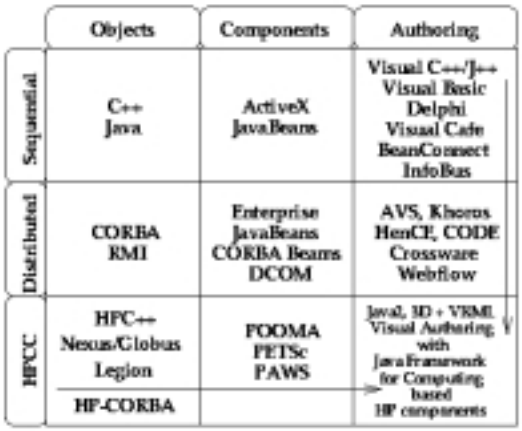


Figure 15: *System Complexity (vertical axis) versus User Interface (horizontal axis) tracking of some technologies*

In fig. 15, 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 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] 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. 17 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.

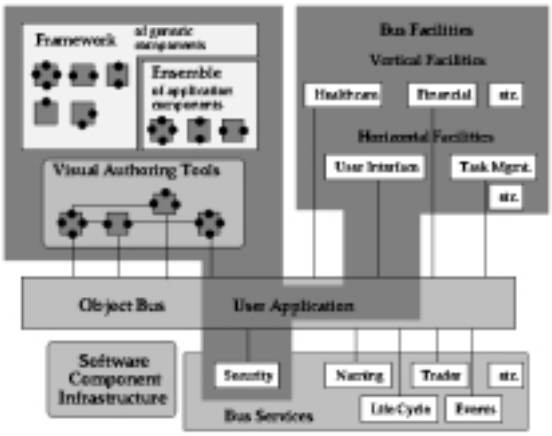


Figure 16: *Visual Authoring with Software Bus Components*

We then generate an environment such as fig. 16 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. 16, 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.

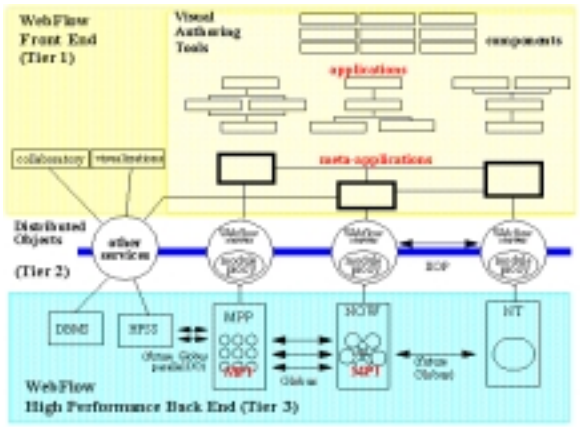


Figure 17: *Top level view of the WebFlow environment with JWORB middleware over Globus metacomputing or NT cluster backend*

## 5.5 Current WebFlow Prototype

We describe here a specific high-level programming environment developed by NPAC - WebFlow [9] - that addresses the visual componentware programming issues discussed above and offers a user friendly visual graph authoring metaphor for seamless composition of world-wide distributed high performance dataflow applications from reusable computational modules.

Design decisions of the current WebFlow were made and the prototype development was started in '96. Right now, the system is reaching some initial stability and is associated with a suite of demos or trial applications which illustrate the base concepts and allow us to evaluate the whole approach and plan the next steps for the system evolution. New technologies and concepts for Web based distributed computing appeared or got consolidated during the last two years such as CORBA, RMI, DCOM or WOM. In the previous Chapters, we summarized our ongoing work on the integration of these competing new distributed object and componentware technologies towards what we call Pragmatic Object Web [16]. To the end of this Chapter, we present the current WebFlow system, its applications and the lessons learned in this experiment. While the implementation layers of the current (pure Java Web Server based) and the new (JWORB based) WebFlow models are different, several generic features of the system are already established and will stay intact while the implementation technologies are evolving. We present here an overview of the system vision and goals which exposes these stable generic characteristics of WebFlow.

Our main goal in WebFlow design is to build a seamless framework for publishing and reusing computational modules on the Web so that the end-users, capable of surfing the Web, could also engage in composing distributed applications using WebFlow modules as visual components and WebFlow editors as visual authoring tools. The success and the growing installation base of the current Web seems to suggest that a suitable computational extension of the Web model might result in such a new promising pervasive framework for the wide-area distributed computing and metacomputing.

In WebFlow, we try to construct such an analogy between the informational and computational aspects of the Web by comparing Web pages to WebFlow modules and hyperlinks that connect Web pages to inter-modular dataflow channels. WebFlow content developers build and publish modules by attaching them to Web servers. Application integrators use visual tools to link outputs of the source modules with inputs of the destination modules, thereby forming distributed computational graphs (or compute-webs) and publishing them as composite WebFlow modules. Finally, the end-users simply activate such compute-webs by clicking suitable hyperlinks, or customize the computation either in terms of available parameters or by employing some high-level commodity tools for visual graph authoring.

New element of WebFlow as compared with the current "vertical" instances of the computational Web such as CGI scripts, Java applets or ActiveX controls is the "horizontal" multi-server intermodular connectivity (see Fig. 18), specified by the compute-web graph topology and enabling concurrent world-wide data transfers, either transparent to or customizable by the end-users depending on their preferences. Some examples of WebFlow computational topologies include: a) ring - postprocessing an image by passing it through a sequence of filtering (e.g. beautifying) services located at various Web locations; b) star - collecting information by querying a set of distributed databases and passing each output through a custom filter before they are merged and sorted according to the end-user preferences; c) (regular) grid - a

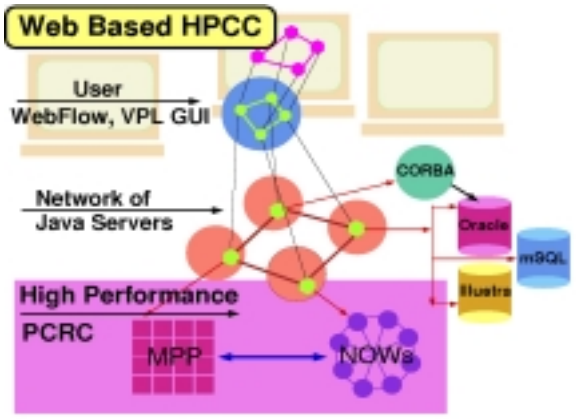


Figure 18: *Top view of the WebFlow system: its 3-tier design includes Java applet based visual graph editors in tier 1, a mesh of Java servers in tier 2 and a set of computational (HPC, Database) modules in tier 3.*

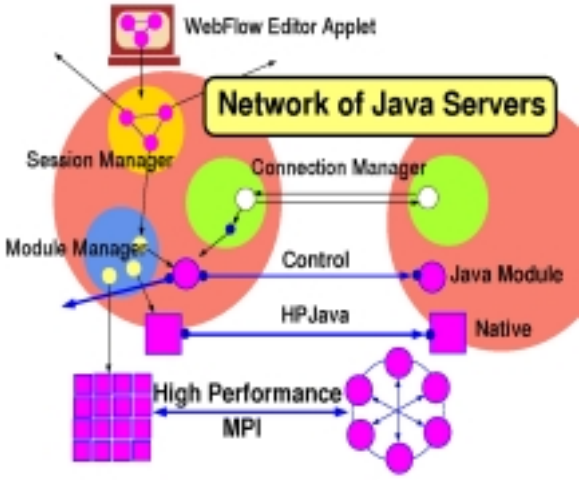


Figure 19: *Overall Architecture of the 3-tier WebFlow model with the visual editor applet in tier-1, a mesh of Java Web Servers in tier 2 (including WebFlow Session Manager, Module Manager and Connection Manager servlets), and (high performance) computational modules in tier-3.*

large scale environmental simulation which couples atmosphere, soil and water simulation modules, each of them represented by sub-meshes of simulation modules running on high performance workstation clusters; d) (irregular) mesh - a wargame simulation with dynamic connectivity patterns between individual combats, vehicles, fighters, forces, environment elements such as terrain, weather etc.

When compared with the current Web and the coming Mobile Agent technologies, WebFlow can be viewed as an intermediate/transitional technology - it supports a single-click automation/aggregation for a collection of tasks/modules forming a compute-web (where the corresponding current Web solution would require a sequence of clicks), but the automation/aggregation patterns are still deterministic, human designed and manually edited (whereas agents are expected to form goal driven and hence dynamic, adaptable and often stochastic compute-webs).

Current WebFlow is based on a coarse grain dataflow paradigm (similar to AVS or Khoros models) and it offers visual interactive Web browser based interface for composing distributed computing (multi-server) or collaboratory (multi-client) applications as networks (or compute-webs) of Internet modules.

WebFlow front-end editor applet offers intuitive click-and-drag metaphor for instantiating middleware or backend modules, representing them as visual icons in the active editor area, and interconnecting them visually in the form of computational graphs, familiar for AVS or Khoros users.

WebFlow middleware is given by a mesh of Java Web Servers, custom extended with servlet based support for the WebFlow Session, Module and Connection Management. WebFlow modules are specified as Java interfaces to computational Java classes in the middleware or wrappers (module proxies) to backend services (Fig. 19).

To start a WebFlow session over a mesh of the WebFlow enabled Java Web Server nodes, user specifies URL of the Session Manager servlet, residing in one of the server nodes (Fig. 20). The server returns the WebFlow editor applet to the browser and registers the new session. After a connection is established between the Editor and the Session Manager, the user can initiate the compute-web editing work. WebFlow GUI includes the following visual actions:

- Selecting a module from the palette and placing its icon in the active editor area. This results in passing this module tag to the Session Manager that forwards it to the Module Manager. Module Manager instantiates the module and it passes its communication ports to the Connection Manager.
- Linking two visual module icons by drawing a connection line. This results in passing the connected modules tags to the Ses-


Figure 20: *Architecture of the WebFlow server: includes Java servlet based Session, Module and Connection Managers responsible for interacting with front-end users, backend modules and other WebFlow servers in the middleware.*

sion Manager, and from there to the Connection Managers in charge of these module ports. WebFlow channels are formed dynamically by Connection Managers who create the suitable socket connections and exchange the port numbers. After all ports of a module receive their required sockets, the module notifies the Module Manager and is ready to participate in the dataflow operations.

- Pressing the Run button to activate the WebFlow computation
- Pressing the Destroy button to stop the WebFlow computation and dismantle the current compute-web.

WebFlow Module is a Java Object which implements webflow.Module interface with three methods: init(), run() destroy(). The init() method returns the list of input and output ports used to establish inter-modular connectivity, and the run() and destroy() methods are called in response to the corresponding GUi actions described above.

### 5.6 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. 21 and 22 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.

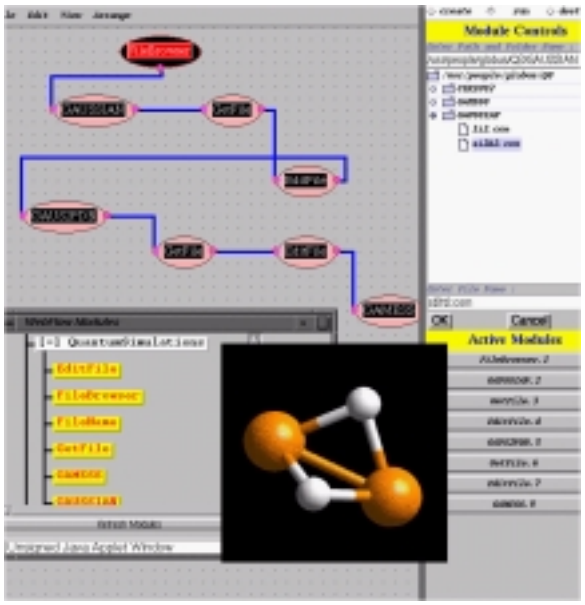


Figure 21: *Screendump of an example WebFlow session: running Quantum Simulations on a virtual metacomputer. Module GAUSSIAN is executed on Convex Exemplar at NCSA, module GAMESS is executed on SGI Origin2000, data format conversion module is executed on Sun SuperSparc workstation at NPAC, Syracuse, 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.

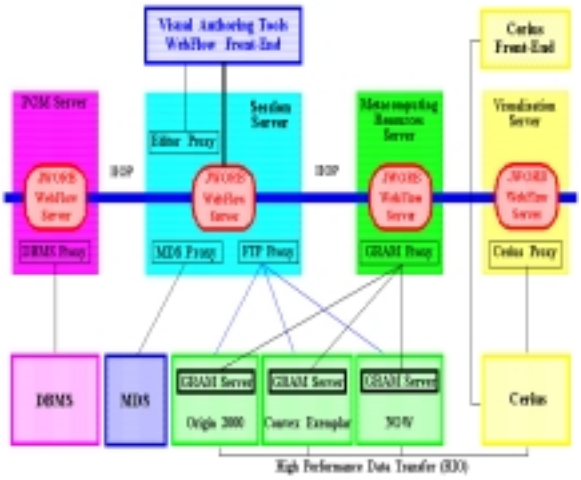


Figure 22: *WebFlow implementation of the Quantum Simulations problem*

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 [15] 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.

We illustrate here our WebHLA approach on the example of using WebFlow for visual HLA authoring. DMSO has emphasized the need to develop automated tools with open architectures for creating, executing and maintaining HLA simulations and federations. The associated Federation Development Process (FEDEP) guidelines enforce interoperability in the tool space by standardizing a set of Data Interchange Formats (DIF) that are mandatory as input or output streams for the individual HLA tools. In consequence, one can envision a high-level user friendly e.g. visual dataflow authoring environment in which specialized tools can be easily assembled interactively in terms of computational graphs with atomic tool components as graph nodes and DIF-compliant communication channels as graph links (Fig. 23)

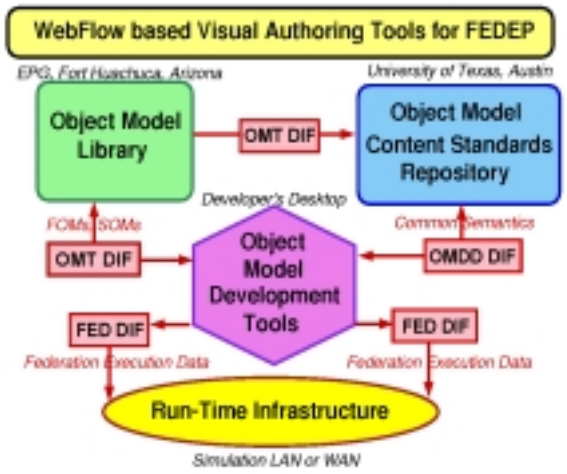


Figure 23: *WebFlow based representation of DMSO FEDEP: individual FEDEP tools are mapped on WebFlow modules and the DIF files are mapped on WebFlow communication channels.*

Within our HPCMP FMS PET project at NPAC we are building such visual HLA tool assembly framework [17] on top of the NPAC WebFlow system. We started this project by analyzing currently existing tools in this area. In particular, we inspected the Object Model Development Tool (OMDT) by Aegis Research Center, Huntsville, AL as a representative current generation DMSO FEDEP tool. OMDT is a Windows 95/NT-based application that supports building HLA Object Model Template (OMT) tables such as Class, Interaction or Attribute Tables using a spreadsheet-like user. We found OMDT useful in the standalone mode but not yet ready to act as a standardized reusable component in larger toolkits. We therefore decided to build an OMDT-like editing tool based

on Microsoft Component Object Model (COM) architecture.

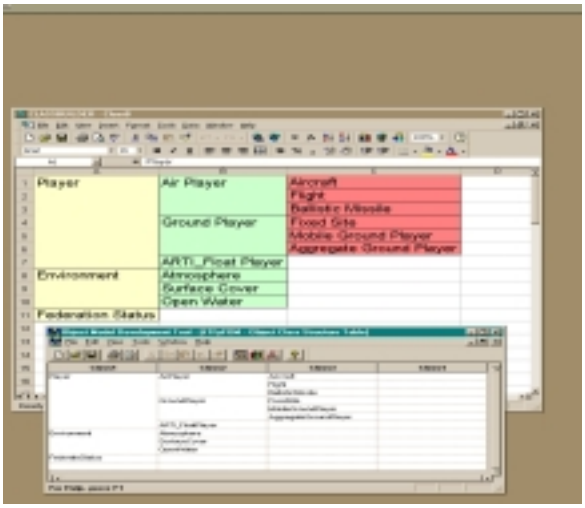


Figure 24: *OMDT by Aegis Corporation (front window) compared with our OMBuilder (back window): both tools have similar functionality and look-and-feel but OMBuilder is constructed via VBA scripting on top of Microsoft Excel.*

Rather than building our sample tool from scratch, we construct it by customizing Microsoft Excel Component using the Visual Basic for Applications and the OLE automation methodology. Using this approach, we were able to emulate the look-and-feel of the OMDT tool, while at the same time packaging our tool, called OMBuilder, as a reusable COM or ActiveX component that can smoothly cooperate with other visual authoring tools within the WebFlow model (Fig. 24). We also extended the OMDT functionality in OMBuilder by adding support for initializing and runtime steering the simulation attributes and parameters.

Next, we constructed a proof-of-the-concept demo that offers WebFlow and OMBuilder based visual authoring toolkit for the Jager game (available as part of the DMSO RTI distribution). Each Jager player (both human and robot) is represented by a suitable visual icon in the WebFlow editor. A particular game scenario is constructed by selecting the required player set and registering / connecting them as nodes of a suitable connected graph (e.g. ring in the lower left corner in Fig 25). We also constructed a 3D viewer, operating in parallel with the standard 2D Jager viewer and used for experiments with parameter initialization via the extended OMBuilder editor (Fig 26).

In parallel with prototyping core WebHLA tech-

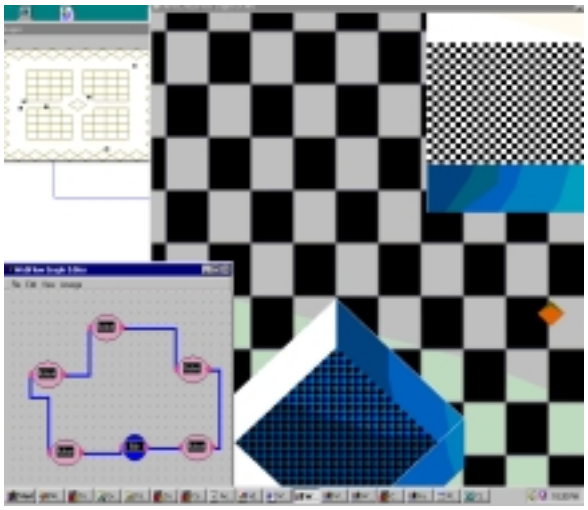

Figure 25: *Proof-of-the-concept visual authoring tools for Jager: WebFlow editor (lower left corner) allows to specify the configuration for the Jager game players. Both the original 2D (upper left corner) and our trial 3D viewer s are displayed.*

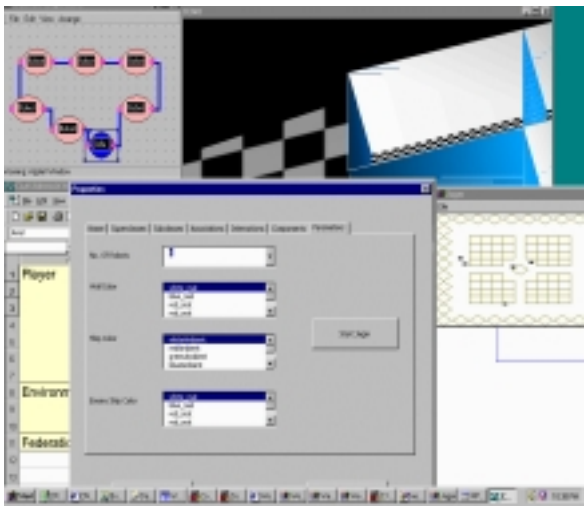

Figure 26: *Proof-of-the-concept visual authoring tools for Jager: a runtime extension of the OMBuilder tool allows the user to initialize, monitor and steer the simulation or rendering parameters.*

nologies described above, we are also analyzing selected large scale Modeling and Simulatin applications that could be used as high performance simulation modules in tier-3 of our framework. In particular, we were recently studying the CMS (Comprehensive Mine Simulator) system developed by Steve Bishop's team at Ft. Belvoir, VA that simulates mines, mine fields, minefield components, standalone detection systems and countermine systems including ASTAMIDS, SMB and MMCM. The system can be used to enable a virtual prototyping tool in the area of new countermine systems and detection technologies of relevance both for the Army and the Navy. We are currently analyzing the CMS source code and planning the parallel port of the system to Origin2000.

The CMS system, when viewed as an HLA federation, decomposes naturally into the minefield and vehicle (tanks, contermines etc.) federates. Each of these federates might require high fidelity HPC simulation support (e.g. for large minefields of millions mines, or for the engineering level countermine simulation), whereas their interactions (vehicle motion, mine detonation) requires typically only a low-to-medium bandwith. Hence, the system admits a natural metacomputing implementation, with the individual federates simulated on the HPC facilities at the geographically distributed MSRCs and/or DCs, glued together and time-synchronized using the Object Web RTI discussed in Sect 2.1.

We are currently in the planning stage of such a metacomputing FMS experiment, to be conducted using ARL (Maryland) and CEWES (Mississippi) MSRC and perhaps also NRaD / SPAWAR (California) and NVLD (Virginia) facilities in '99.

We are also participating in the new FMS project aimed at developing HPC RTI for Origin2000 that will provide useful HPC infrastructure for the metacomputing level FMS simulations. Fig. 27 illustrates the natural relative interplay between the DMSO RTI (most suitable for local networks), HPC RTI (to run on dedicated HPC systems) and Object Web RTI (useful for wide-area integration and real-time control via the Web or CORBA channels).

Fig 28 illustrates our envisioned end product in the WebHLA realm - a distributed, Web / Commodity based, high performance and HLA compliant Virtual Prototyping Environment for Simulation Based Acquisition with Object Web RTI based software bus, integrating a spectrum of Modeling and Simulation tools and modules, wrapped as commodity (CORBA or COM) components and accessible via interactive Web browser front-ends.

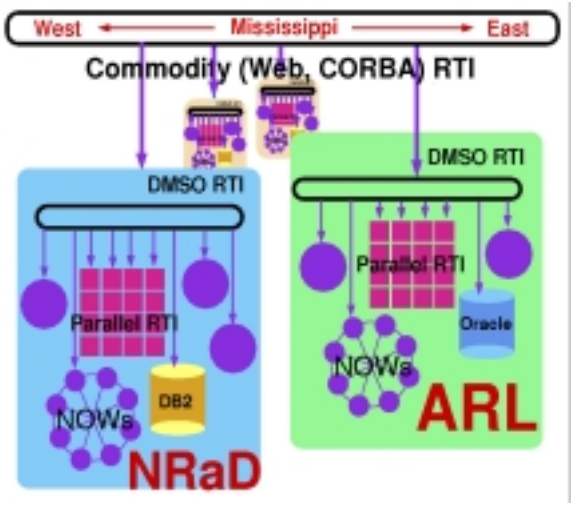


Figure 27: *Illustration of the interplay between DMSO RTI (running on Intranets), Parallel RTI (running on HPC facilities) and Commodity (such as Object Web) RTI. The latter is running in the Web / Internet Domain and connects geographically distributed MSRCs and DCs.*

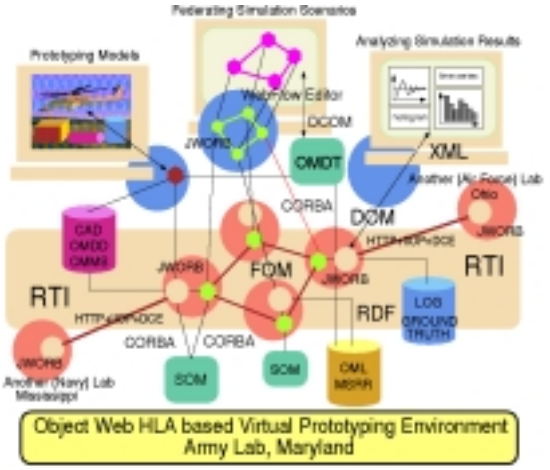


Figure 28: *Overall architecture of a planned WebHLA based Virtual Prototyping Environment for Simulation Based Acquisition: Object Web RTI acts as a universal software bus, managing a spectrum of resources for simulation, clustering, collaboration, training, data mining, data warehousing etc.*

Such environments, currently operated only by large industry such as Bowing, become affordable within the current technology convergence process as envisioned in Fig. 1 and quantified in our WebHLA integration program

Indeed, the challenge of SBA is to successfully integrate Modeling and Simulaton, Test and Evaluation, High Performance Computing, Metacomputing, Commodity software for Real-Time Multimedia front-ends and Database back-ends, Collaboration, Resource Management and so on - these capabilities represent individual WebHLA components which are being now prototyped in a coordinated, Pragmatic Object Web based development and integration framework.

## 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

[15] G. C. Fox, W. Furmanski, S. Nair, H. T. Ozdemir, Z. Odcikin Ozdemir and T. A. Pulikal, "WebHLA - An Interactive Programming

and Training Environment for High Performance Modeling and Simulation", to be published in Proceedings of the Simulation Interoperatiliby Workshop SIW Fall 1998, Orlango FL, Sept 14-18, 1998.

[16] G. C. Fox, W. Furmanski, H. T. Ozdemir and S. Pallickara, "Building Distributed Systems on the Pragmatic Object Web", Wiley '99 book in progress.