New Systems Technologies and Software Products for HPCC: Volume III - High Performance Commodity Computing on the Pragmatic Object Web

Geoffrey C. Fox, Wojtek Furmanski, Hasan T. Ozdemir and Shrideep Pallickara

Northeast Parallel Architectures Center, Syracuse University
111 College Place, Syracuse University, Syracuse NY 13244-4100
{gcf, furm, timucin, shrideep} @ npac.syr.edu

Technology Assessment Report for RCI, Ltd., October 1998

Abstract

In this paper, we describe an approach to high performance computing which makes extensive use of commodity technologies. In particular, we exploit new Web technolgies such as XML, CORBA and COM based distributed objects and Java. The use of commodity hardware (workstation and PC based MPP's) and operating systems (UNIX, Linux and Windows NT) is relatively well established. We propose extending this strategy to the programming and runtime environments supporting developers and users of both parallel computers and large scale distributed systems. We suggest that this will allow one to build systems that combine the functionality and attractive user environments of modern enterprise systems with delivery of high performance in those application components that need it. Critical to our strategy is the observation that HPCC applications are very complex but typically only require high performance in parts of the problem. These parts are dominant when measured in terms of compute cycles or data-points but often a modest part of the problem if measured in terms of lines of code or other measures of implementation effort. Thus rather than building such systems heroically from scratch, we suggest starting with a modest performance but user friendly system and then selectively enhancing performance when needed. In particular, we view the emergent generation of distributed object and component technologies as crucial for encapsulating performance critical software in the form of reusable plug-and play modules. We review here commodity approaches to distributed objects by four major stakeholders: Java by Sun Microsystems, CORBA by Object Management Group, COM by Microsoft and XML by the World-Wide Web Consortium. Next, we formulate our suggested integration framework called Pragmatic Object Web in which we try to mix-and-match the best of Java, CORBA, COM and XML and to build a practical commodity based middleware and front-ends for today's high performance computing backends. Finally, we illustrate our approach on a few selected application domains such as WebHLA for Modeling and Simulation and Java Grande for Scientific and Engineering Computing.

1. Introduction

In this paper, we describe an approach to high performance computing which makes extensive use of commodity technologies. In particular, we exploit Web technology, distributed objects and Java. The use of commodity hardware (workstation and PC based MPP's) and operating systems (UNIX, Linux and Windows NT) is relatively well established. We propose extending this strategy to the programming and runtime environments supporting developers and users of both parallel computers and large scale distributed systems. We suggest that this will allow one to build systems that combine the functionality and attractive user environments of modern enterprise systems with delivery of high performance in those application components that need it. Critical to our strategy is the observation that HPCC applications are very complex but typically only require high performance in parts of the problem. These parts are dominant when measured in terms of compute cycles or data-points but often a modest part of the problem if measured in terms of lines of code or other measures of implementation effort. Thus rather than building such systems heroically from scratch, we suggest starting with a modest performance but user friendly system and then selectively enhancing performance when needed.

In section 2, we describe key relevant concepts that are emerging in the innovative technology cauldron induced by the merger of multiple approaches to distributed objects and Web system technologies. This cauldron is largely fueled by development of corporate Intranets and broad based Internet applications including electronic commerce and multimedia. We define the "Pragmatic Object Web" approach which recognizes that there is not a single "best" approach but several co-existing technology bundles within an object based web. In particular, CORBA (Corporate Coalition), COM (Microsoft), Javabeans/RMI (100% pure Java), and XML/WOM/DOM (from World Wide Web Consortium) have different tradeoffs. One can crudely characterize them as the most general, the highest performance, the most elegant and simplest distributed object models respectively. This merger of web and distributed object capabilities is creating remarkably powerful distributed systems architecture.

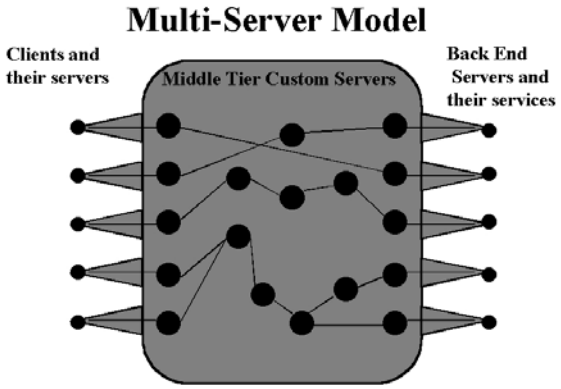

Figure 1.1: *Continuum tier multi-server model*

However the multiple standards -- each with critical capabilities -- implies that one cannot choose a single approach but rather must pragmatically pick and choose from diverse inter-operating systems. Another key community concept is that of a multi-tier enterprise system where one no longer expects a simple client server system. Rather clients interact with a distributed system of servers, from which information is created by the interactions of modular services, such as the access to and filtering of data from a database. In the Intranet of a modern corporation, these multiple servers reflected both the diverse functionality and geographical distribution of

the components of the business information ecosystem.

It has been estimated that typical Intranets support around 50 distinct applications whose integration is an area of great current interest. This middle tier of distributed linked servers and services gives new Operating System challenges and is current realization of the WebWindows concept we described in an earlier RCI article [X]. Note that Java is often the language of choice for building this tier but the object model and communication protocols can reflected any of the different standards CORBA, COM, Java or WOM. The linked continuum of servers shown in Figure 1.1, reflects the powerful distributed information integration capabilities of the Pragmatic Object Web.

In Section 3, we present our basic approach to achieving high performance within the pragmatic object web (POW) multi-tier model. We make the simple observation that high performance is not always needed but rather that one needs hybrid systems combining modest performance high functionality components of the commodity Intranet with selected high performance enhancements. We suggest a multi-tier approach exploiting the separation between invocation and implementation of a data transfer that is in fact natural in modern publish-subscribe messaging models. We illustrate this HPcc -- High Performance commodity computing -- approach with a Quantum Monte Carlo application integrating NPAC's WebFlow client and middle tier technology with Globus as the back end high performance subsystem.

In the following Sections, we refine these basic ideas. In Section 4, we present the four major players on the Pragmatic Object Web scene – Java, CORBA, COM and WOM. This is followed in section 5 by the discussion of POW integration concepts and prototypes. We describe there the natural POW building block JWORB -- a server written in Java which supports all 4-object models. In particular JWORB builds its Web Services in terms of basic CORBA capabilities. We present performance measurements which emphasis the need to enhance the commodity tier when high performance messaging is needed. We discuss how JWORB allows us to develop HPCC componentware by using the JavaBean visual-computing model on top of this infrastructure. We describe application integration and multidisciplinary problems in this framework.

In section 6, we discuss the new DMSO (Defense Modeling and Simulation Office) HLA (High Level Architecture) and RTI (Run Time Infrastructure) standards. These can naturally be incorporated into HPcc, giving the WebRTI runtime and WebHLA distributed object model. These concepts suggest a novel approach to general metacomputing, built in terms of a coarse grain event based runtime to federate, manage and schedule resources. This defines a HPcc "virtual machine" which is used to define the coarse grain distributed structure of applications.

We return to the HPCC mainstream in Section 7, where we present *Java Grande* - an application of Web technologies to the finer grain aspects of scientific computing - including the use of the Java language to express sequential and parallel scientific kernels.
Section 8 discusses parallel computing in the HPcc framework and Section 9 summarizes general implications for the architecture of a grande computing environment.

2. Pragmatic Object Web and Commodity Systems

2.1 DcciS: Distributed commodity computing and information System

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 Section 2.2, we define what we mean by commodity technologies and explain the different ways that they can be used in HPCC. In Section 2.3, 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 2.4 and 2.5 by more detailed discussion of the HPcc core technologies and high-level services.

In this and related papers [7][8][10][11][12][19], 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 present here our approach at NPAC within the general context of DcciS which is based on integrating several popular distributed object frameworks. We call it *Pragmatic Object Web* and we describe a specific integration methodology based on multi-protocol middleware server, JWORB - *Java Web Object Request Broker*.

2.2 Commodity Technologies for 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" [2] 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 (see Section 6.4.3) are important for scientific visualization;
- Web (including Java applets and ActiveX controls) front-ends provide convenient customizable interoperable user interfaces to HPCC facilities (see Section 7.4);
- 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 (see Section 7);
- 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 (see Section 5.5);

- JavaBeans, COM, CORBA and WOM form the basis of the emerging "object web" which could encourage a growing use of modern object technology (see Section 4);

- 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.

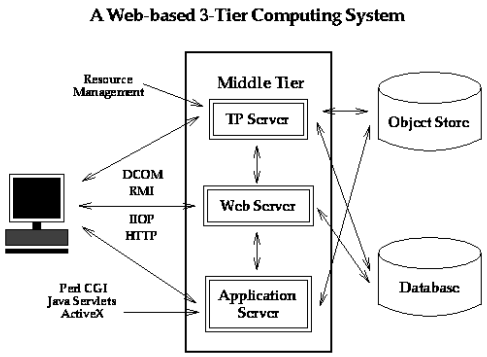

Figure 2.1: *Industry 3-tier view of enterprise Computing*

However probably more important is the strategic impact 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.

2.3 From Three- to Multi-Tier HPcc

We start with a common modern industry view of commodity computing with the three tiers shown in Figure 2.1. 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 due 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[34] 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 [18] 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 [9] 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 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 produce a three tier distributed service model with an HTTP server using a CGI program (running Perl for instance) to access the database at the backend. Today we can build databases as distributed objects with a middle tier JavaBean using JDBC to access the backend database. Thus a conventional database is naturally evolving to the concept of managed persistent objects.

Today as shown in Figure 2.2, we see a mixture of distributed service and distributed object architectures. CORBA, COM, Javabean, HTTP Server + CGI, Java Server and servlets, databases with specialized network accesses, and other services co-exist in the heterogeneous environment with common themes but disparate implementations. We believe that there will be significant convergence as a more uniform architecture is in everyone's best interest.

We also believe that the resultant architecture will be integrated with the web so that the latter will exhibit distributed object architecture shown in Figure 2.3.

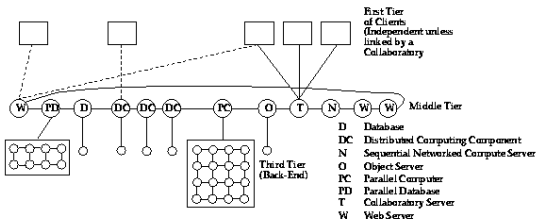

Figure 2.2: *Today's Heterogeneous Interoperating Hybrid Server Architecture. HPcc involves adding to this system, high performance in the third tier.*

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 Figure 2.3 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 browser models and for various 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. 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.

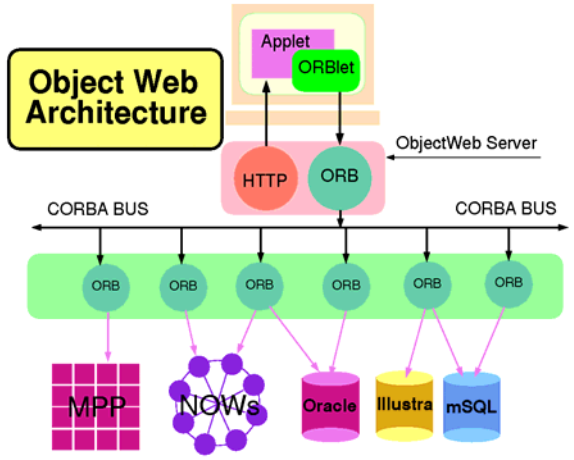

Figure 2.3: *Integration of Object Technologies (CORBA) and the Web*

Thus we exploit the three-tier structure and keep HPCC enhancements in the third tier, which is inevitably the home of specialized services in the object-web architecture. This strategy isolates HPCC issues from the control or interface issues in the middle layer. If successful we will build an HPcc environment that offers the evolving functionality of commodity systems without significant re-engineering as advances in hardware and software lead to new and better commodity products.

Returning to Figure 2.2, we see that it elaborates Figure 2.1 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 Figure 2.3, 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.

2.4 Commodity Services for 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.

2.4.1 Distributed Collaboration Mechanisms

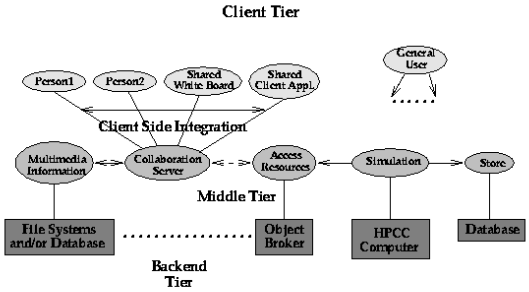

Figure 2.4: *Collaboration in today's Java Web Server implementation of the 3 tier computing model. Typical clients (on top right) are independent but Java collaboration systems link multiple clients through object (service) sharing*

The current Java Server model for the middle tier naturally allows one to integrate collaboration into the computing model and our approach allows one to "re-use" collaboration systems built for the general Web market. Thus one can without any special HPCC development, address areas such as computational steering and collaborative design, which require people to be integrated with the computational infrastructure. In Figure 2.4, 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. If we are correct in viewing collaboration (see Tango [16] and Habanero [17]) as sharing of services between clients, the 3-tier model naturally separates HPCC and collaboration. This 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 workflow 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.

In Section 5.2.3 we illustrate one of POW approaches to collaboration on the example of our JDCE prototype that integrates CORBA and Java/RMI based techniques for sharing remote objects. In Section 6.2.5, we also point out that the HLA/RTI framework for Modeling and Simulation can be naturally adapted for collaboratory applications such as distance training, conducted in a multiplayer real-time interactive gaming framework.

2.4.2 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 longer term 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 DoD HPC Modernization program.

We believe that the integration of Web, Enterprise, Desktop and Defense standards proposed by our Pragmatic Object Web methodology will lead to powerful new generation systems capable to address in affordable way the new computational challenges faced by the DoD such as Virtual Prototyping for Simulation Based Acquisiton.

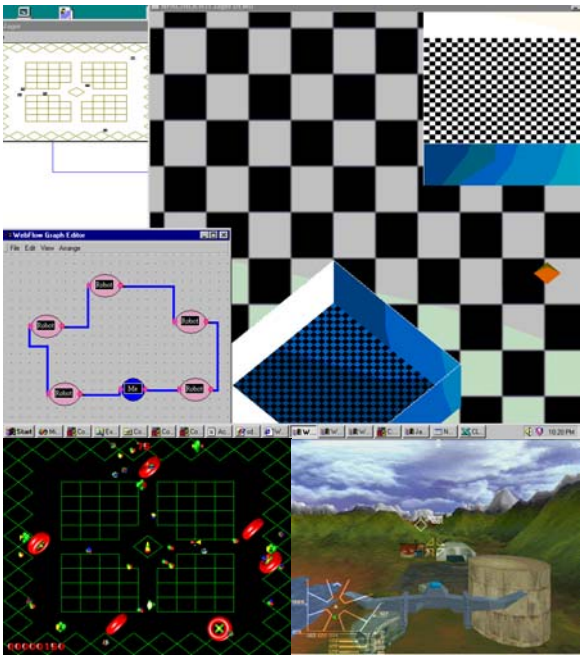

Figure 2.5: *Sample screendumps from the POW based visual authoring (upper frame) and multimedia runtime (lower frames) environment for Jager game – a standard Modeling and Simulation application distriburted by DMSO as part of the RTI release.*

Fig 2.5 illustrates the current snapshot of our work in this area discussed in Section 6. The figure also includes an HLA/RTI application (DMSO Jager) running on top of POW middleware (OWRTI) and integrated first with our WebFlow visual authoring front-end (upper screen) and then with the DirectX multimedia front-ends from Microsoft (lower screens).

2.4.3 Visual Metacomputing

The growing heterogeneous collection of components, developed by the Web / Commodity computing community, offers already now a powerful and continuously growing computational

infrastructure of what we called DcciS – Distributed commodity computing and information System. However, due to the vast volume and multi-language multi-platform heterogeneity of such a repository, it is also becoming increasingly difficult to make the full use of the available power of this software. In our POW approach, we provide an efficient integration framework for several major software trends but the programmatic access at the POW middleware is still complex as it requires programming skills in several languages (C++, Java, XML) and distributed computing models (CORBA, RMI, DCOM). For the end users, integrators and rapid prototype developers, a more efficient approach can be offered via the visual programming techniques. Visual authoring frameworks such as Visual Basic for Windows GUI development, AVS/Khoros for scientific visualization, or UML based Rational Rose for Object Oriented Analysis and Design are successfully tested and enjoy growing popularity in the respective developer communities. Several visual authoring products appeared also recently on the Java developers market including Visual Studio, Visual Age for Java, JBuilder or J++.

HPC community has also explored visual programming in terms of custom prototypes such as HeNCE or CODE, or adaptation of commodity systems such as AVS. At NPAC, we are developing a Web based visual programming environment called WebFlow. Our current prototype summarized below and discussed in detail in Section 5.7 follows the 100% Java model and is currently being extended towards other POW components (CORBA, COM, WOM) as discussed in Sections 5.8 and 5.9.

WebFlow[1][38] is a Java based 3-tier visual data flow programming environment. Front-end (tier-1) is given by a Java applet, which offers interactive graphical tools for composing computational graphs by selecting, dragging and linking graph nodes represented by visual icons. Each such node corresponds to a computational module in the back-end (tier 3), instantiated and managed by the WebFlow middleware (tier 2). WebFlow middleware is implemented as a group of management servlets, hosted by individual Java Web Servers and including Session Manager, Module Manager and Connection Manager.

WebFlow modules are represented as simple Java interfaces that implement methods such as *initialize*, *run* and *destroy*. Each module has some specified number of input and output ports. Data flows between connected modules from input to output ports. Each new data input activates internal computation of a module and results in generating some new data on the module output ports. This way a computational graph, once setup by the user via the visual authoring tools, can realize and sustain an arbitrary coarse grain distributed computation.

Dataflow model for coarse grain distributed computing has been successfully tested by the current generation systems such as AVS or Khoros, specialized for scientific visualization tasks and offering rich libraries of image processing filters and other visualization modules.

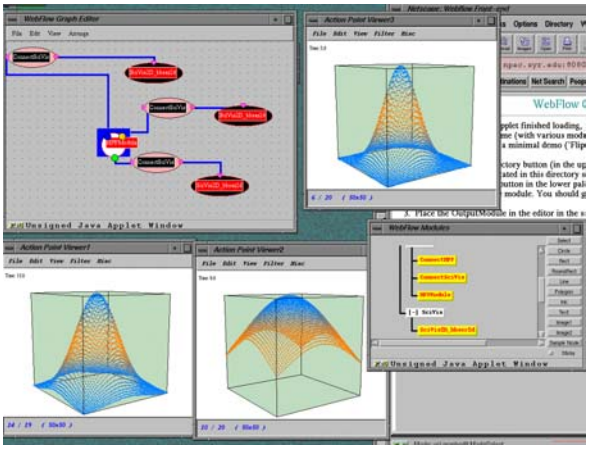

Figure 2.6: *Sample screendump from the WebFlow demo presented at Supercomputing '97: a set of real-time visualization modules is attached to the HPC simulation module (Binary Black Holes) using WebFlow visual dataflow editor (upper left frame).*

The distinctive feature of WebFlow is that it is constructed as a mesh of Web servers and hence it can be viewed as a natural computational extension of the Web information model. In the Web information model, individual developers publish information pages (located on different Web servers) and the user navigates visually such distributed information space by clicking hyperlinks. In the WebFlow computation model, individual developers publish computational modules (located on different Web servers), and the user connects them visually in terms of a computational graph to create a distributed dataflow computation.

3. Hybrid High Performance Systems

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 Figure 2.2 will become confused. In particular the physical realization does not necessarily reflect the logical architecture shown in Figures 2.1 and 2.3.

3.1 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 Figure 3.1, 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).

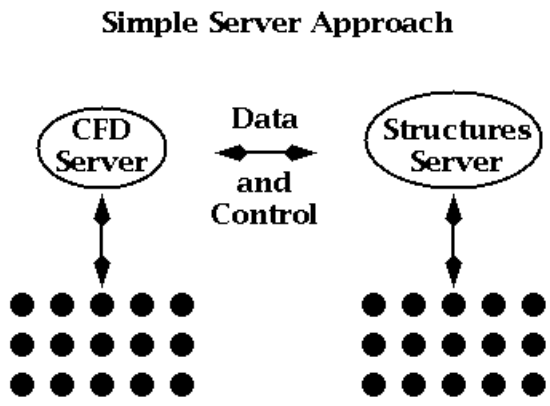

Figure 3.1: *Simple sequential server approach to LinkingTwo Modules*

Here we view MPI as the "machine language" of the higher-level commodity communication model given by approaches such as WebFlow from NPAC.

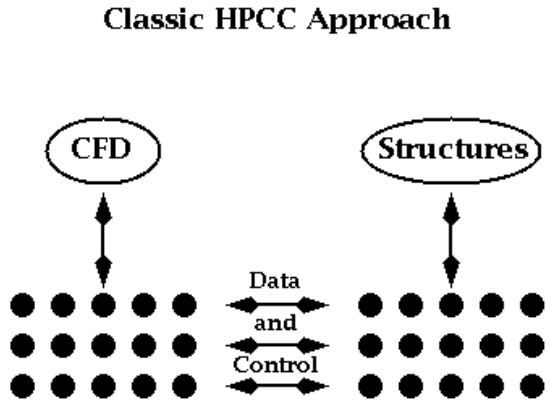

Figure 3.2: *Full HPCC approach to Linking Two Modules*

There is the "pure" HPCC approach of Figure 3.2, which replaces all commodity web communication with HPCC technology. However there is a middle ground between the implementations of Figures 3.1 and 3.2 where one keeps control (initialization etc.) at the server level and "only" invokes the high performance back end for the actual data transmission.

This is shown in Figure 3.3 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.

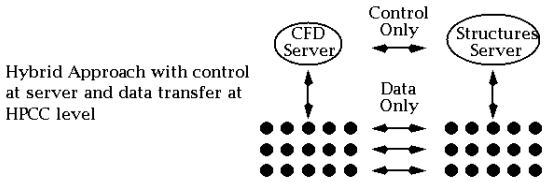

Figure 3.3: *Hybrid approach to Linking Two Modules*

3.1 Publish / Subscribe Model for HPcc

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 Figure 3.4, JavaBeans use the JDK 1.1 AWT event model with listener objects and a registration/call-back mechanism.

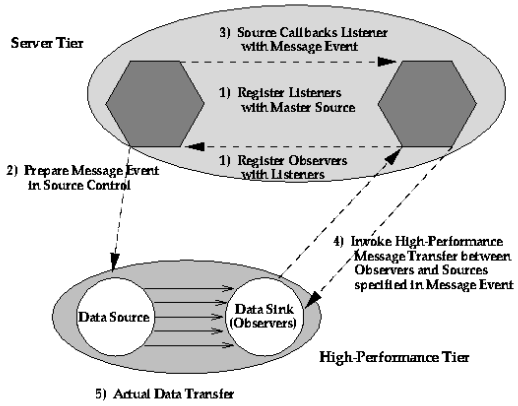

Figure 3.4: *JDK 1.1 Event Model used by (inter alia) Javabeans*

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.

3.2 Example: WebFlow over Globus for Nanmomaterials Monte Carlo Simulation

We illustrate here our concepts of hybrid communication approach discussed above on example of the WebFlow system at NPAC. A more detailed presentation of WebFlow can be found in Section 5.7. In a nutshell, WebFlow can be viewed as an evolving prototype and testbed of our High Performance Commodity Computing and Pragmatic Object Web concepts and it is therefore referenced from various perspectives in several places in this document. Here we summarize the architecture and we expose the middleware and/or backend communication aspects of the system.

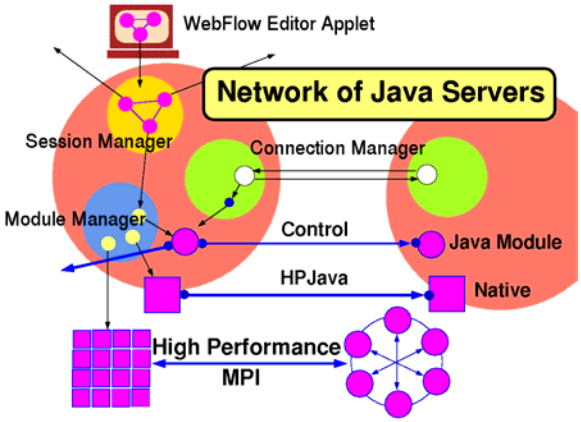

Figure 3.5: *Overall Architecture of the 3-tier WebFlow model with the visual editor applet in tier-1, a mesh of Java Web Serversin tier 2 (including WebFlow Session Manager, Module Manager and Connection Manager servlets), and (high performance) computational modules in tier-3.*

WebFlow is a distributed, Web based visual dataflow authoring environment, based on a mesh of middleware Java servers that manage distributed computational graphs of interconnected back-end modules, specified interactively by users in terms of the front-end graph editing applets. Figure 3.5 illustrates two natural communication layers in WebFlow: high functionality low performance pure Java middleware control and high performance MPI based data transfer in the backend. The coming HPJava layer (discussed in Section 7.6) can be viewed as interpolating between these two modes.

In the early WebFlow prototype, demonstrated at SC'97, we used Java sockets to connect between module wrappers in the middleware and the actual HPC codes in the backend. The new version of WebFlow under development based on JWORB middleware servers (see Section 5) will offer more powerful CORBA wrapper techniques for binding middleware control written in Java with multi-language backend codes (typically written in C, C++ or FORTRAN). CORBA wrapping technology offered by WebFlow, enables visual Web based interfaces for the current generation pure HPCC and Metacomputing systems such as Globus or Legion. We are currently experimenting with WebFlow-over-Globus interfaces (see Figure 3.6) in the context of some selected large scale applications described below.

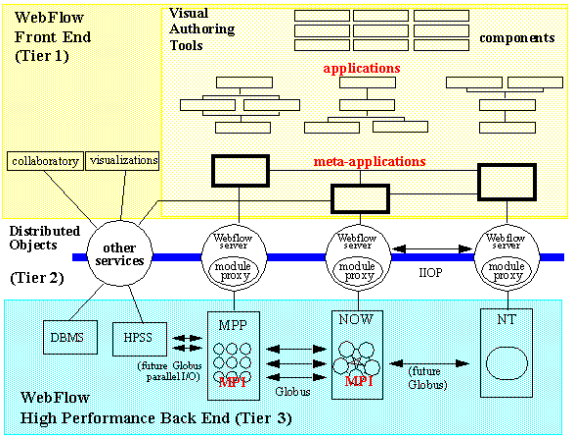

Figure 3.6: *Top level view of the WebFlow environment with JWORB middleware over Globus metacomputing or NT cluster backend*

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 simulation's [19]. This application is illustrated in Figures 3.7 and 3.8 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.

The high performance part of the backend tier in implemented using the GLOBUS toolkit [20]. 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.

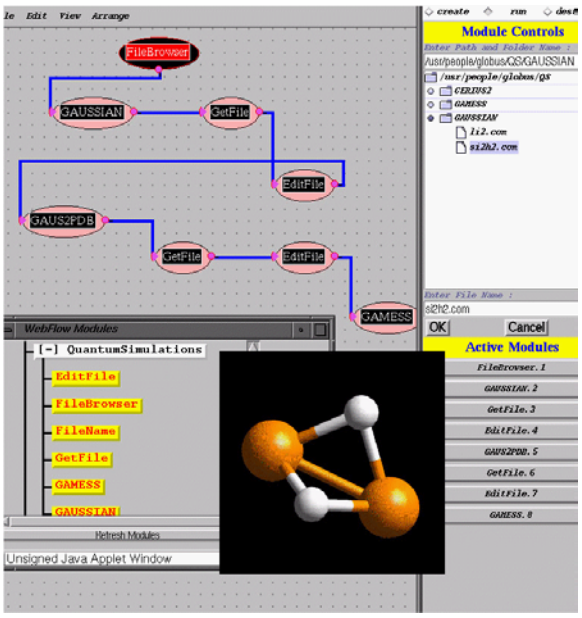

Figure 3.7: *Screendump of an example WebFlow session: running Quantum Simulations on a virtual metacomputer. Module GAUSSIANis executed on Convex Exemplar at NCSA, module GAMESS is executed on SGI Origin2000, data format conversion module is executed on Sun SuperSparc workstation at NPAC, Syracuse, and file manipulation modules (FileBrowser, EditFile, GetFile) are run on the researcher's desktop.*

The high performance part of the backend 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 Figure 3.8.

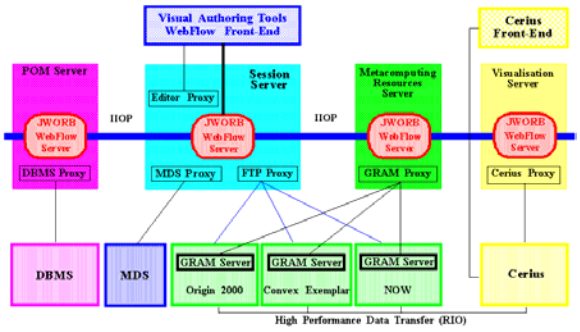

Figure 3.8: *WebFlow-over-Globus implementation of the Quantum Simulations*

4. Pragmatic Object Web – Stakeholders

We discuss now in more detail the four major players in the area of Object Web computing: Java, CORBA, COM and WOM. Each of these models offers an attractive and poweful enough framework, capable of addressing most of the relevant challenges in modern distributed computing programming. In consequence, each of these models claims completeness and tries to dominate and / or monopolize the market. Most notably, Java appeared during the last few years as the leading language candidate for distributed systems engineering due to its elegant integrated support for networking, multithreading and portable graphical user interfaces.

While the "Java Platform" or "100% Pure Java" philosophy is being advocated by Sun Microsystems, industry consortium led by the OMG pursues a multi-language approach built around the CORBA model. It has been recently observed that Java and CORBA technologies form a perfect match as two complementary enabling technologies for distributed system engineering. In such a hybrid approach, referred to as Object Web [2], CORBA is offering the base language-independent model for distributed objects and Java offers a language-specific implementation engine for the CORBA brokers and servers.

Meanwhile, other total solution candidates for distributed objects/components are emerging such as DCOM by Microsoft or WOM (Web Object Model) by the World-Wide Web Consortium. However, standards in this area and interoperability patterns between various approaches are still in the early formation stage. For example, recent OMG/DARPA workshop on compositional software architectures [33] illustrated very well both the growing momentum and the multitude of options and the uncertainty of the overall direction in the field. A closer inspection of the distributed object/component standard candidates indicates that, while each of the approaches claims to offer the complete solution, each of them in fact excels only in specific selected aspects of the required master framework. Indeed, it seems that WOM is the easiest, DCOM the fastest, pure Java the most elegant and CORBA the most realistic complete solution.

In our Pragmatic Object Web [3] approach at NPAC we adopt an integrative methodology i.e. we setup a multiple-standards based framework in which the best assets of various approaches accumulate and cooperate rather than competing. We start the design from the middleware which, offers a core or a `bus' of modern 3-tier systems and we adopt Java as the most efficient implementation language for the complex control required by the multi-server middleware. We adopt CORBA as the base distributed object model at the Intranet level, and the (evolving) Web as the world-wide distributed (object) model. System scalability requires fuzzy, transparent boundaries between Intranet and Internet domain's which therefore translates into the request of integrating the CORBA and Web technologies. We implement it by building a Java server (JWORB [13]) which handles multiple network protocols and includes support both for HTTP and IIOP. On top of such Pragmatic Object Web software bus, we implement specific computational and collaboration services.

We discuss our POW concepts and prototypes in more detail in the next section. Here, we summarize first the four major technologies that enable the POW infrastructure: Java, CORBA, COM and WOM.

4.1 Java

Java is a new attractive programming language based on architecture-neutral byte code interpretation paradigm that took the Web / Commodity community by storm in 1995 when the language and the associated software: Java Virtual Machine and Java Development Kit were published for the Internet community. The Java programming language though closely aligned to the C++ syntax, avoids C++

language features that lead to programming errors, obfuscated code, or procedural programming. Java can be looked upon as an isotope of C++ minus the pointers and accompanying pointer arithmetic, operator overloading, *struct* and *union*. The automatic memory management, instead of requiring applications to manage its heap-allocated memory thus minimizing memory leaks and erroneous memory references, and Operating System abstractions contributes to making it an intriguingly productive environment. However Java's platform independence and the write-once-run-anywhere promise is not without its baggage of drawbacks, thanks to its assumption of the lowest possible denominator of operating system resources.

Nevertheless, building on top of the rapid success of Java as a programming language, and exploiting the natural platform-independence of Java byte codes, Sun is developing Java as a complete computing platform itself. They do this by offering an already extensive and continuously growing suite of object libraries, packages as frameworks for broad computational domains such as Media, Security, Management, Enterprise, Database, Commerce, Componentware, Remote Objects, or Distributed Computing services.

Hence, from the Sun Microsystems perspective, Java is more than a language - its a framework that comprises many components. It includes PicoJava, a hardware implementation of the Virtual Machine; the JavaOS, an operating system implementation, and application programming interfaces to facilitate development of a broad range of applications, ranging from databases (JDBC) and distributed computing (RMI) to online commerce (JavaCard) and consumer electronics (Personal Java).

In our approach, Java plays a dual role. Like the rest of the computing community, we embrace Java as a programming language, ideal for middleware programing in our POW framework. In particular, our JWORB server that forms the core of the POW middleware is written in 100% pure Java. As such, however, it is heavily used to facilitate interfaces to codes in languages other than Java such as C/C++ back-ends, XML middleware or VBA, HTML or JavaScript front-ends.

Another role played by Java in our POW framework is as one of the four major competing Object Web technologies. Java model for distributed computing includes JavaBeans based componentware, RMI based remote objects and JINI based distributed services. In the following, we review these components of Java, promoted by Sun and opposed by Microsoft and viewed in Redmond as competing with and inferior to ActiveX, DCOM and Millenium, respectively (discussed in turn in Section 4.3).

4.1.1 Java Beans

JavaBeans formalizes the component reuse process by providing mechanisms to define components in Java and specify interactions amongst them. Java Beans components provide support wherein component assemblers discover properties about components. One of the other interesting aspects of JavaBeans is that they also accommodate other component architectures such as OpenDoc, ActiveX and LiveConnect. So by writing to JavaBeans the developer is assured that the components can be used in these and other component architectures.

GUI Beans Since it's a "component architecture" for Java, Beans can be used in graphical programming environments, such as Borland's JBuilder, or IBM's VisualAge for Java. This means that someone can use a graphical tool to connect a lot of beans together and make an application, without actually writing any Java code -- in fact, without doing any programming at all. Graphical development environments let you configure components by specifying aspects of their visual appearance (like the color or label of a button) in addition to the interactions between components (what happens when you click on a button or select a menu item).

Enterprise Beans One important aspect of Java Beans is that components don't have to be visible. This sounds like a minor distinction, but it's very important: the invisible parts of an application are the parts that do the work. So, for example, in addition to manipulating graphical widgets, like checkboxes and menus, Beans allows you to develop and manipulate components that do database access, perform computations, and so on. You can build entire applications by connecting pre-built components, without writing any code. Such middleware Beans come with their own management model, specified recently by the Enterprise JavaBeans (EJB) interfaces, now being implemented and tested by various vendors in several domains of enterprise computing.

Introspection A "Bean" is just a Java class with additional descriptive information. The descriptive information is similar to the concept of an OLE type library, though a bean is usually self-describing. Any

Java class with public methods can be considered to be a Bean, but a Bean typically has properties and events as well as methods.

Because of Java's late binding features, a Java.*class* file contains the class's symbol information and method signatures, and can be scanned by a development tool to gather information about the bean. This is commonly referred to as "introspection" and is usually done by applying heuristics to the names of public methods in a Java class. The Beans specification refers to these heuristics of introspection as "design patterns".

Properties The property metaphor in Java essentially standardizes what is common practice both in Java and other object-oriented languages. Properties are set of methods that follow special naming conventions. In the case of read/write properties, the convention is that if the property name is XYZ, then the class has the methods setXYZ and getXYZ respectively. The return type of the getter method must match the single argument to the setter method. Read-only or write-only properties have only one of these methods. In addition to single and multi-value properties, JavaBeans defines bound and constrained property types. Bound properties use Java events to notify other components of a property value change; constrained properties let these components veto a change. Constrained properties provide a uniform language-based approach to basic validation of business rules.

For those who are queasy about the idea of enforced naming conventions, JavaBeans provides an alternate approach. Explicit information about a class can be provided using the BeanInfo class. The programmer sets individual properties, events, methods using a Bean Info class and several descriptor class types (viz. Property Descriptor, for specifying properties or the Method Descriptor for specifying methods). To some extent, naming conventions do come into play here as well, as when defining the a BeanInfo class. When an RAD Tool wants to find out about a JavaBean, it asks with the Introspector class by name, and if the matching BeanInfo is found the tool uses the names of the properties, events and methods defined inside that pre-packages class. If not the default is to use the *reflection* process to investigate what methods exist inside a particular JavaBean class.

Java Event Model A software event is a piece of data sent from an object, notifying the recipient object of a possibly interesting occurrence. This occurrence could be a mouse move used in windowing systems, or it could also be the notification for a datagram packet arriving from a network. Basically any occurrence can be modeled as an event and the relevant information regarding the event can be encapsulated within the event. To put it simply an event is self describing, viz. the mouse click event would include the time the click occurred. It might also include such information as where on the screen the mouse was clicked, the state of the SHIFT and ALT buttons, and an indication of which mouse button was clicked. The object sending the event is said to be *firing* the event, while the object receiving the event is said to be the *recipient*. Software systems are usually modeled in terms of the event-flow in the system. This allows for clear separation of the components firing events and components responsive to those events. For example, if the right-mouse button is pressed within a Frame, then a popup menu is being thrown; or if a datagram packet just arrived, some relevant is being processed. Firing and responsive handling of events are one of two ways that objects communicate with each other, besides invoking methods on each other.

Java 1.1 Event Model In Java1.0.2 and earlier versions, events were passed to all components that could possibly have an interest in them. Events traversed upward the whole component/container hierarchy until they either found an interested component, or they reached the top frame without anyone's interest and they were then discarded. Java1.1 introduced a new event API, based on the delegation model, and now extensively used by Beans. In this model, events are distributed only to objects that have registered an interest in event reception. Event delegation improves performance and allows for clearer separation of event-handling code.

The delegation model implements the Observer-Observable design pattern with events. The flexible nature of the current event model allows classes that produce events to interact with other classes that don't. Instead of defining event-processing methods that client subclasses must override, the new model defines interfaces that any class may implement if it wants to receive a particular message type. This is better because an interface defines a "role" that any class may choose to play by implementing the set of operations that the interface defines. Instead of searching for components that are interested in an event - the handleEvent() mechanism, the new model requires objects to be registered to receive events, only then are the objects notified about the occurrence of that particular event. To receive an

event, the objects are registered with event source via a call to the *addListener* method of the source. Event listeners provide a general way for objects to communicate without being related by inheritance. As such, they're an excellent communication mechanism for a component technology -- namely, JavaBeans which allows for interaction between Java and other platforms like OpenDoc or ActiveX.

Most of the listener interfaces have a corresponding abstract adapter class providing a null implementation to all the methods within the interface. Rather than implementing the listener interface directly, one has the option of extending an adapter class (since it is an abstract class) and over-riding only methods relevant to the design. The adapters are available for those listener interfaces that have more than one method in the listener interface.

4.1.2 Java RMI

Java Remote Method Invocation (RMI) is a set of APIs designed to support remote method invocations on objects across Java virtual machines. RMI directly integrates a distributed object model into the Java language such that it allows developers to build distributed applications in Java. More technically speaking, with RMI, a Java program can make calls on a remote object once it obtains a reference to the remote object. This can be done either by looking up the remote object in the bootstrap naming service provided by RMI or by receiving the reference as an argument or a return value.

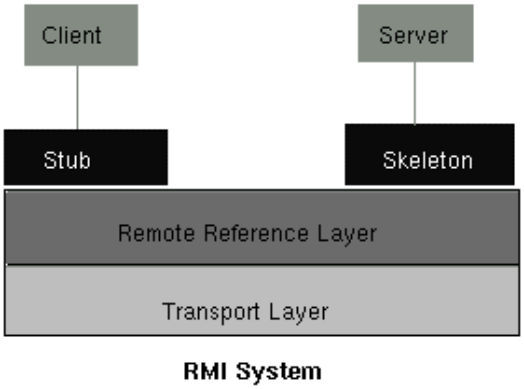

Figure 4.1: *The RMI Sub-system*

One of the key design requirements in the Distributed Object environment is to create nuggets of behavior which can be shipped from place to place. Oddly enough, classes provide a nice encapsulation boundary for defining what one of these nuggets are. Java with its mobile code facility (the ability to transfer executable binaries to wherever they need to be executed) handles inter-object communications within its native environment. The Java RMI and Java Serialization interfaces allow Java-Objects to migrate around the network, heedless of what hardware platform they land on, and then control each other remotely. Currently, Java RMI uses a combination of Java Object Serialization and the Java Remote Method Protocol (JRMP) to convert normal-looking method calls into remote method calls. Java RMI supports its own transport protocol today (JRMP) and plans to support other industry standard protocols in the near future, including IIOP. In order to support IIOP as it is today, JavaSoft will define a restricted subset of features of Java RMI that will work with IIOP. Developers writing their applications to this restricted subset will be able to use IIOP as their transport protocol.

The Java Distributed Object model differs from its intra-virtual-machine counterpart in the following important ways. Clients of remote objects interact with remote interfaces, never with the implementation classes of those interfaces. Thus any remote object passed as an argument or return value (either directly or embedded within a local object) must be declared as the remote interface, not the implementation class. Since references to objects are only useful within a single virtual machine, non-remote arguments to, and results from, a remote method invocation are passed by copy rather than by reference. Also a remote object is always passed by reference, and never by copying the actual remote implementation. Besides these, given a remote interface each method must declare java.rmi.RemoteException in its throws clause, in addition to any application-specific exceptions, to account for failure modes during method invocations.

4.1.3 JINI

Beans and RMI can be viewed as infrastructure elements, enabling various specific 100% pure Java based distributed computing environments. JINI is new Sun's effort towards a more complete distributed system, aimed at the emerging market of networked consumer electronic devices. A Jini system is based on the idea of federating groups of users and the resources required by those users. The overall goal is to turn the network into a flexible, easily administered tool on which human or computational clients can find resources efficiently. Examples include a JINI enabled printer that installs and configures itself automatically when plugged into the

office LAN; or a JINI-capable appliance that can be activated on the home JINI network in a true plug-and-play mode.

JINI Service is an entity that can be used by a person, a program, or another service. Service Protocol defines the interface of the service. Services are found and resolved by a *lookup service*.

Lookup service is a major component in the system, used to establish connection between users and services. Objects in lookup service may include other lookup services so that hierarchical lookup can be provided with no cost. Some services can have *user interface*(GUI) so that the service can be controlled by the user in real time. Objects which are related to a particular service builds the *object group* and they live in a single address space together. Each object group lives in different virtual machine so that a certain location and security-based requirements can be supplied. Lookup process includes the following steps described below.

Lookup occurs when a client or user needs to locate and invoke service described by its type (its java interface) and possibly, other attributes. If user can not find a lookup service, then it can send out the same identification message, which the lookup service uses, to request service providers to register. Then, user can pick the service from the incoming registration requests. User downloads the proxy object for the service from lookup service and uses this object to access to the required service.

User interacts with the service through this proxy object which provides a well-defined interface for this service. A proxy object may use its proprietary communication protocol with the actual server. If a service has an user interface, then user gets a proxy object in *Applet* type and this object can be displayed by browsers or other user-interface tools.

When services join or leave a lookup service, events are signaled, and objects that have registered interest in such events get notifications.

Discovery process indicates that the service is discovered by Jini system. In other words, service is registered to the lookup service. Discovery occurs when a new service joins (a device is plugged in) to the Jini system and it includes the steps listed below.

The service provider locates a lookup service by broadcasting a presence announcement.

A proxy containing the interface and other descriptive attributes for the service is loaded into the lookup service.

RMI provides the communication protocol between services.

Security is defined with two concepts: principal and access control list (ACL). Services in the system are accessed on behalf of some entity (principal) which generally leads to a particular user in the system. ACL controls the access rights for the particular object (service).

Lease denotes that user has an access permission to the service over a time period. Lease expires at the end of the given time and user of the service loses its privilege to use the service unless user gets another lease before the expiration of the pervious one. Lease concept brings the time factor as a new ingradient to the validity of the object reference.

Leases can be exclusive or non-exlusive. Meaning that user can be the only user of a service or it can share the service with others.

Transaction interface provides a service protocol needed to coordinate a two-phase commit. The correct implementation of the desired transaction semantics is up to the implementers of the particular objects that are involved in the transaction.

Distributed Events This technology allows an object in one Java virtual machine to register interest in the occurence of some event occurring in an object in some other Java virtual machine.

Distributed Event API allows to introduce third party objects between event generators and consumers so that it is possible to off-load the objects from excessive notifications, to implement various delivery guarantees, storing of notifications until needed or desired by a recipient, and the filtering and rerouting of notifications.

Remote Event has the following fields: a) event kind; b) a reference to the object in which this event is occurred; c) a sequence number of the event; d) an object supplied by the remote event listener at the registration phase.

An object, which wants to receive a notification of an event from some other object, should support *RemoteEventListener* interface. An object registers itself to the generator object, which supports

EventGenerator interface. In the registration, listener provides the event kind, a reference to listener object, how long it wants to stay in the listener mode(lease concept), and a marshalled object which will be handed back to the listener at notification. Registration method (register()) returns an EventRegistration object which contains the kind of event an object is registered for listening, the current sequence number of those events and a Lease object.

An object, which wants to receive a notification of an event from some other object, should support *RemoteEventListener* interface and registers itself to the generator.

Whenever event arrives, notify() method of the object will be invoked. This method is invoked with an instance of RemoteEvent object. RemoteEvent object contains a reference to the object in which this event is occurred, the kind of event, a sequence number of the event(event id), and a marshalled object provided during registration process.

FEATURE	JavaOM (RMI+JINI)	DCOM	CORBA
Multiple Inheritance	Java objects can implement multiple interfaces.	An object can support multiple interfaces.	IDL definition allows to define an interface which can inherit from multiple interfaces.
Registering Object	Lookup Service	Registry. Defines an association between the CLSID and the path name of the server executable. Since interface proxy/stub is a COM object, its DLL should be registered, too.	Implementation Repository. The interface name and the path name of server executable should be registered.
Wire Protocol	Java Remote Method Protocol (JRMP).	Object Remote Procedure Call(ORPC) – Microsoft extension of DCE RPC	General Inter-ORB Protocol(GIOP) A specific mapping of GIOP on TCP/IP is known as Internet Inter-ORB Protocol(IIOP)
Data Marshaling/Unmarshaling Format	Object Serialization	Network Data Representation(NDR)	Common Data Representation(CDR)
Dynamic Invocation	Java's reflection support allows user to obtain the necessary operation information about an object.	Type Library and IDispatch interface or Type Library-driven Marshaling.	Interface Repository. Dynamic Invocation Interface(DII) and Dynamic Skeleton Interface(DSI).
Exceptions	Java provides necessary exceptions as an object. User can define additional exceptions.	A 32-bit error code called an HRESULT should be returned from each method.	Predefined exceptions and allows user to define his/her own exceptions in IDL.

Figure 4.2: *Comparison of some major design, implementational and programmatic features in the distributed object / component models of JINI, COM and CORBA.*

JavaSpaces technology acts as a lightweight infrastructure de-coupling the providers and the requestors of network services, by delivering a unified environment for sharing, communicating, and coordinating. Developed before JINI and initially driven by the Linda concepts, JavaSpaces are now being repackaged as an example of JINI Networking Service.

Using Javaspaces, one can write systems that use flow of data to implement distributed alogorithms while implementing distributed persistence implicitly. This is different from the approach towards Distributed computing which involves creation of remote method invocation-style protocol. The JavaSpaces' "flow of objects" paradigm is based on the movements of objects into and out of JavaSpaces implementations. JavaSpaces uses the following programming language API's to achieve this end - Java RMI, Object Serialization, Distributed events and Transactions.
A Space holds entries, which are a typed group of objects, expressed by a class that implements the interface *space.entry*. There are four primary kinds of operations that you can invoke on a space. Each operation has parameters that are *entries* and *templates*, a special kind of entry that have some or all of its fields set to specified *values* that must be matched exactly. The operations are: a) *write* a given entry into the space; b) *read* an entry that matches the given template from the space; c) *take* an entry that matches the given template from this space, thereby removing it from the space; and d) *notify* a specified object when entries that match the given template are written into the space.

All operations that modifythe JavaSpaces server are performed in a transactionally secure manner with respect to that space. The JavaSpaces architecture supports a simple transaction mechansism that allows multi-operation and/or multi-space updates to complete atomically using the two-phase commit model under default transaction semantics.

All operations are invoked on a local smart proxy for the space. The actual implementation could reside on either the same machine or a remote machine. The server for the application that uses JavaSpaces technology will be completely implemented once the entries are designed. This obviates the need to for implementing a remote object for a specified remote interface or using *rmic* to generate client stubs and implementation skeletons.
In contrast, a customized system needs to implement the server. The designer of this server need to deal with concurrency issues and atomicity of operations. Also, someone must design and implement a reliable storage strategy that guarantees the entries written to the server are not lost in an unrecoverable or undetectable way. If multiple bids need to be made atomically, a distributed transaction system has to be implemented. JavaSpaces technology solves these problems, the developer does not have to worry about them. The spaces handle concurrent access. They store and retrieve entries atomically and provide implementation of the distributed transaction mechanism.

4.2 CORBA by OMG

CORBA (Common Object Request Broker Architecture) defines a set of specifications formalizing the ways software objects cooperate in a distributed environment across boundaries such as networks, programming languages, and operating systems. Supported by the Object Management Group (OMG), the largest software industry consortium of some 700+ companies, CORBA is the most ambitious ongoing effort in the area of distributed object computing. CORBA supports the design and bottom-up development of new enterprise systems and could also be utilized for the integration of legacy systems and sub-systems.

4.2.1 Object Request Broker

An implementation of the CORBA standard defines a language and platform-independent object bus called an ORB (Object Request Broker), which can translate between different data formats (big-endian or little-endian) besides other attributes. The ORB, thus lets objects transparently inter-operate, and discover each other, across address spaces, networks, operating systems and languages. Distributed objects cooperate by sending messages over a communication networks. Each implementation of the CORBA standard, the object bus, is able to communicate with any other implementation of the standard., the protocol used to achieve this end is the Internet Inter-ORB Protocol (IIOP).

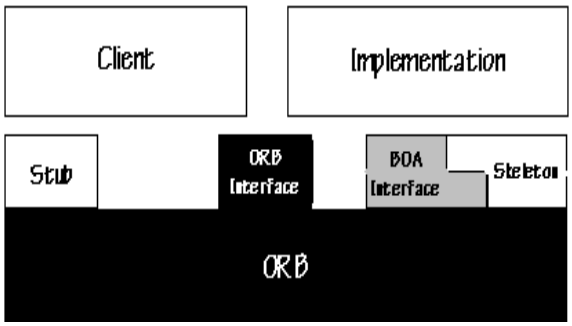

Figure 4.2: *The Object Request Broker*

A client application needn't know the location details of the object it needs to use. The only information

that is required on the client side is the object's name and details on how to use the remote objects interface. Details pertaining to object location, request routing, invocation and returning of the result are transparently handled by the ORB. Using IIOP its possible to use the Internet itself as a backbone ORB through which other ORB's can merge. CORBA is widely implemented in many languages (though they vary in degree of CORBA-compliance, portability and avaiability of additional features), besides supporting the mixing of languages within a single distributed application.

4.2.2 IIOP - Internet Inter-ORB Protocol

The General Inter-ORB Protocol (GIOP) defines a set of message formats and data formatting rules, for communication between ORB's. GIOP's primary goal is to facilitate ORB-to-ORB communications, besides operating directly over any connection oriented protocol. The Common Data Representation (CDR), which is tailored to the data types supported in the CORBA Interface Definition Language (IDL), handles inter-platform issues such as byte ordering. Using the CDR data formatting rules, the GIOP specification also defines a set of message formats that support all of the ORB request/reply semantics defined in the CORBA core specification. GIOP also defines a format for Interoperable Object References (IOR). ORB's create IOR's whenever an object reference needs to be passed across ORBs. IORs associate a collection of tagged profiles with object references, which provide information on contacting the object using the particular ORB's mechanism.

GIOP messages can be sent over virtually any data transport protocol, such as TCP/IP, Novell SPX, SNA protocols, etc. To ensure "*out-of-the-box*" interoperability between ORB products, the IIOP specification requires that ORBs send GIOP messages over TCP/IP connections. TCP/IP is the standard connection-oriented transport protocol for the Internet. To be CORBA 2.0 compatible, an ORB must support GIOP over TCP/IP, hence IIOP now has become synonymous with CORBA. Using IIOP any CORBA client can speak to any other CORBA Object. The architecture states that CORBA objects are location transparent. The implementation, therefore, may be in the same process as the client, in a different process or on a totally different machine. Also it should be noted that has built in mechanisms for implicitly transmitting context data associated with transactions and security services.

4.2.3 The Object Management Architecture Model

The Object Management Architecture (OMA) is composed of an *Object Model* and a *Reference Model*. Object Model defines how objects are distributed across a heterogeneous environment. The Reference Model defines interactions between those objects. The OMA essentially, is the high level design of a distributed system provided by the OMG.

OMA Object Model. The OMG Object Model defines common object semantics for specifying the externally visible characteristics of objects in a standard and implementation-independent way. In this model clients request services from objects (which will also be called servers) through a well-defined interface. This interface is specified in OMG IDL (Interface Definition Language). A client accesses an object by issuing a request to the object. The request is an event, and it carries information including an operation, the object reference of the service provider, and actual parameters (if any).

OMA Reference Model The OMA Reference Model consists of the following components: a) Object Request Broker; b) Object Services; c) Common Facilities; d) Domain Interfaces; e) Application Interfaces. We discuss these components in more detail in the following Sections.

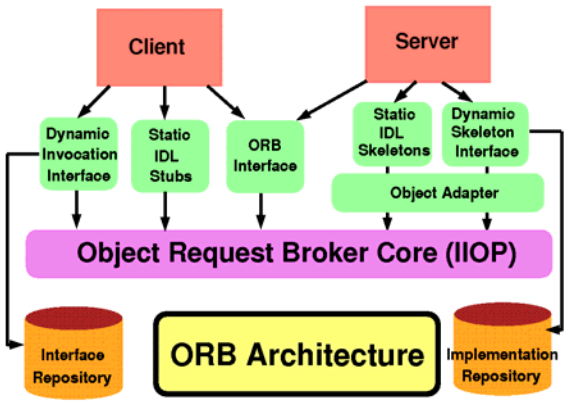

Figure 4.3: *CORBA Architecture, including client and server communication terminals, IIOP transport and persitent repository components, all participating in the remote method invocation process.*

Object Request Broker : CORBA (Common Object Request Broker Architecture) specification defines Interface Description Language (IDL) language and platform-independent object bus called ORB (Object request Broker), which lets objects transparently make requests to, and receive responses from, other

objects located locally or remotely. It takes care of locating and activating servers, marshaling requests and responses, handling concurrency, and handling exception conditions. Hence, ORB technology offers a useful approach for deploying open, distributed, heterogeneous computing solutions. IIOP (Internet Inter ORB Protocol) is an ORB transport protocol, defined as part of the CORBA 2.0 specification, which enables network objects from multiple CORBA-compliant ORBs to inter-operate transparently over TCP/IP.

Object Services : Collection of services, that support basic functions for using and implementing objects. These services standardize the life-cycle management of objects by providing interfaces to create objects, to control access to objects, to keep track of relocated objects, and to control the relationship between styles of objects (class management).These services are independent of application domains and do not enforce implementation details on the application. Current services are listed below.

Concurrency Control Service protects the integrity of an object's data when multiple requests to the object are processed concurrently.

Event Notification Service notifies interested parties when program-defined events occur.

Externalization Service supports the conversion of object state to a form that can be transmitted between systems.

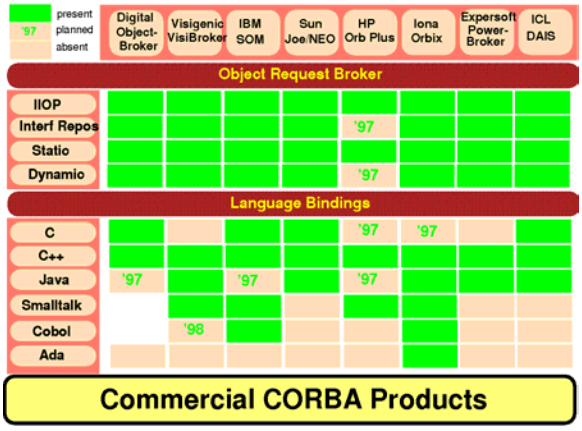

Figure 4.4: *Status of multi-language support in CORBA by major ORB vendors.*

Licensing Service control and manage remuneration of suppliers for services rendered.

Naming Service provides name binding.

Object Lifecycle Service supports creation, copying, moving, and destruction of objects.

Persistent Object Service supports the persistence of an object's state when the object is not active in memory and between application executions.

Property Service support the association of arbitrary named values (the dynamic equivalent of attributes) with an object.

Query Service supports operations on sets and collections of objects that have a predicate-based, declarative specification and may result in sets or collections of objects.

Relationship Service supports creation, deletion, navigation, and management of relationships between objects.

Security Service supports integrity, authentication, authorization and privacy to degrees, and using mechanisms, that are yet to be determined.

Time Service provides synchronized clocks to all objects, regardless of their locations.

Transaction Service ensures that a computation consisting of one or more operations on one or more objects satisfies the requirements of atomicity, isolation and durability.

Trader Service provides a matchmaking service between clients seeking services and objects offering services.

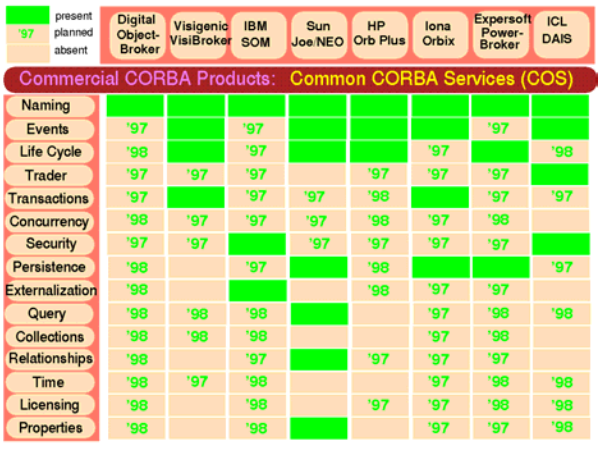

Figure 4.5: *Status of Common Services support in CORBA by major vendors.*

Common Facilities commercially known as CORBAfacilities provide a set of generic application functions that can be configured to the specific requirements of a particular configuration. These services however, aren't as fundamental as object Services. Examples include WorkFlow, Mobile Agents or Business Objects.

Domain Interfaces represent vertical areas that provide functionality of direct interest to end-users in particular application domains. These could combine some common facilities and object services. Examples of active vertical domains in the OMG include: Manufacturing, Telemedicine, Telecommunication, Enterprise Computing, Modeling and Simulation etc.

Application Interfaces are CORBA interfaces developed specifically for a given application. Because they are application-specific, and because the OMG does not develop applications, these interfaces are not standardized.

These components cater to the three main segments of the software industry viz. Application Oriented, System Oriented and Vertical Market Oriented. For handling the Application-Oriented systems, the OMA characterizes Interfaces and Common Facilities as solution-specific components that rest closest to the user. The ORB's and object Services help in defining System and Infrastructure aspects of distributed object computing and management. The Vertical Market segment is handled by Domain Interfaces, which are vertical applications or domain-specific interfaces.

4.2.4 Interface Definition Language

IDL is a specialized language for defining interfaces, and this is what facilitates the notion of interacting objects, so central to CORBA. The IDL is the means by which objects tell their potential clients what operations are available and how they should be invoked. The IDL definition defines types of objects, their attributes, the methods they export and the method parameters. From the IDL definitions, it is possible to map CORBA objects into particular programming languages or object systems.

A particular mapping of OMG IDL to a programming language should be the same for all ORB implementations. Language mapping includes definition of the language-specific data types and procedure interfaces to access objects through the ORB. It includes the structure of the client stub interface (not required for object-oriented languages), the dynamic invocation interface, the implementation skeleton, the object adapters, and the direct ORB interface. A language mapping also defines the interaction between object invocations and the threads of control in the client or implementation

4.2.5. CORBA 3.0

As a large open consortium, OMG proceeds slower, with the development and adoption of new standards, than the less constrained single vendor approaches of Java or COM. However the rapid onset of Web technologies has coincided with the invigoration and acceleration of OMG activities and we are witnessing a broad spectrum of new standard efforts. Some of these projects, to be released as part of the coming CORBA 3.0 specification are currently at various stages of RFP, proposals review, voting or standard adoptions, include

Multiple Services and Versioning: The former allows an object to support different interfaces, while the latter allows a composite object to support different versions of the same interface. This gives rise to a composite structure that is in a position to host interfaces which are not only in distinct in nature, but also have the capabilities of multiple versions instantiated by the client.

Sessions: Provide means for an object to maintain per-client context akin to a database/network server managing sessions of connection from the client. This it achieves by having multiple instances of a service within an object implementation.

COM support: Provide better support for the COM object model. COM allows a client application to query for one interface at a time. For CORBA to inter operate with the navigation code, of a client application, written for a specific COM infrastructure, the same code should be respected.

Messaging Service: OMG proposes messaging to be a service in its own right. The proposal requires the presence of interfaces that would,
a) Allow clients to make requests on an object without blocking the client execution thread b) Enable a client to make requests that may not complete during the lifetime of the client execution environment c) Allow object servers to control the order of processing incoming requests.

Object Pass-by-value : As of now CORBA supports passing of objects by reference only, thus if the receiver intends to access data or operations within the object it does so using the reference passed to it. Needless to say this adds to the wire traffic. The objects pass by value RFP addresses this issue. However there are some issues that need to be dealt with in the pass-by-value situation. For e.g. if an object is associated with different kinds of resources, and the same resources aren't available at the receiver's end then the consequent incompatibilities must be handled separately.

Component Model: Requirements specified by the OMG in the RFP for the CORBA component model are a) Definition of the concept of component type, and the structure of the component typing system b) Lifecycle of components and interfaces, and mechanisms for managing components c)Relationship between the component event mechanism and the existing event service d) specify a mapping of the proposed component model to the Java Beans component model.

Portable Object Adapter (POA): Takes into consideration all the design constraint the Basic Object Adapter (BOA) had, and makes it solvable. POA is far more scalable and portable than its predecessor. Features include a) Support for objects with persistent identities b) Support for the transparent object activation model. c) POA supports two types of threading model ORB controlled and single threaded behavior.

CORBA Scripting: The RFP requirements include specifying a scripting language that fits naturally into the CORBA object and (*proposed*) component models. Responses must also specify a)how scripts can invoke operations on CORBA objects b)relationship between the CORBA component model's event mechanism and scripting languages c)how the scripting language exposes and manages the proposed component model properties.

4.3 COM by Microsoft

COM (Component Object Model) allows an object to expose its functionality to other components and applications through its interface. It defines both how the object exposes itself and how this exposure work across processes and across networks. Interface based approach allows to ensure dynamic interoperability of binary objects written in different programming languages. Object lifecycle support in COM loads the server objects on demand if they are not loaded already and releases them whenever it finds out there is no user for this object.

COM supports Interface based programming framework where object exposes its functionality through interface. An object can support multiple interfaces. An interface is a pointer to a block of memory in which the first word is a pointer to a table of function addresses (virtual function table or v-table) as in C++ virtual function table. A client uses the interface only through its v-table since the object may actually be nothing more than a proxy which forwards the method calls to actual object.

All COM components provide *IUnknown* base interface which has the following methods in the first three entries of the v-table.: a) *QueryInterface* returns pointer of given interface (via IID or Interface Identifier) if the object supports this interface; b) *AddRef* increments the object's reference count; and c) *Release* decrements the object's reference count, freeing the object if the count becomes zero.

AddRef and *Release* methods are directly related to object lifecycle support of COM. Whenever client gets the interface pointer of the server object, the client counter for object is incremented so that system can track down the number of concurrent users of this particular object. Clients invoke *Release* method on interface pointers when they do not need it anymore. It has the same functionality as in destructor method of an object in C++. Invoking *Release* method on the interface decrements the client counter for this object in the COM internals and COM removes this object from runtime as soon this count drops zero (no client). When object supports multiple interfaces, client needs to get the interface pointer which, is requred*. QueryInterface* method provides the neccessary navigational functionality to obtain the required interface.

Any given function is identified with three elements:
a) The object's class identification number (CLSID); b) The interface type (Interface Identification Number-IID) through which the client will call member functions; and c) The offset of the particular member function in the interface.

CLSID and IID are unique 16-byte values and are generated by Globally Unique Identifier generators (such as GUIDGEN). These generators use the current time and the IP address of the machine to generate the number. This technique is defined in

OSF's specifications as Universally Unique Identifier (UUID).

An instance of an COM object is created with *CoCreateInstance* method by providing CLSID. COM keeps a registry for each CLSID and returns the necessary object's pointer. It starts the object if the object (library-EXE or DLL) is not loaded already. If the server object implemented in DLL format, then this object will be defined in the client processe's memory (*LoadLibrary*). If the server object implemented in EXE format, then this object will run in different process (*CreateProcess*).

If the server object is in another machine (its registery entry in the COM registry tells this), then COM starts a proxy server and this talks to Service Control Manager(SCM) of the other machine and starts the necessary object on the remote machine and returns the pointer of the object on the remote machine. Client's process receives the pointer of the proxy object running on the local machine.

COM supports *Location Transparency* so that an object can be in the client's address space, or in another container. The later case, proxy/stub pair will do the marshaling/unmarshaling so that the same server object can be shared by multiple clients. COM places its own remote procedure call(RPC) infrastructure code into the *vtable* and then packages each method call into a standard buffer representation, which it sends to the component's side, unpacks it, and reissues the original method call on a server object.

Versioning in COM is provided by defining additional interfaces for the new server object. Clients has to take care of controlling (QueryInterface()) whether server supports particular interfaces or not. Objects can support multiple interfaces.

Microsoft recomments developers to use *aggregation* for components which are quite substiantial and composed of a lot of interfaces. An object can contain another object responsible for the specific interface and when this interface is asked it will return the pointer of this object so that client continues to think that it is dealing with the old object. This has several advantages. For example, if the server object has a lot of services, it can be divided into sub-interfaces and each interface can be handled by different object. Wrapper object contains all of them, and when COM starts the object, only the wrapper starts and it takes less time and less memory. Whenever one interface is asked, the related object is being brought to memory not all of them. It is also possible to put different sub-components to different machines.

COM supports persistent object stores for the components which provides *IPersistStorage* interface which allows to store the object.

DCOM (Distributed Component Model) is a protocol that enables software components to communicate directly over a network in a reliable, secure, and efficient manner. Microsoft's Distributed Component Object Model is currently avaialble for two operating systems: Window95 and Windows NT4.0. DCOM ports to the UNIX environment are under way by the third party vendors. DCOM is language-independent, and its objects are described via interfaces using Microsoft's Object Description Language(ODL). DCOM retains full COM functionality and adds core distributed computing services summarized below.

Distributed Object Reference Counting (Pinging Protocool) is implemented by client machines sending periodic messages to their servers. DCOM considers a connection as broken if more then three ping periods pass without the component receiving a ping message. If the connection is broken, DCOM decrements the reference count and it releases the component if the count becomes zero.

Transport Protocol DCOM's preferred transport protocol is the connectionless UDP. DCOM takes advantage of connectionless protocol and merges many low level acknowledgement messages with actual data and pinging messages. Proxies that handle client requests can cache and then bundle several calls to the server, thereby optimizing the network traffic. The actual protocol used by DCOM is Microsoft's RPC (Remote Procedure Call) protocol, based on extended DCE RPC developed by the Open Software Foundation.

Distributed Security Services Each component can have an ACL (Access Control List). When a client calls a method or creates an instance of a component, DCOM obtains the client's current username associated with the current process. Windows NT authenticates this information. On the server object, DCOM validates this information by looking at component's ACL. Application developers can use the same security services to define finer security restrictions on the object such as interface based or method based security.

COM+ COM+ will provide a framework that will allow developers to build transaction-based distributed applications from COM components, managed from a single point of administration. COM+ will include: a) MTS transactions; b) Security administration and infrastructure services; c) Queued Request so that invocations on COM components can be done through Microsoft's message-queuing server (MSMQ); d) Publish-and-subscribe Event service; d) Load balancing service to allow transaction requests to be dynamically spread across multiple servers.

Unlike COM requires a low level programming style, COM+ offers higher level, more user-friendly API in which COM+ classes are handled in a similar way as the regular (local) C++ classes and the whole formalism shares several design features with the Java language model. A *coclass* keyword indicates that a particular C++ class should be exposed as a COM+ class. This informs C++ compiler to produce metadata about class. Aon the other hand, all (Microsoft) Java classes become automatically COM+ classes. The *coclass* specification defines the public view of an object and its private implementation.

Unlike COM, COM+ defines a data type set so that various languages can exchange data through method calls. Coclass variables include: a) *Fields* i.e. linternal variables that follow the low level memory layout and are accessed from within the class; b) *Properties* which are implemented as accessor functions which can be intercepted by the COM+ system services or user provided callbacks.

Method overloading follows the same rules as in Java i.e. it is allowed as long as signatures of various calls of the same method are different. COM+ allows also to define exceptions that might be generated by a method. Necessary conversions are taken care of by the COM+ execution environment whenever language boundaries are crossed.

COM+ instance creation and destruction is also similar as in C++, i.e. COM+ can recognize *constructor* and *destructor* special methods. It calls constructor whenever it creates a new instance of the object class and it calls destructor before it destroys the object instance.

COM+ interfaces are specified by the *cointerface* keyword. It is a recommended practice to use interface to define the public behavior and state of coclasses. Method and property definitions can be done in cointerface except that everything must be public and abstract. *cointerface* supports multiple inheritance.

COM+ supports several types of event mechanisms:

interface-based events In this model, sinks implement interfaces to catch events and sources fire events by calling methods on those interfaces. The problem with this approach is that developer needs to support all methods defined in the interface.

method-based events Here, instead of implementing the whole interface, only the particular method is implemented (and the others are automatically assigned with a default / null implementation via a suitable adapter).

persistent events Here the sinks and sources are fully decoupled and and only dynamically linked via the publish/subscribe techniques. Sources publish information about events that can happen and sinks can subscribe to them.

COM+ supports both *interface inheritance* and *implementation inheritance*. In a similar way as in Java, coclass indicates with *implements* keyword that it will implement the particular cointerface. coclass can inherit both interface and implementation from another coclass. COM+ does not permit multiple inheritance of coclasses.

Interception is a new key concept in COM+. Whenever a method or property call is being executed, compiler/interpreter transfer control to the COM+ object runtime service instead of performing direct method call on an object. The type of interception related to services can be defined with attribute of coclass, cointerface and their methods therefore

Closely related COM+ classes can be packaged in two ways: a) *logical* – using namespace as a collector of classes and possibly other namespaces, with each class belonging to only one namespace; b) *physical* or module based with COM+ module such DLL or EXE acting as a single deployable unit of code.

Millenium COM+ is yet to be released and we are already getting first insights in the next step after-COM+ technologies and systems being constructed by Microsoft under the collective name *Millenium*. Millenium is is an ambitious effort by Microsoft Research towards a new high level user-friendly

distributed operating system. So far, only the general set of requirements or goals and the associated broad design principles were published but one can naturally expect Millenium systems to layer on top of COM+ infrastructure in a similar way as JINI is building its services on top of Java. Further, given obvious affinity between COM+ and Java and the generally similar goals of the currently competing global software systems, we could expect Millenium and JINI to share in fact many common design features.

Some general features and goals of Millenium include: a) seamless distribution; b) worldwide scalability; c) fault-tolerance; d) self-tuning; e) self-configuration; f) multi-level security; g) flexible resource controls. Main design principles, identified to accomplish these goals, include: a) aggressive abstraction; b) storage-irrelevance; c) location-irrelevance; d) just-in-time-binding; e) introspection.

Networking vision of Millenium and JINI is similar. Any new computing or consumer electronic device, when plugged into Millenium network, publishes its characteristics, looks up and retrieves the information about the available capabilities, and it configures itself automatically and optimally in a given environment. Hardware failures are detected and corrected automatically via build-in heartbeat and code mobility services. Transient network congestion problems e.g. in case of Internet traffic peaks are resolved in the global Millenium mode by predictive replication of the to-be-hot servers, followed by the cache cleanup and service shrinkage after the hit wave is over.

Some Millenium prototype subsystems under development include: a) *Borg* – a distributed Java VM with single system image across many computers; b) *Coign* – an Authomatically Distributed Partitioning System (ADPS) for COM+ based applications; and c) *Continuum* which offers Single System Image (SSI) support for multi-language distributed Windows applications.

4.4 XML based WOM by W3C

The evolution of distributed object technologies was dramatically accelerated in mid '90s by the Web phenomenon. Proprietary technologies such as OAK->Java or OLE->COM got published, and open standard activities such as CORBA got significantly accelerated after the rapid world-wide adoption of the Web paradigm. In turn, the World-Wide Consortium is bringing now the next suite of standards based on XML technology such as RDF, DOM or XSL which, when taken collectively, can be viewed as yet another, new dynamic framework for what is sometimes referred to as the Web Object Model (WOM). The current XML technology landscape is rapidly evolving, multi-faceted and complex to monitor and evaluate. At the time of this writing (Aug/Sept 98), the core WOM technologies such as XML, DOM, RDF or XSL are at the level of 1.0 releases. The WOM –like approaches are still in the formation stage and represented by models such as WIDL, XML RPC, WebBroker and Microsoft SOAP. In the following, we summarize briefly the stabilizing core technologies and review the emergent WOM candidates.

XML is a subset of SGML, which retains most of the SGML functionality, while removing some of the SGML complexity and adapting the model for the Web needs. Hence XML offers an ASCII framework for building structured documents based on custom markup defined by users in terms of dedicated tags. A set of tags forming a particular document type is naturally grouped in the DTD (Document Type Definition) files. RDF (Resource Definition Framework) is an XML based formalism for specifying metadata information about the Web resources. Several early XML applications, i.e. little languages for specialized domains based on XML meta-syntax were recently constructed, for example CML (Chemical Markup Language), MathML for mathematical notation, CDF (Channel Definition Format) to support push technology for Web pages, SMIL (Simple Multimedia Integration Language) or OSD (Open Software Description).

The power of the XML approach stems from the growing family of freely available tools. For example, each of the XML-compliant languages listed above and all other that are under construction can be parsed by a general purpose parser and there is already some 20+ free XML parsers available on the Web. These parsers generate intermediate representations in one of the several standard formats at various levels of abstraction, ranging from low level ESIS (Element Structure Information Set) which emits events corresponding to the XML tokens detected by the scanner, to callback based SAX (Simple API for XML) model, to object-oriented DOM (Document Object Model) to rule based XSL (XML Style Languague).

Current main focus of the XML standard development effort is on more ogranized and

formalized middleware support for handling complex documents, their databases, metadata, transformations, agent based autonomous filters etc. In parallel with these activites, several early attempts towards WOM are already emerging such as WebBroker or RIO server by DataChannel, XML RPC and B2B (Business-to-Business) Integration Server by WebMethods, or SOAP (Simple Object Access Protocol) by Microsoft. In the following, we review some of these models.

Microsoft SOAP stands for Simple Object Access Protocol and it enables Remote Procedure Calls (RPC) to be sent as XML wrapped scripts across the Web using HTTP protocol. As of summer/fall '98, SOAP is still a work in progress, pursued by Microsoft, UserLand Software and DevelopMentor. At present, SOAP enables distributed Web based interoperability between COM and DCOM applications across Win9X and WinNT platforms. Microsoft encourages also other platform (Java, CORBA) developers to use SOAP as the common denominator fully interoperable protocol for Web computing.

SOAP details are still unknown but the protocol is developed jointly with UserLand Software who previously released similar specification for RPC XML. It is therefore plausible that SOAP will share some features with this model which we describe in the following topic. Dave Wiener of UserLand Software developed Frontier – a scripting language for PC and Macintosh which is now often compared with Perl – the leading Web scripting technology under UNIX. Meanwhile, Larry Wall, the creator of Perl announced recently that he is working hard to make Perl the language of choice for XML scripting. Apparently, Perl and Frontier address the two extreme domains of Web hackers and end-users, and hence both technologies will likely coexist within the emergent WOM models for remote scripting.

XML RPC by UserLand Software, Inc. defines a simple RPC framework based on HTTP POST mechanism. Methods are developed in Frontier scripting language and servers have the necessary interpreter to execute them. A version of XML-RPC has been used in WebMethods B2B Integration Server. Combining this technology with Python (or other OO scripting language) with a suitable directory service (Naming) might make it a good alternative for programming platform on heterogenous machines based on a very simple API.
An XML-RPC message is an HTTP-POST request. The body of request contains the RPC in XML format. Client should put /RPC2 in the URI field of header so that the server can reroute this request to the RPC2 responder. If the server is only processing XML-RPC calls, then, client does not need to send /RPC2 in the URI field of header at all. Once body arrives to the RPC2 responder, it extracts the request body from XML format, processes the message(executing a script, accessing a database for a record, etc..) and sends the response back to the client within the body of HTTP Reply message with 200 OK status unless there is a lower-level error.

XML-RPC defines a serialization format for method calls and return values in XML. Each parameter / return value is tagged with <value> and its type can be one of the followings:

Data Type Tag	Explanation
<i4>	four-byte signed integer
<int>	four-byte signed integer
<boolean>	0(false) or 1(true)
<double>	double precision signed floating point number
<dateTime.iso8601>	Date/Time in ISO 8601 Ex: 199807117T14:08:55
<struct>	contains <members> tag and each member contains <name> and <value> tag
<array>	Contains <data> elements which contains multiple <value> elements

Note that <array> and <struct> tags allow nested definitions.

Method call message is defined with <methodCall> structure which contains name of the method with <methodName> tag and parameters of the method with <params> tag (where each parameter is tagged with <param> containing <value> tag). Response is defined with <methodResponse> tag and response can be one of the following formats: a) if the call succeeds, response may contain only one return value (only one <params> field with only one <param> field with only one <value>); b) if there is a failure, response contains <fault> tag which in turn contains <faultCode> and <faultString> items in its <struct> data type.

Web Interface Definition Language (WIDL) The purpose of the Web Interface Definition Language (WIDL) by webMethods, Inc. is to enable automation of all interactions with HTML/XML documents and forms, to provide a general method of

representing request/response interactions over standard Web protocols, and to allow the Web to be utilized as a universal integration platform. WIDL provides a programmatic interfaces which can allow to define and manage data (HTML,XML,text,etc.) and services (CGI, servlet, etc.).

Interface Definition Languages (IDLs) are used for defining services offered by applications in an abstract but highly usable fashion in all DOT technologies such CORBA, DCOM, and COM+. WIDL describes and automates interactions with services hosted by Web servers so that it transforms the Web into a standard integration platform and provides a universal API for all Web enabled systems. WIDL provides a metadata that describes the behavior of services hosted by Web servers. A service defined by WIDL is equivalent to a function call in standard programming languages. At the highest level, WIDL files describe: a) locations (URLs) of services; b) input parameters to be submitted (via GET and POST methods) to each service; c) conditions for successful/failed processing; and d) output parameters to be returned by each service.

Web based solution bypasses the firewall problems while providing strong security support. WIDL maps existing Web content into program variables allowing the resource of the Web to be made available, without modification, in formats well-suited to integration with *diverse business systems*. The use of XML to deliver metadata about existing Web resources can provide sufficient information to empower *non-browser applications* to automate interactions with Web servers so that they do not need to parse(or at least try) some unstructured HTML documents to find out the information they are looking for.
Defining interfaces to Web-enabled applications with XML metadata can provide the basis for a common API across lagacy systems, databases, and middleware infrastructures, effecively transforming the Web from an access medium into an integration platform.

WebBroker Current Web technology allows to publish documents and open channels for communication between distributed objects.

The goal of *Web Computing* exemplified by efforts such as WebBroker is to develop sytems that are less complicated than the current middleware technologies such as DCOM, DCE, CORBA, COM+,..etc. and more powerful than HTML forms and CGI scripts. In particular, the current HTML form POSTing system lacks a foundation for application specific security, scalability, and object interoperability.

WebBroker by DataChannel, Inc. is based on HTML, XML, and URIs, and it tries tries to address the following issues: a) the communication between software components on the Web; and b) the description of software components available on the Web. Therefore, XML is used by WebBroker for a) *Wire Protocol* i.e. the format of serialized method calls between software components; and b) *Interface* i.e. the format of documents which characterize the objects and the messages which can pass between them.

The style of inter-component communication is based on interfaces. An interface based distributed object communication enables a software object on one machine to make location-transparent method calls on a software object located on another machine. Advantages of WebBroker architecture can be enumerated as follows: a) *Unified Framework*. It unifies the Web browsing and distributed object computing. There is no need to ship additional protocols with browsers. b) *Thin clients*. Client side software needs to deal with only one protocol instead of supporting several protocols. This leads to a light client software architecture. C) *Better Security*. Using HTTP POSTs to enable communication between objects provide better framework for secured firewalls than POSTed HTML forms since posted document can be filtered more securely. (Processing binary data versus structured XML data)

In WebBroker, CORBA IDL files and COM TypeLibs can both be expressed as XML documents. Web native inter-ORB protocol (Web-IOP) can be defined as a common denominator between COM+, CORBA and Java based on HTTP, XML and URIs so that the interoperability can be provided. Notification is handled with small HTTP deamon on the client side. Using INVOKE method instead of GET and POST allows firewalls and proxies to let GET/POST messages go without controlling data while scrutinizing distributed object communications. Note that HTTP protocol still stays stateless with the INVOKE method.

Several main DTDs defined in WebBroker architecture can be summarized as follows: a) *PrimitiveDataTypeNotations:* primitive data type notations; b) *AnonymousData:* defines how to data-type XML elements; c) *ObjectMethodMessages:* defines document types (in XML) which are

serialized method calls and returns between objects. d) *InterfaceDef:* defines software component interfaces and the messages which can pass between them. This document can be used to generate proxy and skeleton implementation code.

5. Pragmatic Object Web – Integration Concepts and Prototypes

5.1 JWORB based Middleware

Enterprise JavaBeans that control, mediate and optimize HPcc communication 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) [13] - a multi-protocol Java network server that integrates several core services within a single uniform middleware management framework.

JWORB is a multi-protocol network server written in Java. Currently, JWORB supports HTTP and IIOP protocols, i.e. it can act as a Web server and as a CORBA broker or server. In the early prototyping stage is the support for the DCE RPC protocol which will also provide COM server capabilities. Base architecture of JWORB can be represented as a protocol detector plus a collection of dedicated servers for the individual protocols. Message packets in IIOP, HTTP and DCE RPC protocols all have distinctive anchors or magic numbers that allow for easy and unique identification of their protocols: IIOP packets always starts with the "GIOP" string, HTTP packages start with one of the protocol methods such as "POST", "GET" etc., and DCE RPC packets start with a numerical value (protocol version number). After the protocol is detected, the appropriate protocol handler code is dynamically loaded and the request handling thread is spawned for further processing.

JWORB is a useful middleware technology for building multi-server multi-vendor distributed object systems and bridges between competing disteributed object technologies of CORBA, Java, COM and the Web. For a user or system integrator who wants to support more then one such model in a given environment, JWORB offers a single server single vendor middleware solution. For server developers, JWORB offers an attractive economy model in which commonalities between server internals for various protocols can be naturally identified and the corresponding system services can be maximally reused when building the multi-protocol server. For example, CORBA services of JWORB can be naturally reused when building the new Web server extensions e.g. related to XML generation, parsing or flitering.

An early JWORB prototype has been recently developed at NPAC. The base server has HTTP and IIOP protocol support as illustrated in Figures 5.1 and 5.5. 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.

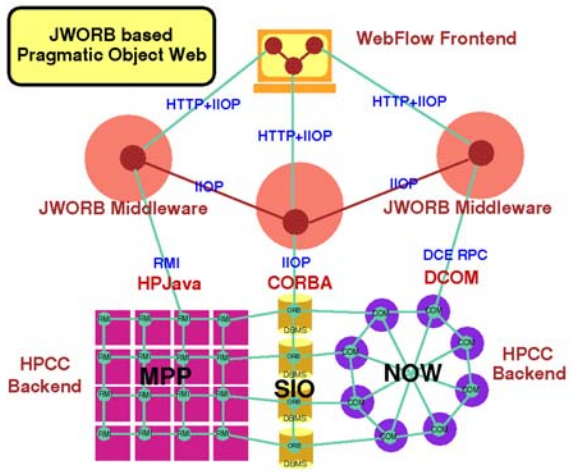

Figure 5.1: *Overall architecture of the JWORB based Pragmatic ObjectWeb middleware*

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 necessaryTester, Configuration and Mediator objects. Tester object inpects the currentnetwork package and it decides how to

interpret this particular message format. Configuration object is responsible for the configuration parameters of a particular protocol. Mediator objectserves 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 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.

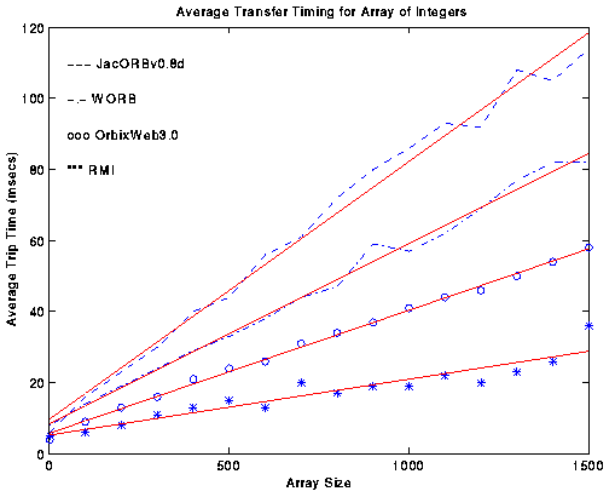

Figure 5.2: *IIOP communication performance for variable size integer array transfer by four Java ORBs: JacORB, JWORB, OrbixWeb and RMI. As seen, initial JWORB performance is reasonable and further optimizations are under way. RMI appears to be faster here than all IIOP based models.*

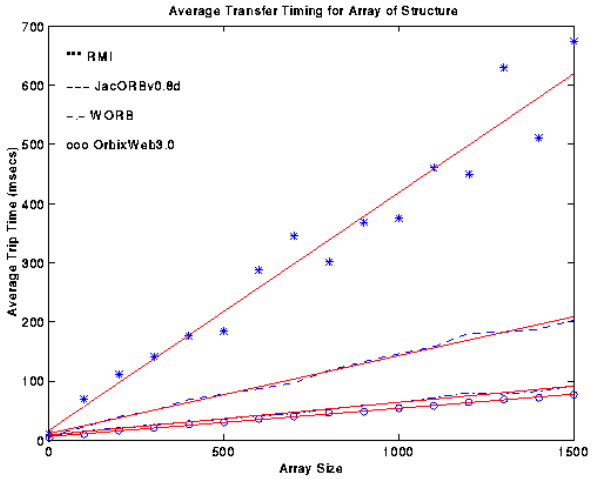

Figure 5.3: *IIOP communication performance for transferring a variable size array of structures by four Java ORBs: JacORB, JWORB, OrbixWeb and RMI. Poor RMI performance is due to the object serializationoverhead, absent in the IIOP/CDR protocol.*

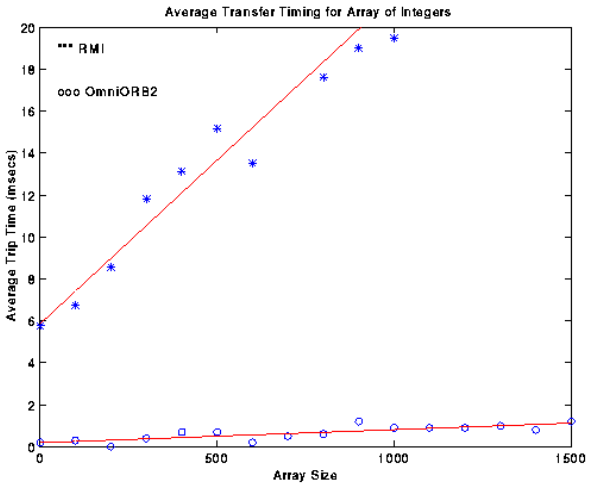

Figure 5.4: *Initial performance comparison of a C++ ORB (omniORB) with the fastest (for integer arrays) Java ORB (RMI). As seen, C++ outperforms Java when passing data between distributed objects by a factor of 20.*

The timing results presented in Figures 5.2-4 indicate that that JWORB performance is reasonable when compared withother 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 [2]. However, more study is 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 currently performs poorly for arrays of structures. JacORB uses object serialization also for arrays of primitive types and hence its performance is poor on both figures.

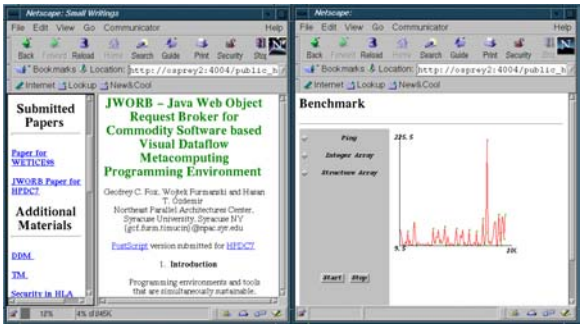

Figure 5.5: *A simple demo that illustrates interplay between HTTP and IIOP protocols in JWORB. A Netscape4 applet connets as ORBlet to JWORB and it displays real-time ping performance (right frame). During this benchmark, client connects to JWORB also via the HTTP channel by downloading a page (left frame) – this results in a transient performance loss in the IIOP channel, visible as a spike in the time-per-ping real-time plot in the right frame.*

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 [19][20] backend (see next Section) and the associated high performance remote I/O communication channels wrappedin terms of C/C++ ORBs (such as omniORB2[36]). 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.

5.2 RTI vs IIOP Performance Analysis

5.2.1 Image Processing

Further the discussion on comparison of RMI and IIOP, we proceed to choose a non-trivial application domain – Image Processing. Using the Java API for Image Processing, Image objects can be created from raw data. The raw data of an existing Image Object can then be examined, and filters created to have modified versions. These Image Objects can be used in exactly the same way as Image objects created by the Java run-time system, they can be drawn to a display surface, or the result of a filtering operation can be used as input for further image processing. Image data is used by objects, which implement the Image Consumer Interface. The Image Consumer & Image Producer are the basis for Image Processing using the AWT.

We implement the BathGlass Filter, which is normally employed to hide the identity of a person in television documentaries. The filter performing this transformation has been implemented both as a CORBA and RMI remote object implementation. We then proceed to compare the results we obtain in each case. The various issues compared are the initialization times, reference latencies, method invocation latencies with variable pixel arrays to the remote object implementation.

	Same Machine	Different Machines
Time to initialize the Object Request Broker	393 ms	475 ms
Time to bind to the Bathglass Filter Object Implementation	40 ms	180 ms
Time to invoke and complete remote operation	1.263 s	1.743 s

Figure 5.6 : *CORBA Imaging results' summary*

Figures 5.6 and 5.7 summarize the initialization and method invocation times when the object implementation resides on the same and different machines.

	Same Machine	Different Machines
Bind to the Bathglass Filter Object implementation	200 ms	614 ms
Time to invoke and complete Remote operation	246 ms	744 ms

Figure 5.7: *RMI Imaging results' summary*

Figure 5.8 provides a comparison of the average time to complete the remote operation, for a given number of (successive) remote method calls. As can be seen clearlythe RMI operation is approximately 50% faster.

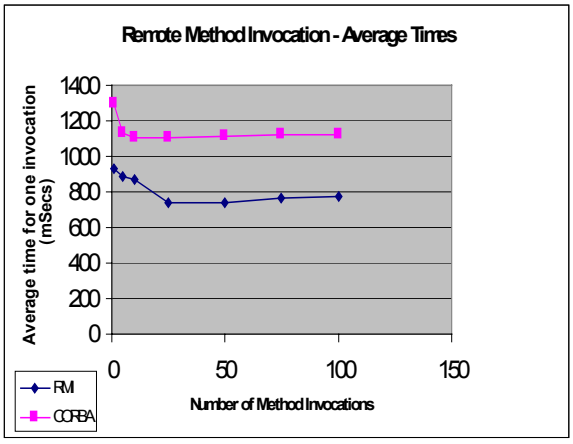

Figure 5.8: Remote Invocations (RMI vs CORBA)

Figure 5.9 demonstrates the average time for completion of remote operation for increasing image sizes, which translates to increasing sizes of the pixel array being sent as an argument in the method invocation. The graph is plotted for 3 different cases Local operation, RMI invocations and CORBA based invocation. As can be seen from the figure, as the size of the array increases RMI operations are approximately 50% faster than CORBA based operations.

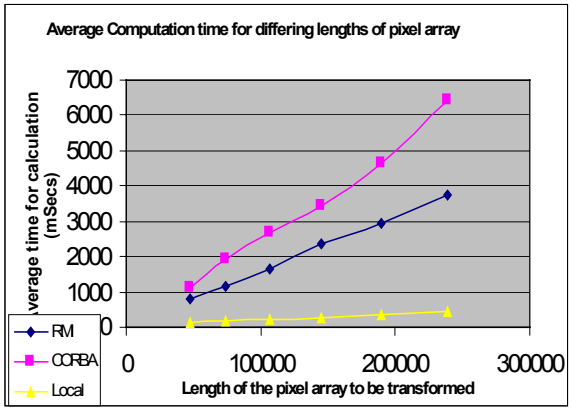

Figure 5.9: Average Times for different array sizes

5.2.2 Benchmarking Callbacks

So far all the measurements we have performed have been based on method invocations initiated by a client on the remote object implementation. However, the concept of callbacks is also a very powerful concept employed in distributed systems. Server callbacks are based on the notion that the server can notify the client about certain occurrences, instead of the client polling the server to check for the occurrence of the aforementioned event. This eliminates *busy waiting* and lends itself for use in asynchronous systems. The only pre-requisite for this operation is for the server to possess a remote reference to the client.

The callback model is prevalent in E-commerce solutions for profiling, personalization, promotions (instant rebates could be offered to a client, if there seems to be an indecision on his part) among others. In the cases mentioned above it's the business logic, which performs the callbacks on the client's profile (which is essentially a remote reference, as far as the server is concerned). The logic residing in the client performs the appropriate notification on the client's windowing system. The callback model also lends itself very naturally to building conferencing infrastructures, where a set of clients could register their remote reference with a central server, and any occurrence on one of the clients is reported to the others by a succession of callbacks by the server. In Java Distributed Collaborative Environment (JDCE) the answer we are looking for is callback performance for the RMI & CORBA case and important clues to their differences in behavior (if there is one). JDCE prototyped at NPAC is an attempt to explore these concepts and to evaluate the

feasibility (quantified in terms of high availability, scalability and fault tolerance) of such systems

Performance Analysis Figure 5.13 details the response times when CORBA & RMI clients are in collaboration mode. RMI (2,2) and CORBA (2,2) corresponds to the case where there are 2 data Bahns, with one CORBA and one RMI client registered to each bahn. RMI (2, 8) and CORBA (2,8) corresponds to the case where there are two data Bahns with each Bahn comprising of 4 RMI and 4 CORBA clients.

The graph indicates the fact that when the CORBA and RMI clients are in collaboration mode, with the respective worker threads being scheduled by the same thread scheduler, the RMI client receives the whole set of broadcast messages even though it was a CORBA client which initiated the broadcast. The graph also demonstrates that as the number of clients increases, as also the messages, the RMI based clients' response times are almost 200% faster than that of the CORBA clients.

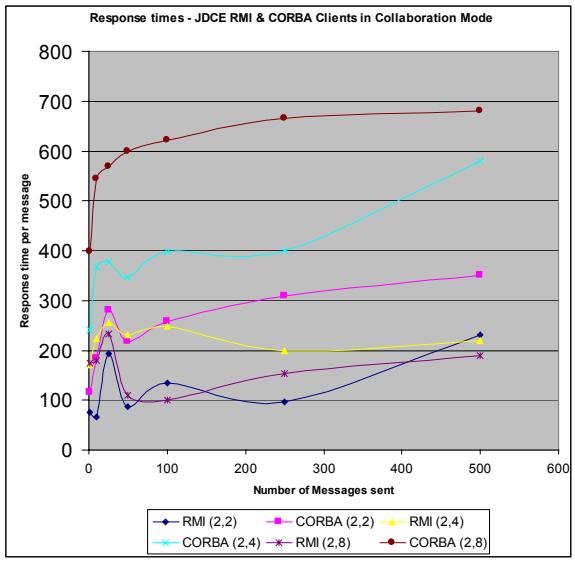

Figure 5.9: *Response times - JDCE RMI & CORBA Clients in Collaboration mode*

5.3 Wrapping Legacy Codes

So far, we discussed JWORB as our POW middleware server and JDCE as an example of a POW collaboratory service. In Section 6, we discuss another POW service – Object Web RTI (OWRTI) – that supports multi-player interactive gaming ranging from entertainment to military modeling and simulation applications. These three software entities: JWORB, OWRTI and JDCE were written by us from scratch using Java and following the specifications of the open standards of CORBA, HTTP and HLA. However, the full strength of the POW approach gets fully unleashed only after we activate the powerful capabilities of wrapping the multi-language legacy codes, available in POW via the CORBA bindings for the individual programming languages.

This process of wrapping, interfacing and including the external codes will typically involve other ORBs that cooperate with JWORB and support required languages and capabilities such as e.g. security, quality of service etc. As a simple example of such procedure, we already performed CORBA wrapping for the Jager video game, distributed by DMSO as part of the RTI release. DMSO Jager is written in C++ and it uses DMSO RTI (also written in C++) for its multi-player communication support. We repackaged the game so that it uses our Java based OWRTI communication. The C++/Java bridge was provided by JWORB from the Java side and omniORB2 public domain ORB by Oracle/Olivetti from the C++ side. A more detailed description of this C++ CORBA wrapping can be found in Section 6.2.1.

We currently start exploring the use of JWORB and the CORBA wrapping techniques for selected large scale application domains such as Landscape Modeling and Simulation (LMS) at CEWES or Quantum Monte Carlo simulation fo Nanomaterials at NCSA.

In Section 5.8, we also discuss our ongoing work on providing WebFlow based visual authoring support for CORBA modules. The end result will be a powerful 'Visual CORBA' enviroment that allows to import codes in any of the CORBA supported languages, to package them as WebFlow modules and to use visual composition tools to build distributed object applications in a user-friendly visual interactive mode.

5.4 Web Linked Databases

One legacy domain where the Web / Commodity computing community already developed and explored a broad range of Web wrapping techniques is the area of databases. Of particular interest here are commercial RDBMS systems, so far operated in terms of proprietary, vendor-specific form interfaces

and now enjoying the open Web browser based GUIs. However, HTML as the topmost and SQL as the bottommost scripting languages is often the only common feature of today's Web linked database technologies exploiting a range of middleware techniques such as PL/SQL, JDBC, LiveWire, ASP, ColdFusion, Perl etc.

We were recently involved in several Web linked database projects, conducted using various techniques for various communities and customers. Specific practical solutions and design decisions appear often as a result of compromise between several factors such as price vs. functionality tradeoffs for various technologies, project specific requirements, customer preferences etc. We indicate these tradeoffs when discussing the application examples below and then we summarize in the next section the lessons learned so far and our approach towards more uniform and universal, transparently persistent data-stores.

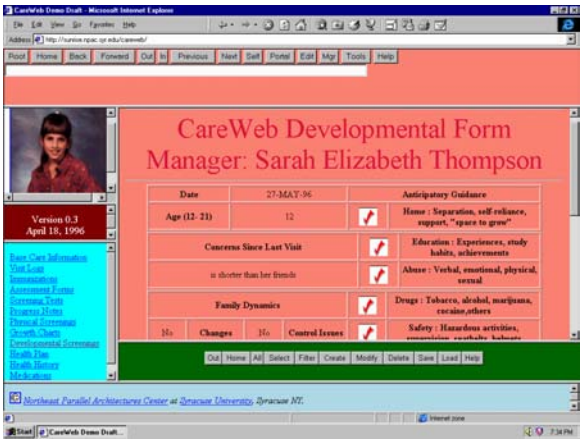

Figure 5.14: *Sample screendump from the early CareWeb prototype at NPAC – Netscape and JavaScript with VIC / VAT collaboratory tools (video, audio, chat, whiteboard) on top of Oracle Database with PL / SQL based HTML generator.*

Careweb is a Web based collaborative environment for school nurses with support for: a) student healthcare record database; b) educational materials for nurses and parents; c) collaboration and interactive consultation between nurses, nurse practitioners and pediatricians, including both asynchronous (shared patient records) and synchronous (audio, video, chat, whiteboard) tools.

Early CareWeb prototype (Figure 5.14) was developed at NPAC using Oracle7 database, PL/SQL stored procedures based programming model, and VIC/VAT collaboration tools. We found the Oracle7 model useful but hardly portable to other vendor models, especially after Oracle decided to integrate their Web and Database services.

New production version of CareWeb (Figure 5.15) under development by NPAC spin-off Translet, Inc. is using exclusively Microsoft Web technologies: Internet Explorer, Access/SQL Server databases, ASP/ADO based programming model, and NetMeeting based collaboration tools.

Figure 5.15: *Sample screendump from the production CareWeb system under development by Translet, Inc. – Internet Explorer 4 with NetMeeting collaboratory (video, audio, chat, whiteboard, shared browser) on top of Access or SQL Server database with Active Server Pages based HTML*

Although also hardly portable beyond the Microsoft software domain, we found this solution more practical from a small business perspective as Microsoft offers now indeed a total and affordably priced solution prototype for all tiers.

Language Connect University is another Web/Oracle service constructed by Translet Inc. for the distance education community. Marketed by the Syracuse Language Systems as an Internet extension of their successful CD-ROM based multimedia courses for several foreign languages, LCU offers a spectrum of collaboratory and management tools for students and faculty of a Virtual University. Tools include customized email, student record management, interactive multimedia assignments such as quizzes, tests and final exams, extensive query services, evaluation and grading support, course management, virtual college administration and so on (see Figure 5.17).

We found the Oracle based high end database solution for LCU displayed in Figure 5.16 as appropriate and satisfactory. Possible follow-on projects will likely continue and extend this model towards heterogeneous distributed database environment as shown in Figure 5.18 by adding suitable transparently persistent (e.g. CORBA PSS based) middleware and assuring compatibility with the emergent public standards for distance education such as IMS [X] by Educom.

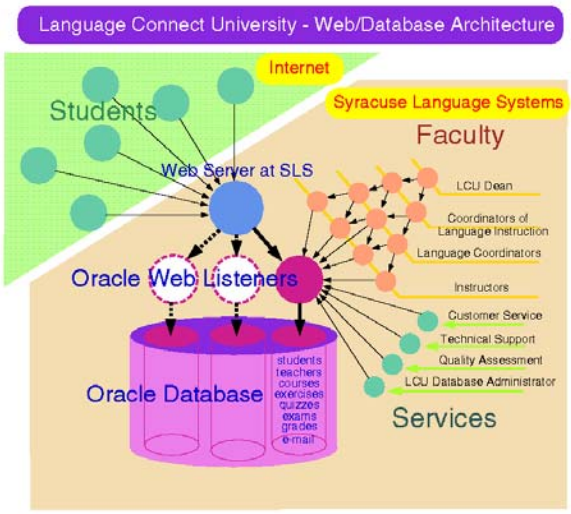

Figure 5.16: *Architecture of the LCU system. Oracle database maintains the course materials nad it provides administrative and content management functions for the Virtual University faculty. Remote students access the database using Internet Explorer Web browser, integrated (as an ActiveX control) with the CD-ROM multimedia application.*

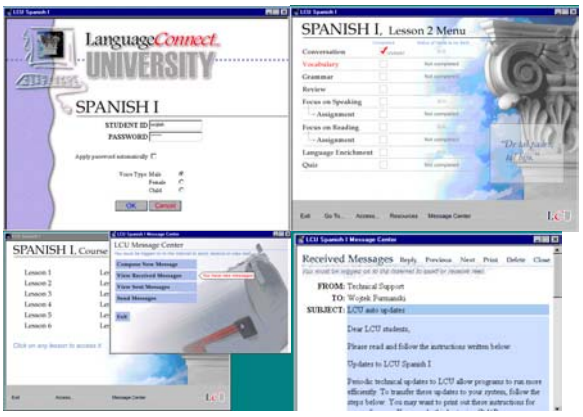

Figure 5.17: *Sample screendumps from the LCU system, illustrating the interplay of the CD-ROM client-side (upper frames) and the Web/Oracle server side (lower frames) processing; the latter includes support for interactive tests, quizzes, exams etc. as well as customized email handling. LCU email is used by students and teachers to communicate about various aspects of the course content, schedule, rules etc.*

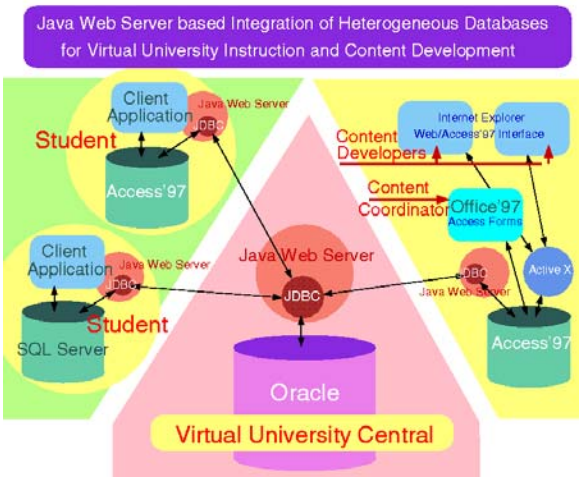

Figure 5.18: *Next step in the LCU evolution, moving towards a POW-like environment with heterogeneous distributed database support. Students download assignments from the central Oracle database to their local Access databases. Content developers use local commodity tools to build new courses and to upload the material to the central database.*

FMS Training Space [37] is an ongoing project at NPAC within the DoD Modernization Program that builds Web based collaboratory training environment for the FMS (Forces Modeling and Simulation) technologies under development by the program. We start with the SPEEDES training which will be gradually extended towards other FMS components such as Parallel CMS, E-ModSAF, HPC RTI, Parallel IMPORT and TEMPO/Thema. FMS Training Space combines lessons learned in our previous Web /Database projects such as CareWeb or LCU with our new WebHLA middleware based on Object Web RTI. The result will be a dynamic interactive multi-user system, with real-time synchronous simulation tools similar to online multi-player gaming systems, and with the asynchronous tools for database navigation in domains such as software documentation, programming examples, virtual programming laboratory etc. Selected screendumps from the preliminary version of the FMS Training Space, including some elements of the HLA, SPEEDES, CMS and ModSAF documentation databases and demonstrated at the DoD UGC 98 Conference [14], are shown in Figure 5.19 and in Section 6.3.1. Object Web RTI based real-time multi-player simulation support is being included in the FMS Training Space with early demos presented during the summer 98 conferences such as the SIW Fall 98 Conference in Orlando [21].

Current version of the FMS Training Space is using Microsoft Web technologies: Internet Information Server, Active Server Pages, Visual Basic Script, Internet Explorer, ActiveX and Java applet plug-ins, Front Page Web Authoring and Access Database. This approach facilitates rapid prototyping in terms of the integrated web/commodity tools from Microsoft and we intend to extend it in the next stage by adding the corresponding support for UNIX, Java, Oracle and Netscape users and technologies.

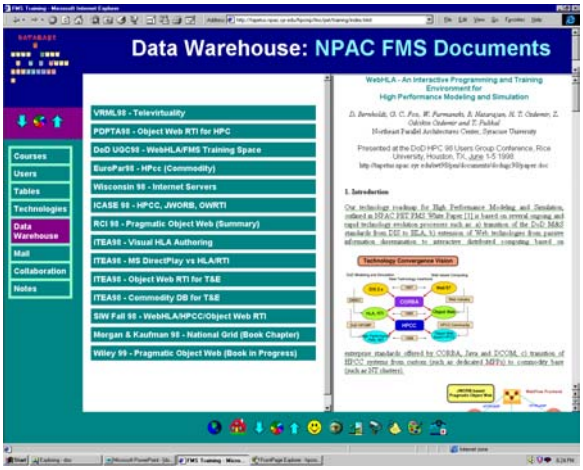

Figure 5.19: *Sample screendump from the FMS Training Space: data warehouse component with the database of NPAC FMS '98 publications*

5.5 Universal Persistence Models

In the ideal world, we could think of a univeral middleware data representation that would bind in some standardized fashion to the relational or object databases and to the flat file systems in the back-ends, and to the HTML in the front-end. This facilitates cross-vendor, cross-platform support for Web linked data-stores. In the real world, we have four competing middleware technologies: Java, CORBA, COM and WOM, and each of them offers or attempts at a different solution for the universal middleware data representation. Following our POW methodology, we summarize here the ongoing activities within the Java, CORBA, COM and WOM communities in the area of universal persistence frameworks, i.e. abstract data models that would span multiple vendors and various storage media such as relational or object databases and flat file systems.

JDBC and JavaBlend JavaSoft's Java Database Connectivity (JDBC) is a standard SQL database access interface for accessing a wide range of relational databases from Java programs. It encapsulates the various DBMS vendor proprietary protocols and database operations and enables applications to use a single high level API for homogenous data access. JDBC API defined as a set of classes and interfaces supports multiple connections to different databases simultaneously.

JDBC API mainly consists of classes and interfaces representing database connections, SQL statement's, result sets, database metadata, driver management etc. The main strength behind the JDBC API is its platform- and database-independence and ease of use, combined with the powerful set of database capabilities to build sophisticated database applications. The new JDBC specification (JDBC 2.0) which was released recently adds more functionality like support for forward and backward scrolling, batch updates and advanced data types like BLOB, and Rowsets which are JavaBeans that could be used in any JavaBean component development, etc. Other important features include support for connection pooling, distributed transaction support and better support for storing Java objects in the database.

Despite of the simplicity of use and the wide acceptability, the JDBC API has its own disadvantages. The API is primarily designed for relational database management systems and thus is not ideal for use with object databases or other non-SQL databases. Also, there are several problems due to various inconsistencies present in the driver implementations currently available.

JavaBlend, a high-level database development tool from JavaSoft that will be released this year, enables enterprises to simplify database application development and offers increased performance, sophisticated caching algorithms and query processing to offload bottlenecked database servers. It is highly scalable because it is multi-threaded and has built-in concurrency control mechanisms. It provides a good object-relational mapping that fits best into the Java object model.

UDA: OLEDB and ADO Universal Data Access (UDA) is Microsoft's strategy for high performance data access to a variety of information sources ranging from relational to object databases to flat file systems. UDA is based on open industry standards collectively called the Microsoft Data Access Components. OLEDB, which is the core of Universal Data Access strategy, defines a set of COM interfaces

for exposing, consuming and processing of data. OLEDB consists of three types of components: *data providers* that expose or contain data; *data consumers* that use this data; and *service components* that processes or transforms this data. The data provider is the storage-specific layer that exposes the data present in various data stores like relational DBMS, ORDBMS, flat files for use by the data consumer via the universal (store-independent) API.

OLEDB consists of several core COM components such as *Enumerators, Data Sources, Commands* and *Rowset*s. Apart from these components, OLEDB also exposes interfaces for catalog information or metadata information about the database and supports event notifications by which consumers sharing a rowset could be notified of changes at real time. Other features of OLEDB that will be added in future are interfaces to support authentication, authorization and administration as well as interfaces for distributed and transparent data access across threads, process and machine boundaries.

While OLEDB is Microsoft's system-level programming interface to diverse data sources, ActiveX Data Objects (ADO) offers a popular, high / application-level data consumer interface to diverse data. The main benefits of ADO are the ease of use, high speed, low memory overhead, language independence and other benefits that comes with the client side data caching and manipulation. Since ADO is a high-level programming interface application developers need not be concerned about memory management and other low-level operations. Some of the main objects in ADO Object model are: *Connection, Command, Recordset*, *Property, Field*.

CORBA Persistent State Service The initial CORBA standard that was accepted by OMG in the persistent objects domain was the Persistent Object Services (POS). The main goals for such a service were to support corporate centric data-stores and to provide a data-store independent and open architecture framework that allows new data-store products to be plugged in at any time. POS consisted of four main interfaces namely, the Persistent Object interface (PO) that the clients would implement, Persistent Idinterface (PID) for identifying the PO object, Persistent Object Manager interface (POM) that manages the POS objects and the Persistent Data Service interface (PDS) which actually does the communication with the data-store.

Although this specification was adopted more than three years ago, it saw very little implementations because of it's complexity and inconsistencies. The specification also exposed persistence notion to CORBA clients which was not desirable and the integration with other CORBA services were not well defined. Thus OMG issued a new request for proposal for a new specification, *Persistent State Service (PSS)* that is much simpler to use and to implement and is readily applicable to existing data stores.

The Persistent State Service specification, currently still at the level of an evolving proposal to OMG led by Iona / Orbix, uses the value notation defined in the new Objects by Value specification for representing the state of the mobile objects. The PSS provides a service to object implementers, which is transparent to a client. This specification focuses on how the CORBA objects interact with the data-store through an internal interface not exposed to the client. The persistent-values are implemented by application developers and are specific to a data-store and can make use of the features of the data-store. The specification also defines interfaces for application-independent features like transaction management and association of CORBA objects with persistent-values.

Web Object Model World-Wide Web Consortium (W3C) develops a suite of new Web data representation and/or description standards such as XML (eXtensible Markup Language), DOM (Document Object Model) or RDF (Resource Description Framework). Each of these technologies has merit in its own but when combined they can be viewed collectively as a new, very dynamic, flexible and powerful *Web Object Model* (WOM) [33].

XML is a subset of SGML that acts as a metamodel for specialized markup languages i.e. it allows to define new custom / domain specific tags and document templates or DTDs (Document Type Definitions). Such DTDs provide a natural bridge between Web and Object Technologies since XML documents can be now viewed as instances of the associated DTD classes. DOM makes such analogy even more explicit by offering an orthodox object-oriented API (specified as CORBA IDL) to XML documents. Finally, RDF offers a metadata framework that allows association of a set of named properties and their values with a particular Web resource (URL). In the WOM context, RDF is used to bind in a dynamic and transient fashion the Web Object methods located in some programming

language files with the Web Object states, specified in XML files.

Summary As seen from the above discussion, the universal data models are just emerging. Even if most major RDBMS vendors are OMG members, the consensus in the CORBA database community is yet to be reached. WOM is a new, '97 / '98 concept and several aspects of and relations between the WOM components listed above are still in the design stage. However, given the ongoing explosion of the Web technologies, one can expect the WOM approach to play a critical role in shaping the universal persistence frameworks for the Internet and Intranets. At the moment, the single vendor models such as JDBC by Sun and OLEDB by Microsoft are ahead the consortia frameworks and in fact the Microsoft UDA solution is perhaps the most complete and advanced as of mid '98.

5.6 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.

In Figure 5.20, 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 one of the initial entry points towards 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 Figure 5.20.

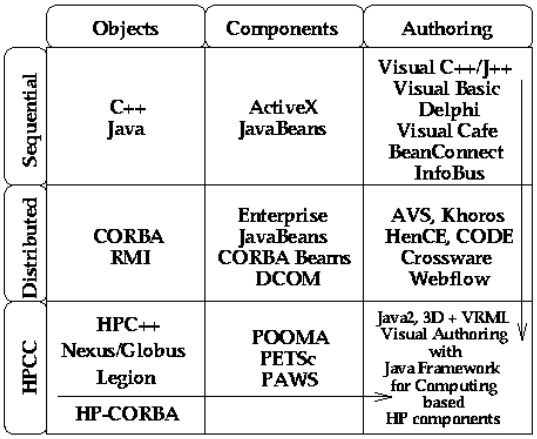

Figure 5.20: *System Complexity (vertical axis) versus User Interface (horizontal axis) tracking of some technologies*

Currently NPAC's WebFlow system [38] 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 Figure 3.4, using a mesh of JWORB servers to manage a recourse pool of distributedHPcc components. An early version of such JWORB based WebFlow environment is in fact operational at NPAC and we are currently building the Object Web management layer including the Enterprise JavaBeans based encapsulation and communication support discussed in the previous section.

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.

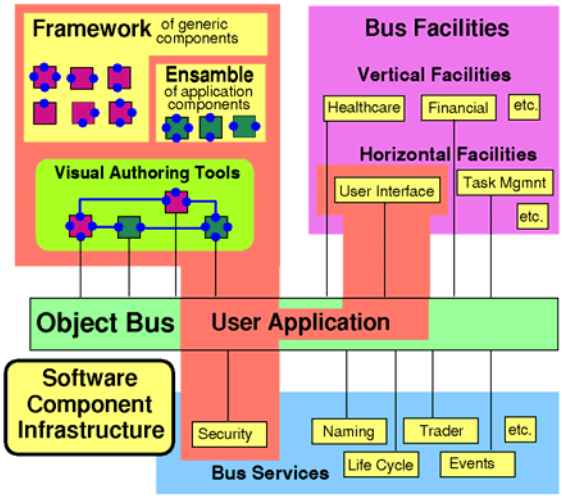

Figure 5.21: *Visual Authoring with Software Bus Components*

We then generate an environment such as in Figure 5.21 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 Figure 5.21, 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.

5.7 WebFlow – Current Prototype

We describe here a specific high-level programming environment developed by NPAC - WebFlow [1][15][38] - 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 [3]. 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 goal's which exposes these stable generic characteristics of WebFlow.

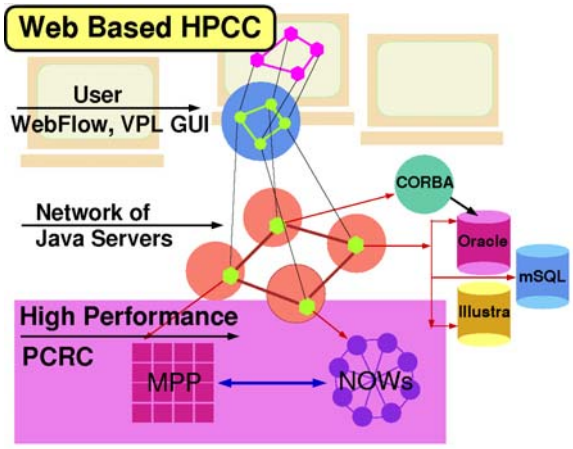

Figure 5.22: *Top view of the WebFlow system: its 3-tier design includes Java applet based visual graph editors in tier 1, a mesh of Java servers in tier 2 and a set of computational (HPC, Database) modules in tier 3.*

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*" (client-server) instances of the computational Web such as CGI scripts, Java applets or ActiveX controls is the "horizontal" (server-server) inter-modular connectivity (see Figure 5.22). This is 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* - post-processing 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 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).

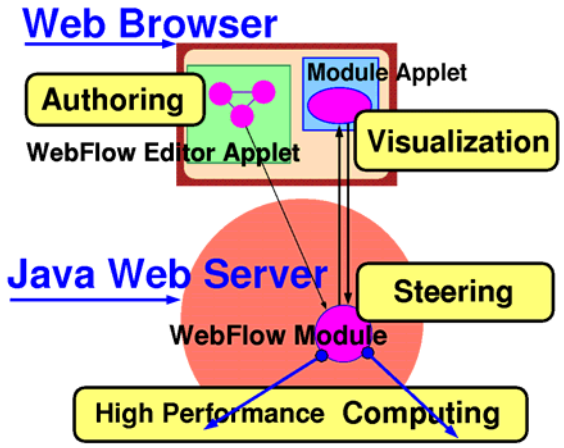

Figure 5.23: *Front-end perspective on the 3-tier architecture of the WebFlow systerm. WebFlow. WebFlow module is instantiated by the WebFlow editor applet, it connects to the HPC backend and it optionally spawns a module specific front-end applet used for the visualization or runtime steering purposes.*

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 (Figure 5.24).

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 (Figure 5.24).

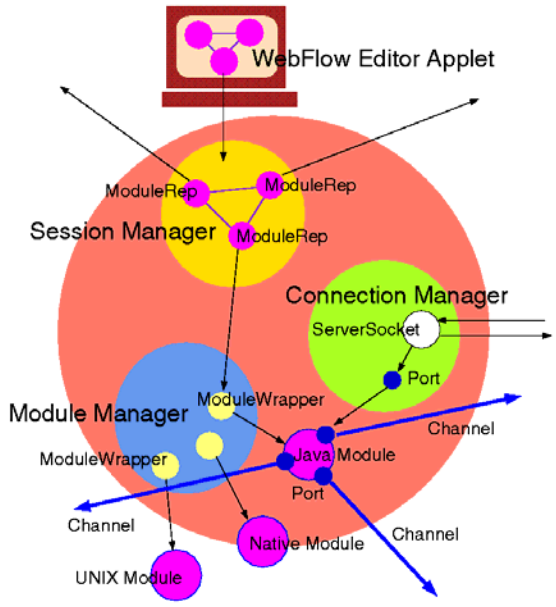

Figure 5.24: *Architecture of the WebFlow server: includes Java servlet based Session, Module and Connection Managers responsible for interacting with front-end users, backend modules and other WebFlow servers in the middleware.*

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 Session Manager, and from there to the Connection Managers in charge of thesemodule 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 theirrequired 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.

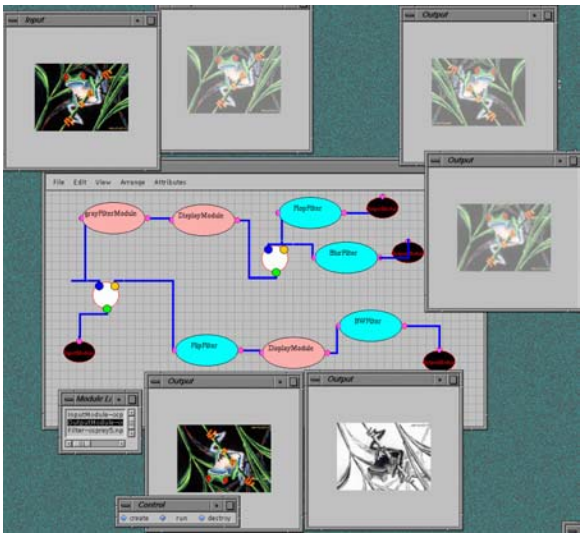

Figure 5.25: *A sample WebFlow application in the imaging domain: an input image stream is forked via multiplexing modules (white blobs) and sent to a set of imaging filters and then to the output modules, displaying the processed images in their visualization applets.*

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.

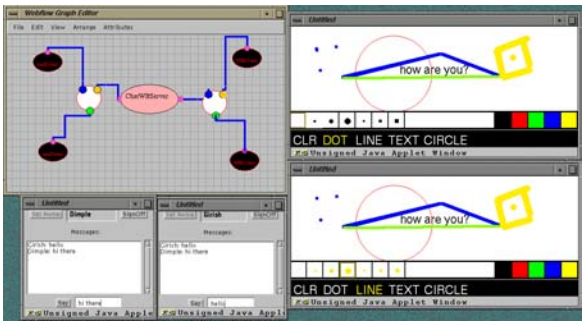

Figure 5.26: *A sample WebFlow application for visual composition of a collaboratory session. Collaboratory server (based on JSDA technology from Sun Microsystems) and a set of standard collaboratory tools (chat, whiteboard) are mapped on WebFlow modules. WebFlow dataflow editor is used to select required tools, participants and to establish required connectivities for a particular collaboratory session and channels.*

5.8 WebFlow meets CORBA and Beans

We are now extending the 100% pure Java WebFlow model described above by including support for CORBA modules and the associated bindings for other programming languages. In this section, we summarize the current status of this ongoing work.
In the standards based approach to WebFlow, we employ CORBA in the implementation of the back-end. One of the primary goals of WebFlow is to provide a language independent interaction between module (units of computation). This translates into the ability of the run-time system to cascade computational units implemented in different languages and perform invocations transparently. CORBA can be viewed as an environment to support the development of a complete new system or an environment for integrating legacy systems and sub-systems. An implementation of the standard defines a language and platform-independent object bus called an ORB (Object Request Broker), which lets objects transparently make requests to, and receive responses from, other objects located locally or remotely. Each implementation of the CORBA standard is able to communicate with any other implementation of the standard, regardless of the language used to implement the standard. This stated, it should be easy to visualize the situation where some modules could published to a C++ ORB, some to a Java ORB and the utilization of these modules in some computation. Next, we employ the CORBA Naming service to create a remote information base of the various modules available for computation.

The CORBA Naming Service The Naming Service provides the ability to bind a name to an object relative to a naming context. A naming context is an object that contains a set of name bindings in which each name is unique. To resolve a name is to determine the object associated with the name in a given context. Through the use of a very general model and dealing with names in their structural form, naming service implementations can be application specific or be based on a variety of naming systems currently available on system platforms.

Graphs of naming contexts can be supported in a distributed, federated fashion. The scalable design allows the distributed, heterogeneous implementation and administration of names and name contexts. Because name component attribute values are not assigned or interpreted by the naming service, higher levels of software are not constrained in terms of policies about the use and management of attribute values.

Through the use of a "names library," name manipulation is simplified and names can be made representation-independent thus allowing their representation to evolve without requiring client changes. Application localization is facilitated by name syntax-independence and the provision of a name "kind" attribute.

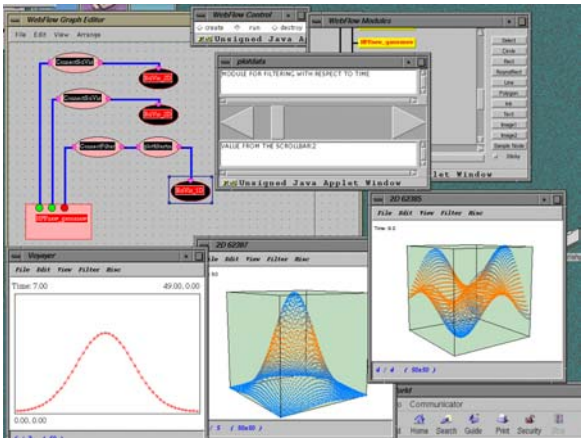

Figure 5.27: *Sample screendump from the WebFlow demo presented at Supercomputing '97: a set of real-time visualization modules is attached to the HPC simulation module (Binary Black Holes) using WebFlow visual dataflow editor (upper left frame).*

WebFlow Widgets A *Connection Object* represents each visually composed computational graph. The Resource Description Format (RDF[39]) is used for capturing metadata about the graph. The validity of the inter-connections within the computational graph is check by the *Connection Manager*. An exception hierarchy is in place to respond to failure methods in invocation or location of the distributed modules. A *module* is a nugget of computation implementing the Module interface. We have an interface inheritance mechanism whereby specific application domains could extend the Module interface to facilitate initializations and retrieval of results. Thus in the Imaging Domain, one would have a method for setting and getting pixels, while employing the run() method in the Module Interface to effect the filtering operation. The *Session Manager* provides information about the namespace and aids in the process of information mining and execution of connection graphs within the name space. Besides this WebFlow also has the notion of *Composite Modules*, which could be published to the Naming Service and thus, be available for future reference. It should be noted that since the composite module

inheritance tree includes the Module interface, the run() method could be invoked as if it were a module.

All modules conform to the Java Beans naming conventions and provide additional component information through the BeanInfo class. Intuitively, it follows therefore that, using the Java core reflection and Beans introspection, the properties, methods and other attributes could be discovered at run-time and connections can be made across module methods. To put it briefly WebFlow extends the Visual Programming paradigm to the distributed object world, and provides the end user with the tools and means to build sophisticated distributed applications without writing a single line of code.

5.9 WebFlow – Next Steps Towards Visual POW

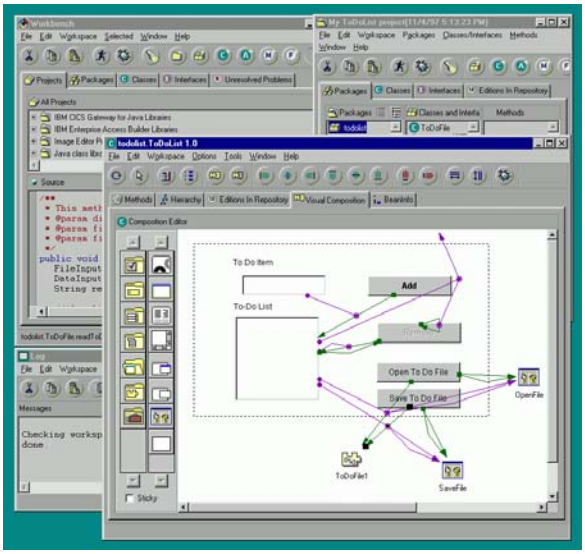

Figure 5.28: *Sample screendump from the VisualAge for Java – a visual authoring tool for Java applications, applets and beans from IBM. The editor panel illustrates GUI applet under construction with nodes representing visual controls and with connection lines representing user or system events.*

Our current visual metaphor in WebFlow is reflecting today's industry standards and best practice solutions, developed and tested by systems such as AVS and Khoros. Thus it is based on visual interactive graph editing with graph nodes representing modules/operations and graph links representing connections and dataflow channels. However, the new generations of client technologies such as JavaBeans or ActiveX admit several more elaborate and dynamic visual paradigms in which module icons are replaced by full-blown applets, OLE controls or other dynamic GUI widgets. Further, fixed type data channels are replaced by dynamic event/listeners or consumer/producer or publisher/subscriber associations.

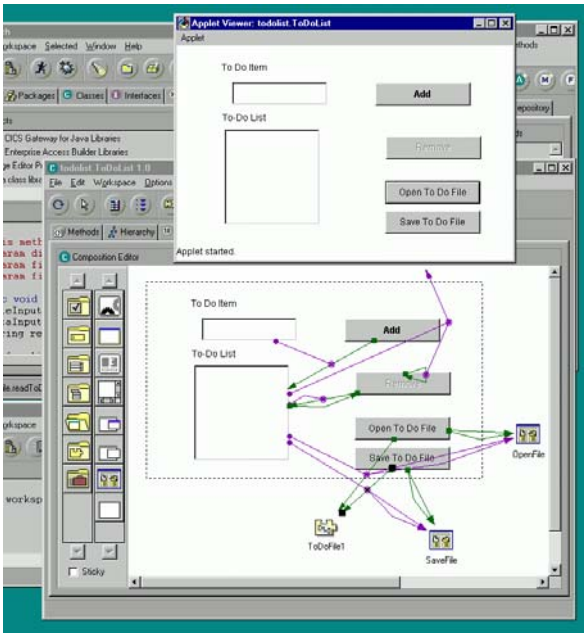

Figure 5.29: *Sample screendump from the VisualAge for Java – a visual authoring tool for Java applications, applets and beans from IBM. The editor (lower) panel is described in Fig. 41; the popup window (upper panel) represents the actual GUI control constructed in the visual mode and wired as specified on the editor panel.*

As part of a study of new visual programming standards, we have analyzed recently a suite of new tools appearing on the rapidly growing visual authoring market, including VisualStudio from JavaSoft, VisualAge from IBM, VisualCafe from Symantec, Rational Rose from Rational and VisualModeler from Microsoft. It appears that the visual componentware authoring products are still based on early custom models but there is already a promising stable standard initiative in the area of visual object-oriented design, represented by Uniform Modeling Language (UML).
UML is developed by an industry consortium led by Rational and supported by Microsoft, and was recently adopted as CORBA standard by OMG (together with the associated Meta Object Facility). Hence, we are witnessing here one distinctive example of a truly universal emerging standard candidate agreed upon by all major players. UML offers a broad spectrum of visual authoring graph topologies, adequate for various stages of the software process and representing diverse views of the modeling system such as use-case view, logical

view, component view, concurrency view or deployment view.

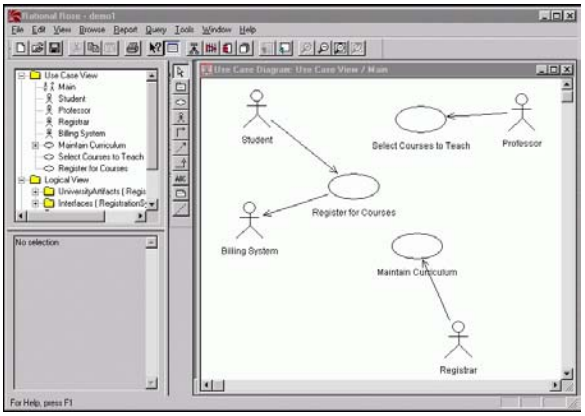

Figure 5.30: *A sample Use Case UML diagram. Typical visual icons here include actors (i.e. user types e.g. student, professor) and their typical actions (e.,g. select course, register etc.). Use case diagrams are helpful in the early design stage to grasp the customer requirements and to provide input for a more formal OOA&D process that follows and other UML diagams topologies described and illustrated below.*

The rich collection of UML graph topologies includes: use-case diagrams (for sketching initial customer requirements), class diagrams (with inter-class relationships such as inheritance or aggregation), object diagrams (with object methods and attributed), state diagrams (visualizing stateful objects are final state machines), sequence diagrams (exposing temporal aspects of object interactions), collaboration diagrams (exposing the spatial/topological aspects of object interactions), activity diagrams (representing the workflow and concurrency relations between object methods), component diagrams (with the actual software entities as nodes), deployment diagrams (with the physical hardware platforms as nodes).

UML is a formally specified open extensible standard. The model is being rapidly adopted by the industry and is already triggering the first generation commodity tools for advanced visual authoring such as Rational Rose from Rational or VisualModeler from Microsoft. These tools are currently focused on fine grain object level visual software development and they offer 'round-trip engineering' support. Visual models can be converted into language specific code stubs/skeletons, and the full codes (both pre-existing and visually developed) can be reverse-engineered to expose their internal structure and to enable further editing in terms of UML graph topologies discussed above.

To our knowledge, current UML tools do not address yet the runtime visual authoring via coarse grain module composition as offered by WebFlow, but such dynamic extensions are natural in UML. We therefore propose to adopt UML as WebFlow tier 1 visual standard and to extend the language using its intrinsic mechanisms so that both the modeling/development time and simulation/runtime visual frameworks are supported within a uniform and consistent standard paradigm. In fact, current WebFlow model can be viewed as a runtime specialization of the UML activity diagrams - hence the dataflow model of WebFlow as derived from AVS/Khoros embeds naturally in UML and it gets reinforced by several other views and complementary graph topologies discussed above.

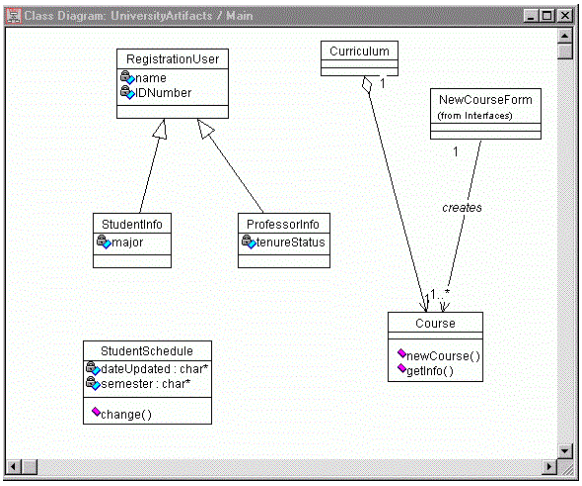

Figure 5.31: *A sample UML Class diagram. Boxes represent classes and include attribute lists, connection lines represent inheritance (super, sub) or aggregation / containment (parent, child) relations.*

UML activity diagrams represent also the first step towards visual standards for parallel programming. The dataflow model itself is based on the OR-gate synchronization (a module responds to data changes on any of its input channels). UML also adds AND-gate synchronization and some other visual parallel constructs (such as implicit iterator that implies concurrent replicas of a marked branch of the graph).

The current early visual parallel processing support in UML is clearly driven by the SMP architectures. Some other previous efforts within the HPCC community such as HeNCE or CODE or more recently Cumulvs are addressing visual concurrency for shared memory massively parallel and distributed processing architectures. We believe that there are

interesting research issues in reexamining these approaches from the UML standard perspective in search of new universal visual extension patterns that would allow one to express platform-independent concurrency in terms of suitable visual controls.

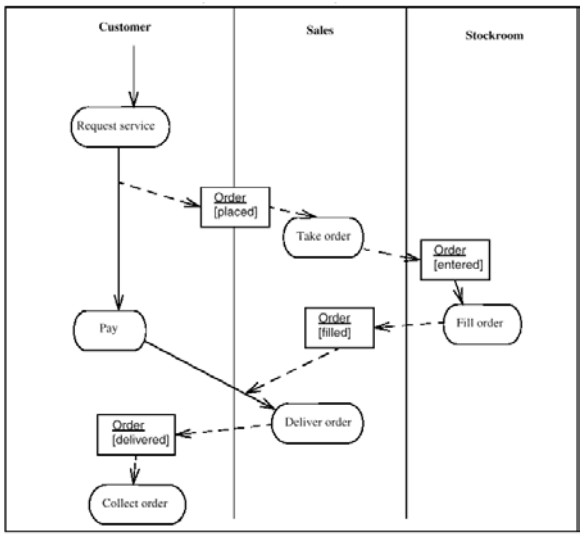

Figure 5.32: *Sample UML Activity Diagram. Columns (verical strips) are mapped on the individual objects, nodes represent methods and connectio n lines represent events, dataflow or argument passing associated with method invocation.*

The complex process of building HPDC software involves several natural hierarchy levels, starting from individual functions/methods, grouped into sequential and parallel objects, then packaged as single- or multi-server modules, and finally composed as local or distributed components to form metacomputing applications. Previous visual approaches addressed selected aspects of but none has succeeded yet in capturing the totality of this complex process.

The unique position of UML is its proposed 'complete solution', realized in terms of a comprehensive set of views, diagrammatic techniques and graph topologies. In the context of WebFlow, it offers a promising common visual platform for (parallel) module developers and (distributed) module integrators. Each of these groups will adopt the most adequate techniques from the UML arsenal, while maintaining the interaction via overlapping topologies such as activity diagrams (that allow one to mix object-oriented, functional and concurrent processing perspectives).

Parallel module developers will be likely using class and object diagrams on a regular basis. They may also find useful the collaboratory diagrams which, can be used. to visualize spatial collaboration of classes implementing a complex PDE stencil. The sequence or state diagrams could perhaps be useful to detect a deadlock via rigorous temporal or/and state analysis. Expression template programmers might also find useful the UML template notation and provide us with the specific extension requirements in this area to visually represent complex expression trees or to visualize/optimize the intermediate steps of the compiler parse tree.

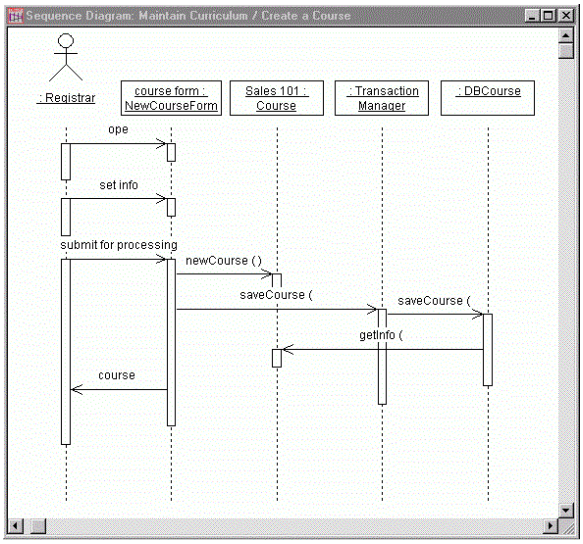

Figure 5.33: *Sample UML Sequence Diagram. Columns (vertical lines) represent objects as in Activity Diagram in Figure 5.32, horizontal lines represent method calls, vertical boxes represent transient / persistent states and the computation time is manifest, flowing vertically down.*

Moving further down in the software hierarchy towards finer elements such as individual methods or the actual lines of the source code, we will provide direct customizable links from visual code icons to developer's favored code editors and debuggers. These could include vanilla public domain tools such as vi or emacs; commodity tools of growing popularity such as MS Visual Studio or Rational Rose; and specialized tools for high performance software prototyping such as our DARP package, so far tested in the HPF programming environment.

More generally, we will be working with various groups of WebFlow users to collect their comments, suggestions and requirements for the UML support in the WebFlow front-end. Based on this information, we will provide core UML support for parallel languages such as C++, HPC++, and HPF as well as for parallel objects in frameworks such as

POOMA[42]. The WebFlow/UML frontend will be packaged as a Java/CORBA facility that can be used in a standalone mode as a visual parallel authoring toolkit for full-cycle and round-trip software engineering. It can also be offered as a parallel processing extension to commercial authoring tools such as Rational Rose, Visual Modeler and others to come over the next two years. These delivery possibilities will be refined in our discussions with users of our system.

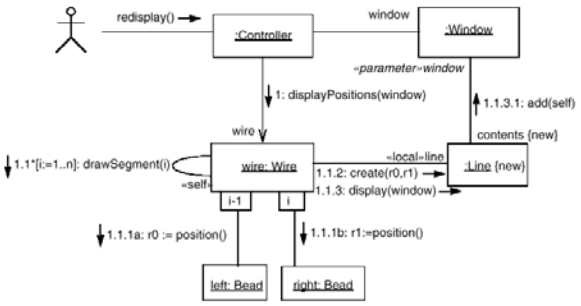

Figure 5.34: *Sample UML Collaboratory Diagram. Nodes represent objects and lines represent method calls as in the Sequence Diagram in Figure 5.33 but time is less explicit here (with time order marked only by integer labels). This technique is used to explose collaboration topologies involving sets of objects (such as a feedback loop in the diagram above).*

Finally, we note that the 'round-trip engineering' requires language compiler front-ends and we intend to provide and use here our suite of parallel compiler front-ends developed by the PCRC[41] led by NPAC.

6. Example of POW Application Domain - WebHLA

6.1 Introduction to WebHLA

The technology roadmap for High Performance Modeling and Simulation (Figure 6.1) which underlies our FMS PET program within the DoD HPC Modernization Program explores synergies between ongoing and rapid technology evolution processes such as: a) transition of the DoD M&S standards from DIS to HLA; b) extension of Web technologies from passive information dissemination to interactive distributed object computing offered by CORBA, Java, COM and W3C WOM; and c) transition of HPCC systems from custom (such as dedicated MPPs) to commodity base (such as NT clusters).

One common aspect of all these trends is the enforcement of reusability and shareability of products or components based on new technology standards. DMSO HLA makes the first major step in this direction by offering the interoperability framework between a broad spectrum of simulation paradigms, including both real-time and logical time models.

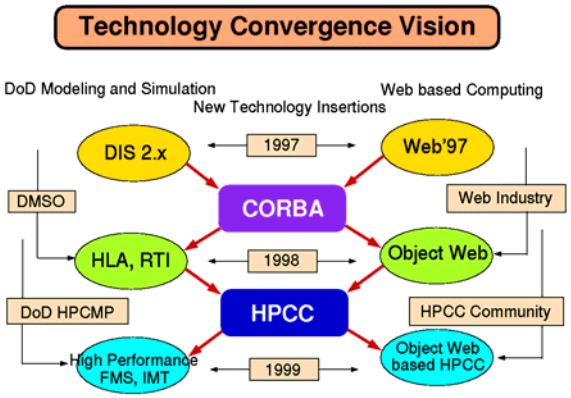

Figure 6.1: *Web/Commodity and DoD M&S Technology Evolution Roadmap which underlies our WebHLA, approach: both domains are switching now to distributed object technologies (CORBA insertion) and will soon acquire High Performance Commodity Computing capabilities, e.g. via NT clusters.*

However, HLA standard specification leaves several implementation decisions open and to be made by the application developers - this enables reusability and integrability of existing codes but often leaves developers of new simulations without enough guidance. In WebHLA, we fill this gap by using the emergent standards of Web based distributed computing – we call it *Pragmatic Object Web* [3][2] - that integrate Java, CORBA, COM and W3C WOM models for distributed componentware.

Traditional HPCC, dominated by data parallel MPP didn't make significant inroads into the DoD M&S where the focus is on task parallel heterogeneous distributed computing. Recent trends towards commodity based HPCC systems such as NT clusters offer a new promising framework for new generation high performance high fidelity M&S environments such as addressed by JSIMS, JWARS, JMASS or Wargame2000 programs.

We therefore believe that WebHLA, defined as the convergence point of the standardization processes outlined above will 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 are addressing WebHLA design and prototype development in a set of PET FMS tasks at ARL and CEWES, including: JWORB based Object Web RTI (Section 6.2.1), WebFlow based Visual Authoring Tools (Section 6.2.2), Data Mining (Section 6.2.4) and HPC Simulation back-ends (Section 6.2.3), and DirectX based multi-player gaming front-ends (Section 6.2.5).

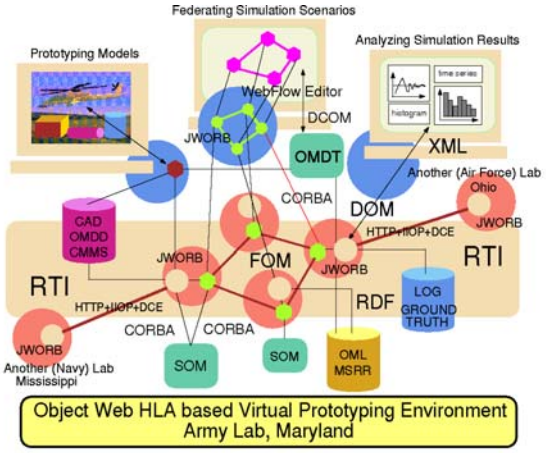

Figure 6.2: *The overall architecture of our WebHLA prototype follows the 3-tier architecture of our Pragmatic Object Web [3] (see Fig. 2) with the RTI-over-JWORB based middleware, backend simulation modules (given by CMS, ModSAF etc. libraries, wrapped via CORBA/COM as FOM or SOM objects) and with the WebFlow/DirectX based visual authoring / runtime front-ends.*

In the following, we describe in more detail in Section 6.2 the WebHLA components listed above, followed by a brief overview in Section 6.3 of the emergent WebHLA application domains such as: Distance Training, Resource Management for Metacomputing and/or Commodity Clusters, and Simulation Based Acquisition

6.2 WebHLA Components

6.2.1 Object Web RTI

Current HLA is a custom distributed object model but DMSO's longer-range plan includes transferring HLA to industry as CORBA Facility for Modeling and Simulation.

Anticipating these developments, we have recently developed in one of our HPCMP FMS PET projects at NPAC an Object Web based RTI [6][8] prototype, which builds on top of our new JWORB (Java Web Object Request Broker) middleware integration technology.

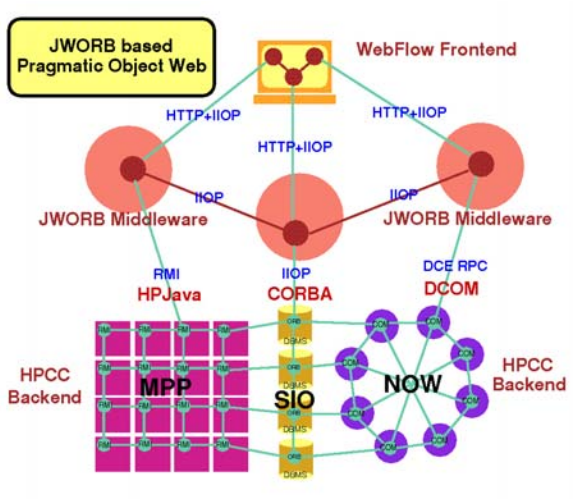

Figure 6.3: *Illustration of the communication protocol integration within our JWORB based Pragmatic Object Web. JWORB uses Java to integrate HTTP with IIOP and then it connects with NT clusters via COM/CORBA bridge.*

JWORB (Figure 6.3) is a multi-protocol Java network server, currently integrating HTTP (Web) and IIOP (CORBA) and hence acting both as a Web server and a CORBA broker. Such server architecture enforces software economy and allows us to efficiently prototype new interactive Web standards such as XML, DOM or RDF in terms of an elegant programming model of Java, while being able to wrap and integrate multi-language legacy software within the solid software-engineering framework of CORBA.

We are now testing this concept and extending JWORB functionality by building Java CORBA based RTI implementation structured as a JWORB service and referred to as *Object Web RTI* (Figure 6.4). Our implementation includes two base user-level distributed objects: RTI Ambassador and Federate Ambassador, built on top of a set of system-level objects such as RTI Kernel, Federation Execution or Event Queues (including both time-stamp- and receive-order models). RTI Ambassador is further decomposed into a set of management objects, maintained by the Federation Execution object, and including: Object Management, Declaration Management, Ownership Management, Time Management and Data Distribution Management.

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 federates 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 re-using these features, already present in the CORBA model.

To be able to run C++ RTI demo examples, we developed a C++ library which: a) provides HLA RTI v1.3 C++ programming interface; and b) it is packaged as a CORBA C++ service and, as such, it can easily cross the language boundaries to access Java CORBA objects that comprise our Java RTI. Our C++ DMSO/CORBA glue library uses public domain OmniORB2.5 as a C++ Object Request Broker to connect RTI Kernel object running in Java based ORB and runs on Windows NT and Sun Solaris 2.5 environments. Through library, user defines the RTIambassador object as usual but in the implementation it actually accesses the OWRTI CORBA service and executes each call on this service. Similarly, user supplied federate ambassador object is managed by a CORBA Object which leaves in the client side and forwards all the call it received from RTI to the user's federate ambassador object.

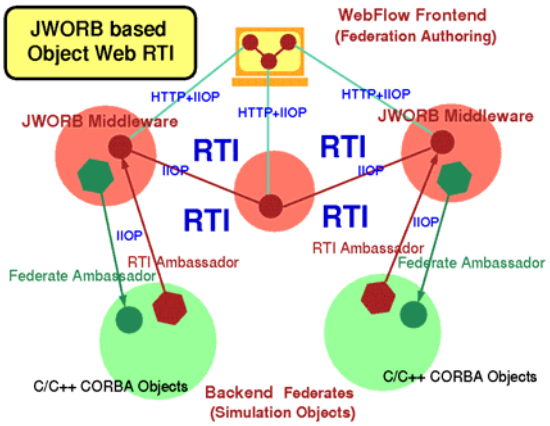

Figure 6.4: *Top view representation of Object Web RTI: RTI Ambassador is Java CORBA object maintained by JWORB middleware; Federate Ambassador is (typically C++) CORBA object maintained by the backend federate; WebFlow front-ends tools are available for visual authoring of federation configuration.*

6.2.2 Visual Authoring Tools for HLA Simulations

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 (Figure 6.5). 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 (Figure 6.6)

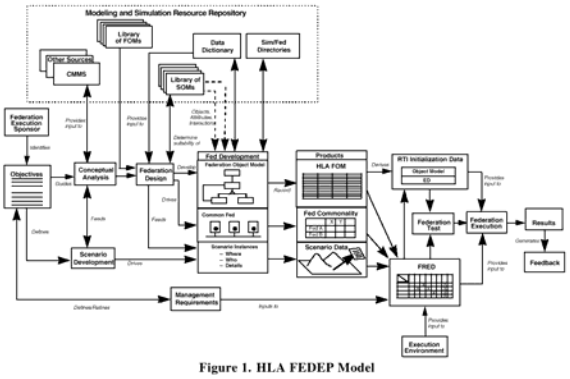
Figure 1. HLA FEDEP Model

Figure 6.5: *DMSO FEDEP Process including databases, authoring, configuration management, runtime and analysis modules.*

To provide a natural linkage between DMSO DIF and the emergent Web data interchange format standards to be formulated within the XML framework, we are constructing a trial mapping between FED and XML which, is used in our current generation tools. In particular, we defined a trial version of HLAML for Federation configuration file definition. We developed two converter programs which read HLAML and produce DIF or HTML file. HLAML is defined via XML DTD (included below) so that we were able to use free XML parsers. Current OWRTI, uses this file format for federation definition files. Our OMBuilder tool discussed in the following also supports the same file formats so that we can develop federations using this tool and import them easily into our HLA RTI framework.

```
<!--

This is a draft and experimental XML
document type definition for DIF files in
HLA Spec v1.3

HLAML v0.1

Date: August 13, 1998

-->


<!-- Simple data types -->

<!ELEMENT parameter   (#PCDATA)>

<!ELEMENT name        (#PCDATA)>

<!ELEMENT transport   (#PCDATA)>

<!ELEMENT order       (#PCDATA)>

<!ELEMENT sname       (#PCDATA)>

<!ELEMENT dimension   (#PCDATA)>

<!ELEMENT Federation  (#PCDATA)>

<!ELEMENT FEDversion  (#PCDATA)>


<!— Dimensions should contain at least one
dimension -->

<!ELEMENT Dimensions (dimension)+>


<!-- Each routing space has name and a set
of dimensions -->

<!ELEMENT space (sname,Dimensions?)>


<!-- Spaces should contain at least one
space -->

<!ELEMENT spaces (space)+>
```

```
<!-- Objects should contain at least one
'oclass' -->

<!ELEMENT objects (oclass)+>


<!-- Object Class contains attributes and
subclasses -->

<!ELEMENT oclass (name, attributeList?,
oclass*)>


<!-- attributeList wrappes the attributes of
object class -->

<!ELEMENT attributeList (attribute)+>


<!-- Each attribute has name, trasport type,
order. Space name is optional -->

<!ELEMENT attribute(name, transport, order,
sname?)>


<!-- All interactions are defined in this
element. -->

<!ELEMENT interactions (iclass)+>


<!-- Each interaction class has name,
transport type, order. It might have routing
space, a set of parameters and subclasses.

-->

<!ELEMENT iclass (name, transport, order,
sname?, parameter*, iclass*)>


<!-- Each federation definition file
containes name, version, object

     classes and interaction classes.
Definition of Routing Spaces are optional.

-->

<!ELEMENT FED (Federation, FEDversion,
spaces?, objects, interactions)>
```

Within our HPCMP FMS PET project at NPAC we are building visual HLA tool assembly framework [4] illustrated in Fig. 4 on top of the NPAC WebFlow [1][19][38] system (see Section 5.7).

WebFlow is a Web/Java based visual dataflow environment with the Web browser based computational graph editor and the runtime given by a mesh of interconnected Java Web Servers (Fig. 6), used to manage WebFlow middleware module wrappers and acting as proxies to the backend computational modules.

Through WebFlow linkage with HPC via modules / wrappers for Globus Metacomputing as well as support for database and visualization, our approach offers a natural platform for addressing HPC and HLA integration issues.

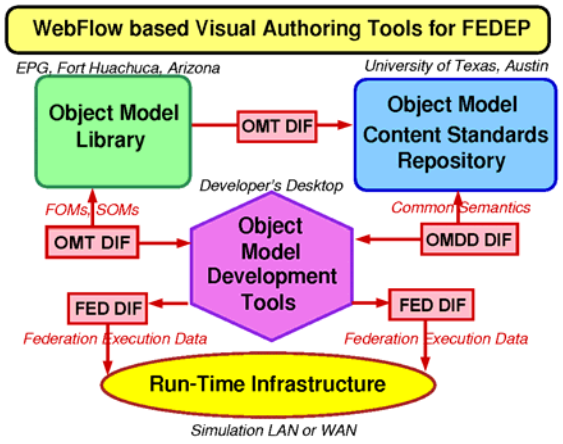

Figure 6.6: *WebFlow based representation of DMSO FEDEP: individual FEDEP tools are mapped on WebFlow modules and the DIF files are mapped on WebFlow communication channels.*

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.

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 (Figure 6.7). 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). A suitable visual icon in the WebFlow editor represents each Jager player (both human and robot).

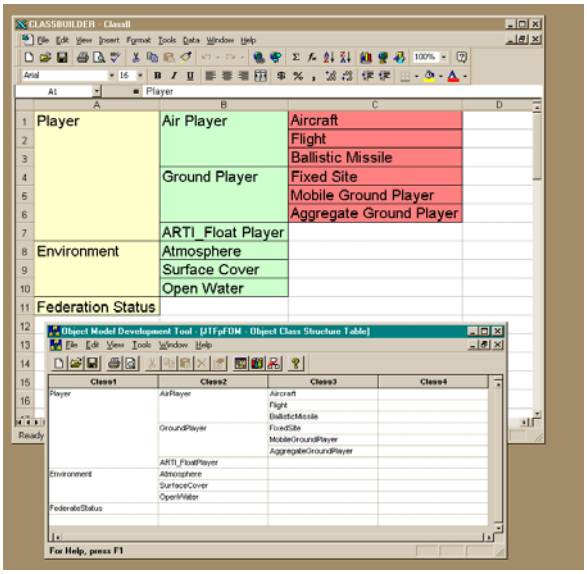

Figure 6.7: *OMDT by Aegis Corporation (front window) compared with our OMBuilder (back window): both tools have similar functionality and look-and-feel but OMBuilder is constructed via VBA scripting on top of Microsoft Excel.*

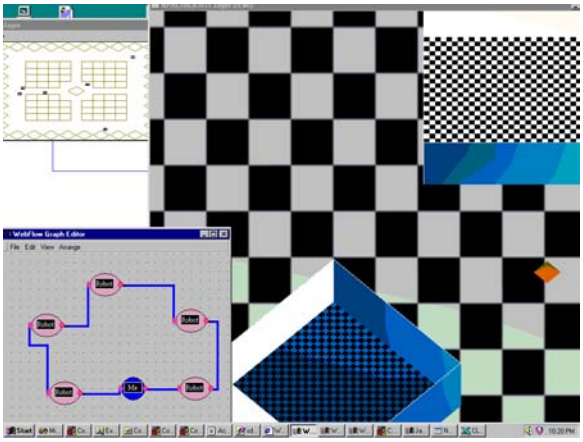

Figure 6.8: *Proof-of-the-concept visual authoring tools for Jager: WebFlow editor (lower left corner) allows to specify the configuration for the Jager game players. Both the original 2D (upper left corner) and our trial 3D viewer s are displayed.*

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 8). We also constructed a 3D viewer (figure 6.8) , operating in parallel with the standard 2D Jager viewer and used for experiments with parameter initialization via the extended OMBuilder editor (Figure 6.9).

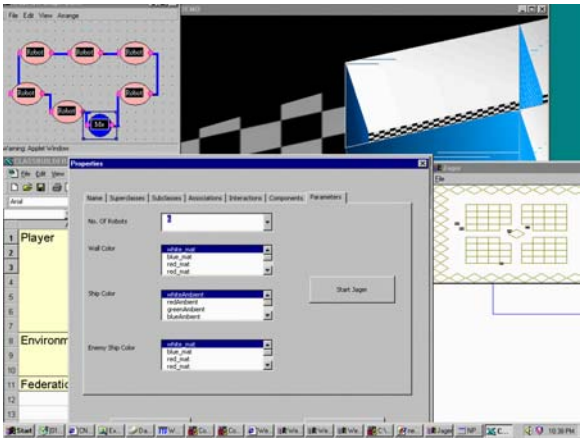

Figure 6.9: *Proof-of-the-concept visual authoring tools for Jager: a runtime extension of the OMBuilder tool allows the user to initialize, monitor and steer the simulation or rendering parameters.*

6.2.3 Parallel ports of selected M&S modules

In parallel with prototyping core WebHLA technologies described above, we are also analyzing selected large scale M&S 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 viewed as a virtual T&E tool to facilitate R&D 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.

CMS simulates mine and other minefield objects, and their interactions with vehicles. Major CMS objects include: a) Mine (landmines including several conventional or custom types); b) Component (a collection of mines of one type), c) Minefield (a collections of components enclosed within a terrain perimeter). The CMS user interface supports a number of functions for editing mines during their creation, such as specifying their location or type.

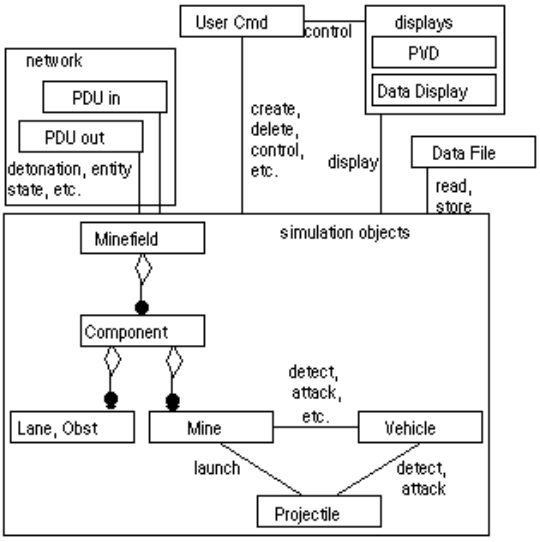

Figure 6.10: *Main components of the CMS system: network, user, display and simulation objects.*

Within CMS, a mine may interact with externally or internally simulated vehicles. Conceptually, the simulation objects operate largely independently. The control system allows these conceptually independent and autonomous objects to operate and interact without knowing whether other objects are on the same or other computer. In addition it controls the simulation timing. The control scheme uses messages to schedule and then execute processes that are conceptually independent. Due to well-defined granularity and independence of the simulation objects, parallel port of CMS to Origin2000 seems to be feasible even if the code is large and includes ModSAF libraries.

We are currently identifying the critical loops over messages, mines, components and minefields that need to be optimized or/and parallelized and we intend to address the actual parallel port implementation in Fall 98.

6.2.4 Database and data mining back-ends

Our database activities for the WebHLA backend include both research into the emergent transparent persistency models such as CORBA PSS, OLEDB, JDBC or XML/RDF [39], and the practical development of specific relational databases tailored for the individual components, projects and customers. For example, we are building SQL databases with the software documentation and PowerPoint presentation for the FMS Training Space and the distributed resources databases for the Metacomputing FMS and Commodity Clusters. We are also interacting with the Virtual Proving Ground team at ARL/ATC in Aberdeen where we currently start providing help with the data mining services for the T&E historical test data. In particular, we received recently from the VPG a MS Access database with a vehicle engineering and testing information and we analyzed it using some selected Web and data Mining technologies.

First, we made the VPG data accessible over the Web using Active Server pages for ease of access across workstations and network, and for use with future distributed datamining applications. An SQL query tool was written to run simple queries on this data and analyze their results.

In the second step, we decided to use a simple decision tree based classification algorithm, for our Data Mining experiments. We choose one of the attributes from the incident data table as our target class, dividing the values into two classes Major and Minor (Figure 6.11). We used a small subset of attributes that we thought would affect the classification process namely the subsystem, course condition and course type for our analysis. See5 from RuleQuest Research Ltd, was selected as the datamining tool for our data.

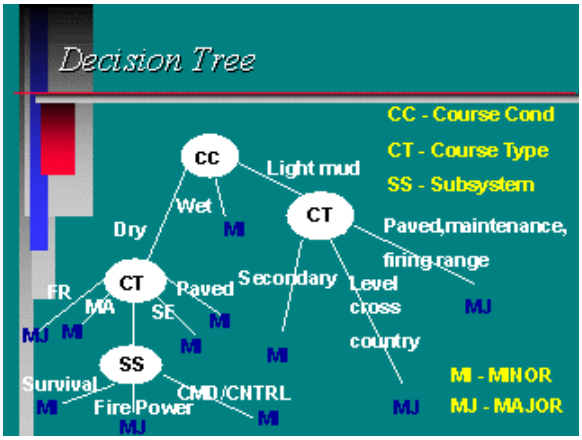

Figure 6.11: *Decision tree for the VPG test vehicle, constructed by the Data Mining algorithm described in this section,.*

Training data and test data were randomly selected with the help of the query tool. The tool was used to run the algorithm over the training data to generate a decision tree and ruleset with an error rate of 3.8%. On the test cases the error rate was found to be 12%, which indicates the abnormalities in the training set selection and the decision tree generation. We are in the process of refining this to get a lower error rate and to generate a better decision tree. We also calculated the information ratio for each of the

selected attributes and thus were able to select the course condition as the root node.

Data Mining experiments as described above allow us to become familiar with the large datasets and the high performance computational challenges of T&E. In the longer run, we view VPG as a promising testbed for WebHLA based virtual prototyping environments.

6.2.5 Realtime multiplayer gaming front-ends

In our Pragmatic Object Web approach, we integrate CORBA, Java, COM and WOM based distributed object technologies. We view CORBA and Java as most adequate for the middleware and backend, whereas COM as the leading candidate for interactive front-ends due to the Microsoft dominance on the desktop market.

Of particular interest for the M&S community seems to be the COM package called DirectX which offers multimedia API for developing powerful graphics, sound and network play applications, based on a consistent interface to devices across different hardware platforms. Though DirectX was primarily developed for PC gaming, it makes an excellent candidate for building simulations for the PC platform, thus bringing HLA closer to the PC environment. DirectX uses fast, low level libraries, which provide a very good control of multimedia hardware - a feature that can be used to build smooth and responsive simulations.

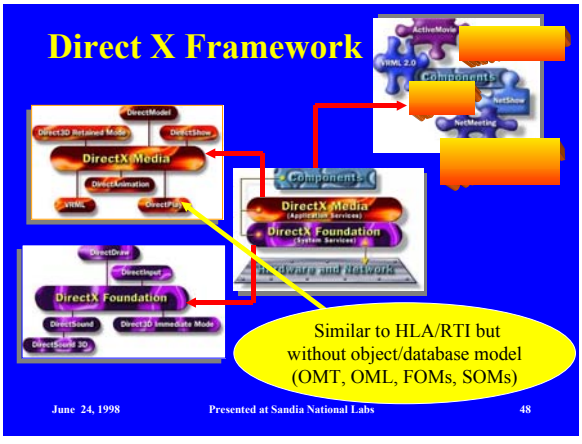

Figure 6.12: *DirectX Framework: includes low level Foundation objects such as DirectSound or Direct3D, Media objects such as DirectPlay, DirectShow or DirectAnimation, and high level components such as NetMeeting for collaboration or ActiveMovie for video streaming.*

DirectX comes as a set of APIs (Figure 6.12), each of which cater to a different aspects of an application being developed - DirectDraw for graphics, DirectSound for adding sound, DirectPlay for networked gaming (Figure 6.13) etc. Of these, the DirectPlay API is of interest to us while discussing HLA/RTI, because of it's similarities with the RTI

Figure 6.13: *Sample multiplayer games (Duel, Paper Planes) based on the DirectX / DirectPlay engine.*

A DirectPlay session is analogous to federation execution in HLA, with each application (federate) interacting via an abstract communication medium. Both DirectPlay and RTI provide interfaces and methods to manage sessions/federations. The DirectPlay player management functions are similar to the RTI object management functions for creating and destroying players within a session. The notable difference is that DirectPlay does not provide time synchronization features, and only a simplistic version of the data management. On the other hand, DirectPlay has certain novel features, which are not available in the RTI. A good example is the Lobby object which offers the functionality of a real world lobby where players can meet, interact and find the right partner to play against.

Due to the design proximity, one could naturally create an HLA simulation using DirectX, while still using RTI as a communication medium. The simulation would thus conform to the HLA/RTI standards and perform well in the PC environment.

We constructed recently a proof-of-the-concept demonstration of such a linkage between Microsoft DirectX and Object Web RTI on a real-time basis using a modified version of the popular DirectX game - *Donuts* (Figure 6.14) as the front end. The objective here, apart from presenting OWRTI, was to

demonstrate the ability to bring together the high-end RTI and commodity technologies like DirectX to build HLA compliant simulations. The whole set-up of the demo included a wide range of technologies like CORBA, Java, C++ and DirectX/COM.

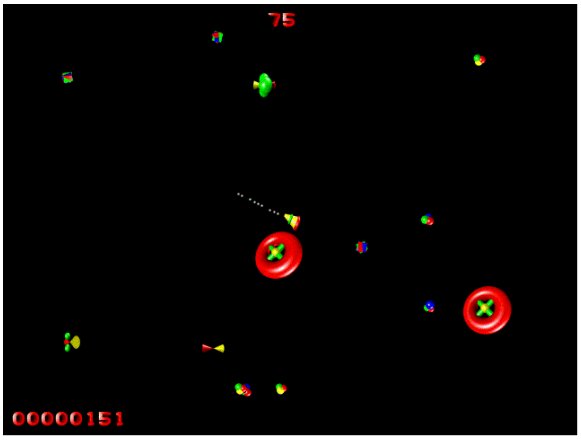

Figure 6.14: *A Sample screen from the original Space Donuts demo by Microsoft.*

The entire front end of the demo was built using Microsoft's DirectX API for gaming & multimedia. The basic structure of the sample game *Donuts* (provided with the Microsoft DirectX SDK) was used and suitably modified to incorporate an RTI layer for communication between two remote games (Figure 6.15).

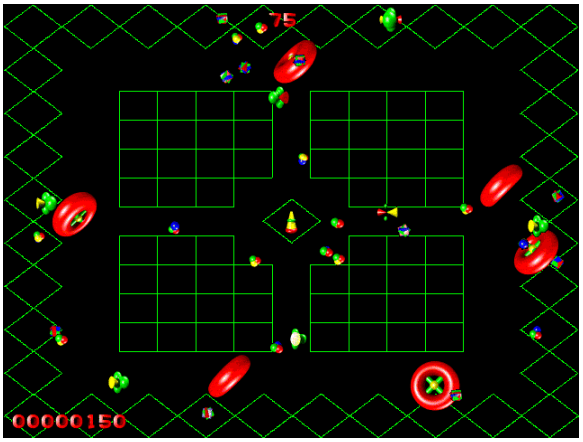

Figure 6.15: *Sample screen from multi-player Jager Donuts demo using modified Donuts frrot-end (to conform to the Jager game geometry) and the Object Web RTI based multi-user communication.*

The game involves shooting down *donuts* using a gun/ship, which can be manipulated around the screen by the user. All the animation and graphic effects of the game are accomplished using DirectDraw, which is a major component of the DirectX API. When the game starts up, it connects to a RTI server and registers itself as a federate. It then receives information about other federates connected to the same server.

The *remote donuts* game, as a first and simple step, sends and responds only to RTI interactions. A game (federate) responds to an interaction from another federate by putting a donut on the screen. A federate can send an interaction to other federates on the command of the user. The RTI server forwards the interaction to all other federates in the federation, each of which would respond by putting a donut on their screens.

In the demo, we provided an automatic interaction generator. This is another federate joining to the Federation and producing only interactions. Its parameters are controlled through a Web interface. A servlet is responsible for getting number of interactions to be produced, the time interval between interactions and message order type (receive/timestamp). Then it forks a separate federate which is written in Java and interacts with the RTI. As a result, user automatically starts receiving messages from this federate.

With the DirectX platform and the associated high level Web tools such as Chromeffects, Microsoft enters the computer gaming market. Hence, once can expect the coming explosion of Web based networked games for Windows.

Figure 6.16: *Sample screen from the DirectX based professional game Art of Flying.*

Figure 6.16 illustrates a state-of-the-art DirectX based real-time commodity graphics scene in a professional game Art of Flying by Blue Moon Interactive. Our DirectX-OWRTI bridge discussed above will allow us to reuse these entertainment front-ends for the military training purposes by coupling them with the HLA compliant simulation backends and delivering the training in the Web based interactive mode.

6.3 Emergent WebHLA Applications

In parallel with developing focused WebHLA components discussed in Section 6.2, we also start identifying some promising WebHLA application domains and we initiate the design and prototyping activities. In this Chapter, we conclude the paper with a brief overview of our current activities in the following emergent WebHLA application areas: Distance Training, Resource Management for Metacomputing and/or Commodity Clusters, and Simulation Based Acquisition.

Figure 6.17: *Initial pages when entering the FMS Training Space. Home Page (upper left) offers external links to several DoD Labs associated with FMS and internal links to the training content. The latter access is password protected (upper right). After authorization, user gets customzed home page (e.g. lower left for database administrator role) which in turn offers access to courses for individual technologies (current list of courses under development includes SPEEDES, CMS, ModSAF, HLA/RTI, WebHLA).*

6.3.1 Distance Training

We are developing an extensive electronic training space for the FMS users and developers (Figure 6.17), using our Object Web RTI as a core technology framework for the interactive collaborative distance training.

In the HLA lingo, each participant of a training session becomes a federate, and so are their trainers/mentors as well as the particular M&S systems or technology modules selected as the current training target. The training session itself becomes a federation which follows the HLA rules for joining, participating, sharing the information, managing assets such as time, space or objects etc.

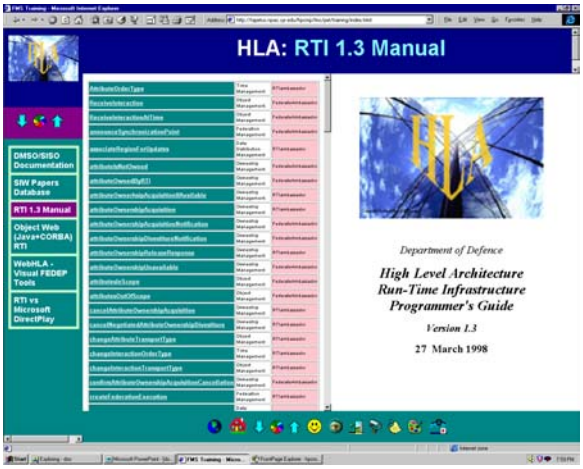

Figure 6.18: *Sample screen from the FMS Training Space: Manual pages for the DMSO RTI 1,.3 C++ implementation, extracted from the Microsoft World documents published ion the DMSO Web site.*

Such training federations can be naturally made World-Wide distributed within our WebHLA and given real-time interactive Web /Commodity based interfaces via suitable JWORB protocols (HTTP, IIOP, DCE RPC etc.).

Our current suite of FMS training materials in the development pipeline includes: a) HLA/RTI itself, see Figure 6.8 (including Web / distributed access to DMSO HLA demos); b) our Object Web RTI implementation; c) SPEEDES (Synchronous Parallel Environment for Emulation and Discrete-Event Simulation) training, see Figure 6.19; d) CMS training (Figure 6.21) ; e) ModSAF training (as required for CMS simulation), see Figure 6.20.

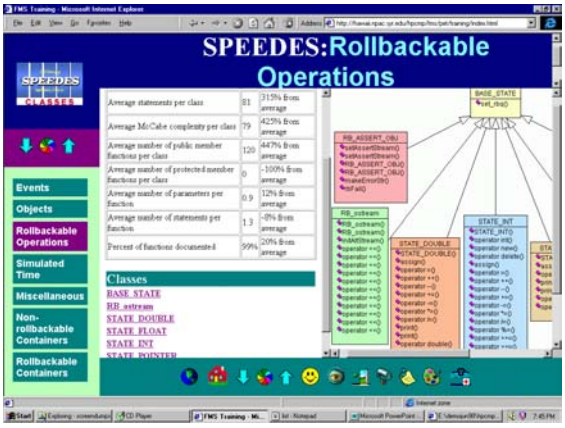

Figure 6.19: *Sample screen from the FMS Training Space: Rollbackable operations in SPEEDES and the associated UML class diagram.*

We are also interacting with FMS CHSSI projects and we plan to include other systems under development such as E-ModSAF, IMPORT, TEMPO / Thema into our evolving training suite.

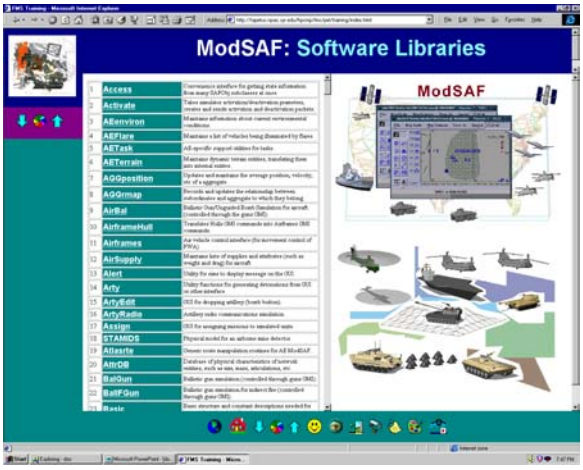

Figure 6.20: *Sample screen from the FMS Training Space:A list of 500+ ModSAF libraries with hyperlinks to the on-line documentation pages.*

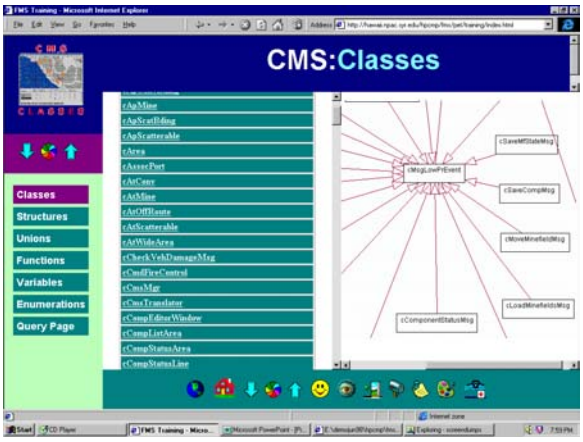

Figure 6.21: *Sample screen from the FMS Training Space: A list of classes in the CMS system and the associated UML Class Diagram.*

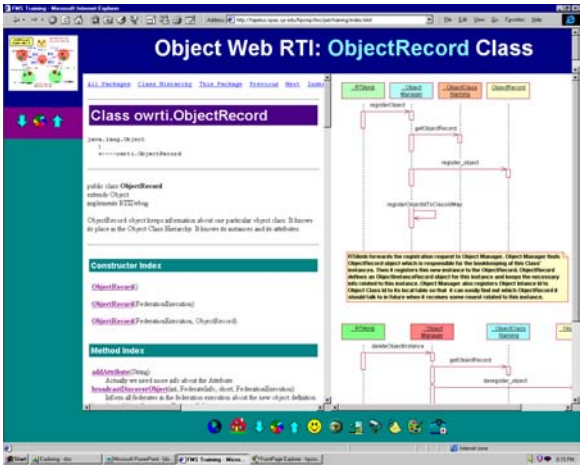

Figure 6.22: *Sample screen from the FMS Training Space: a manual page for the OWRTI Java class and the associated UML Sequence Diagram illustrating a typical use of this class in an application scenario.*

6.3.2 Metacomomputing FMS

CMS system discussed in Section 6.2.3, when viewed as an HLA federation, decomposes naturally into the minefield (Figure 6.23) and vehicle (tanks, contermines etc.) federates (Figure 6.24). 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 Section 6.2.1.

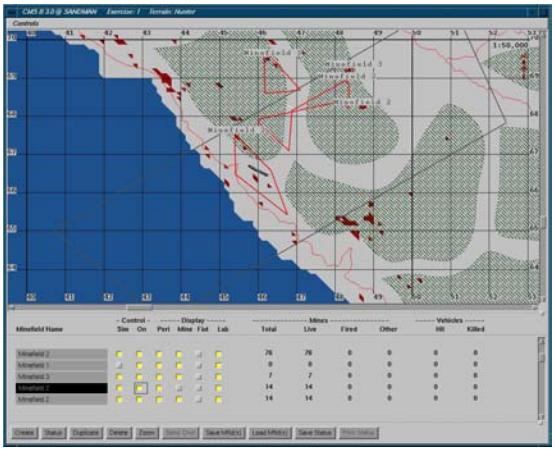

Figure 6.23: *CMS Minefield Visual Editor Front-End*

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.

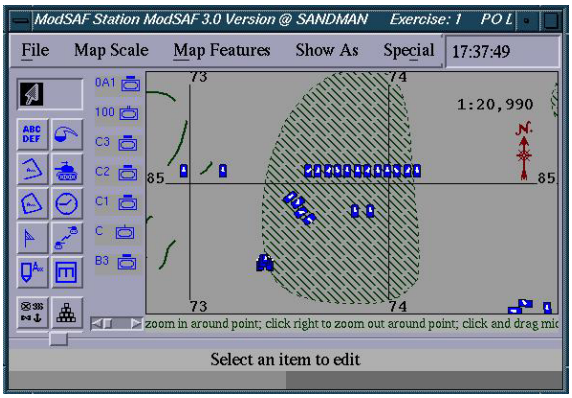

Figure 6.24: *ModSAF Scenario Visual Editor Front-End*

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. 18 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).

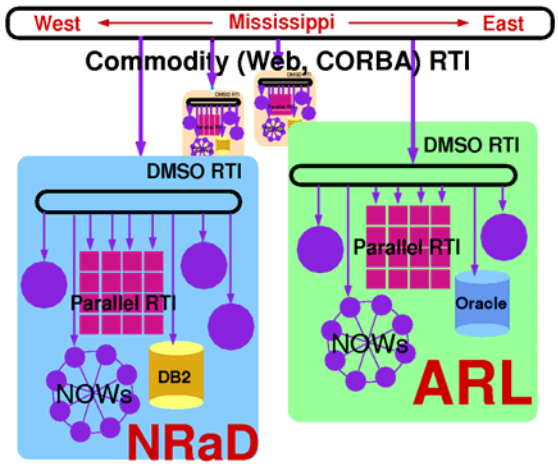

Figure 6.25: *Illustration of the interplay between DMSO RTI (running on Intranets), Parallel RTI (running on HPC facilities) and Commodity (such as Object Web) RTI. The latter is running in the Web / Internet Domain and connects geographically distributed MSRCs and DCs.*

6.3.4 High Performance RTI

We are currently starting new FMS CHSSI project jointly with SPAWAR, San Diego and Metron, Inc. that will build HPC RTI on top of SPEEDES simulation kernel. The core SPEEDES related development effort will be done by Metron, whereas our role is to assure full HLA compliance, based on our experience with implementing Object Web RTI. We will also extend our HLA and SPEEDES training modules discussed above towards HPC RTI training that will teach both systems by illustrating the use of SPEEDES to implement RTI and to sustain high performance for a broad range of simulations.

SPEEDES (The Synchronous Parallel Environment for Emulation and Discrete-Event Simulation) is a framework to develop a high-performance object-oriented simulations. SPEEDES supports both conservative and optimistic(i.e. rollback-based) synchronization strategies which includes Fixed Time Buckets(FTB), Breathing Time Buckets(BTB), Time Warp(TW), and Breathing Time Warp(BTW). With these rich set of synhronization strategies, SPEEDES allows hybrid synchronization techniques to be used at the same time so that external human, software, and/or hardware interactions can be handled in a unified framework. SPEEDES supports transparent rollback/rollforward mechanism through the use of incremental state saving with a rollback queue maintained for each event. SPEEDES communication

library contains a set of portable primitives so that framework can run on various hardware platforms(such as networks of workstations or PCs, shared-memory multiprocessor machines, and/or massively-parallel distributed-memory high-performance computers) with scalable manner. Developer in SPEEDES framework supplies the implementation for possible events in the simulation, simulation objects and managers for these simulation objects. All objects are derived from well defined objects in SPEEDES and with the help of virtual method calls in C++ developer never sees the communication and synchronization protocols going on in the runtime environment. This abstraction provides an efficient framework for developers as developer does not need to worry about all the mechanisms hidden from them while focusing on the simulation problem at hand.

6.3.5 IMPORT / PANDA Training

We are also participating in a new JSIMS/Maritime initiative PANDA (Parallel Advanced Development Architecture), aimed at building visual authoring environment for parallel simulations. Main components of this planned system include: Osim – a UML-like graph based visual front-end by OriginalSim, Inc., SPEEEDES simulation backend and IMPORT programming language as middleware. We intend to develop WEbHLA based IMPORT training and to explore synergies between WebFlow and PANDA authoring concepts.

IMPORT (Integrated Modular Persistent Objects and Relations Technology) is a fifth generation language (5GL) tailored to develop large scale, complex simulations using object-oriented technology. IMPORT tools currently generates: a) QuickThreads Simulation Runtime (QTRS) which runs on single CPU low cost PCs and workstations; b) SPEEDES codes; and c) TEMPO/THEMA codes

IMPORT supports all object-oriented modeling concepts such as inheritance, encapsulation and polymorphism. Modularizan is provided through module structures with well-defined interfaces so that further information hiding and data abstraction can be done. IMPORT supports a process-based simulation by providing a model of object-level concurrency. Each object owns a thread of execution and maintains its own schedule so that it can control the task priorities and executes an appropriate method. It provides language-level constructs to model the passing of simulation time, launch concurrent objects, synchronize the activities of sets of objects, allow communication between concurrent objects, interrupt objects, and prioritize an object's tasks. It utilizes an object oriented database at the heart of its data-management facilities. IMPORT programs may interact directly with several high-performance commercial and public-domain object-oriented database systems through its Generic Object-Oriented Database interface

6.3.6 Commodity Cluster Management

The emergent Pragmatic Object Web, defined as a merger of CORBA, Java, COM and WOM, needs 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 WebHLA, we view HLA/RTI as a potential candidate for such a uniform high level service framework [X]. In fact, the WebHLA application domains discussed in this Chapter can be viewed as various attempts at extending RTI beyond the original M&S domain towards: collaborative training (Section 6.3.1), metacomputing resource management (Section 6.3.2), commodity cluster management (this section), and finally all assets management framework (next section).

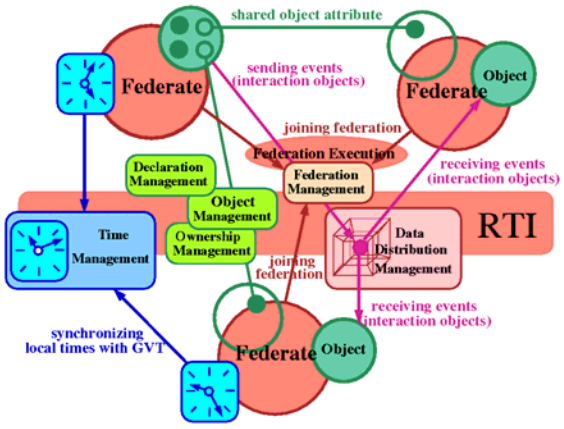

Figure 6.26: *Distributed object based architecture of DMSO RTI – originally constructed for the M&S domain but naturally extensible for other distributed computing management services such as cluster, metacomputing or collaboration management discussed in the text.*

Indeed, as illustrated in Figure 6.26, 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.

6.3.7 Simulation Based Acquisition

Figure 6.27 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 M&S tools and modules, wrapped as commodity (CORBA or COM) components and accessible via interactive Web browser front-ends. Such environments, currently operated only by large industry such as Bowing, become affordable within the current technology convergence process as envisioned in Figure 6.1 and quantified in our WebHLA integration program

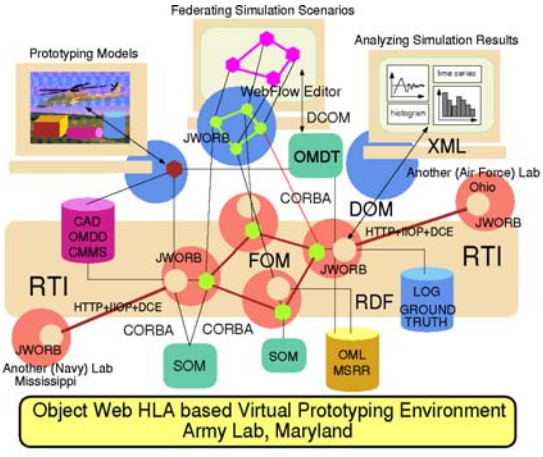

Figure 6.27: *Overall architecture of a planned WebHLA based Virtual Prototyping Environment for Simulation Based Acquisition: Object Web RTI acts as a universal software bus, managing a spectrum of resources for simulation, clustering, collaboration, training, data mining, data warehousing etc.*

Indeed, the challenge of SBA is to successfully integrate M&S, T&E, HPC, 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 discussed in the paper and being now prototyped in a coordinated, Pragmatic Object Web based development and integration framework.

6.4 Towards POW VM

Based on lessons learned so far with our POW prototype servers (JWORB) and services (OWRTI, JDCE), we are now designing an more systematic Virtual Machine (VM) support for POW based HPcc. By POW VM, we mean a mesh of JWORB servers running on various platforms and supported with a suite of services that facilitate formation of distributed computational surfaces, collaboratory spaces or federations of simulation modules. JINI for Java, Millenium for COM or HLA for DoD M&S are examples of such environments, discussed so far. Here we address the corresponding POW architecture for the next generation ubiquitous distributed computing.

We assume commodity (PC) clusters as a typical HPcc architecture, operating either in a pure commodity distributed mode or acting as a gateway for dedicated MPP systems.

The challenge of POW VM is to integrate efficiently the commodity software coming from companies such as Microsoft and gradually extending from the desktop towards distributed computing domain, with the platform-indepdendent middleware (typically given by Java servers) and platform-specific specialized multi-language (perhaps CORBA wrapped) back-ends.

Our POW VM is still under construction. Here we summarize only some initial design guidelines and our overall strategy. We start with the overview of the commodity clusters, followed by an example of binding one of the Microsoft front-end technologies – DirectX for desktop multimedia – with RTI we view as a critical middleware communication service in the POW framework. This example illustrates how COM, CORBA, Java and HLA can be efficiently integrated by exploring the design synergies between the APIs in various competing models. Next, we illustrate how we can fill the gaps in the current HLA/RTI design by using XML as the scripting langauge for HLA attributes and parameters. This leads us in a natural way towards the XML agents, followed by the summary of the overall architecture of the POW VM.

6.4.1 Commodity Clusters

Important new commercial applications, such as internetworking and the WWW, data warehousing and data mining, and enterprise-wide management systems, are driving an explosion in demand for high-end server systems. In the past, commercial users would have looked to mainframes and super-minicomputers - expensive to acquire, maintain, and upgrade - to satisfy these new requirements. But today, servers based on mass-produced personal computer technologies and offering major price / performance advantages over traditional systems are becoming available to meet the needs of the lower end of these new markets.

Clustering commodity PC servers into highly available and adaptable, high-performance computing meshes becomes a viable alternative to the minicomputer or mainframe solutions. Continual increases in the raw power of PC processors and I/O subsystems, combined with the advent of Microsoft's Windows NT operating system, provide the basic ingredients for building a cluster of PC servers for an enterprise database or other mission critical applications. These types of clusters are commonly referred to as a System Area Network (SAN).

In broad terms, commodity clusters are groups of interconnected servers that work as a single system to provide high-speed, reliable, and scalable service. Until recently, only very expensive, proprietary systems could deliver the levels of speed, reliability, and scalability required for high-performance or enterprise computing. With clustering, less expensive, industry-standard systems now have this capability. Cluster configurations are used to address availability, manageability, and scalability. By Availability, we mean that when a system or application in the cluster fails, the cluster software responds by restarting the failed application or dispersing the work from the failed system to the remaining systems in the cluster. Manageability allows administrators to use a graphical console to move applications and data within the cluster to different servers for load balancing purposes. Finally, we probe Scalability when the overall load for a cluster-aware application exceeds the capabilities of the systems in the cluster and the additional systems can be added transparently to the cluster configuration.

Clustering can take many forms. At the low end, a cluster may be nothing more than a set of standard desktop PCs interconnected by an Ethernet. At the high end, the hardware structure may consist of high performance SMP systems interconnected via a high-performance communications and I/O bus. A client interacts with a cluster as though it was a single high-performance, highly reliable server. Additional systems can be added to the clusters as needed to process increasingly complex or numerous requests from clients.

During the evolution of the cluster technology, reliable messaging between clustered servers will become increasingly critical. The VIA initiative is synergistic with current clustering strategies. VIArchitecture was developed to minimize message processing delays and to allow more efficient communication within system area networks. VIArchitecture is the messaging interface for applications that reside on servers connected in a SAN. The concept behind VIA has its roots in the traditional messaging between applications and a computer's Network Interface Controllers. Computer applications operate as if they have unlimited memory. In reality, the OS gives and takes the actual memory away from applications as it is needed to run other applications. Traditionally, if an application wanted to send messages to the NIC using the physical memory, the request had to go through the kernel, which caused processing delays. With VIA, the applicationcan use its virtual memory addresses to speak directly to the SAN NIC without going through the kernel. This will greatly reduce the message latency. Although applications bypass the operating system by sending messages from their virtual addresses, the OS continues to provide security and messaging setup for the applications.

As mentioned previously, VIA was developed by Microsoft, Intel and Compaq. Microsoft is in fact ready to address the full spectrum of new software needs for the cluster computing niche with a familiy of server products. This emergent Microsoft solution includes the combination of COM or/and DCOM based software services such as: a) Microsoft Transaction Server (code name Viper) for component management; b) Microsoft Message Queue Server (code name Falcon) for asynchronous messaging; c) Microsoft Cluster Server (code name Wolfpack) for scalable and highly available NT clusters; and d) Microsoft BackOffice Server for core enterprise services (including Exchange Server for Email, NetMeeting and Internet Location Server for

Collaboration, SQL Server for Relational Databases etc.).

We are currently starting a new project with Sandia National Laboratory in which we will evaluate these servers, compare them with alternative commodity solutions (Linux, PC Solaris) and integrate with our JWORB middlware via COM/CORBA bridges. This way, we will construct a high performance two-layer middleware with parallel I/O support between the COM and CORBA software bus architectures. Such framework will enable smooth integration between new Microsoft front-end technologies for real-time multimedia and network multi-player gaming such as DirectX / DirectPlay discussed in the next section and the CORBA/HLA based simulation modules in the backend.

6.4.2 DirectPlay meets RTI

DirectX – Overview A vital player in transforming Microsoft's Windows Operating System into a major multimedia platform, and in capturing the gaming market, DirectX is a set of APIs for developing powerful graphics, sound and network play applications. The biggest incentive that DirectX offers developers is a consistent interface to devices across different hardware platforms. It achieves this by abstracting away the underlying hardware by using the HAL (Hardware Abstraction Layer) and HEL (Hardware Emulation Layer), both of which are integral parts of the DirectX architecture. This allows for using hardware functionality that might not even be available on the system. DirectX uses fast, low level libraries, which access multimedia hardware in a device independent manner. It thus helps developers get rid of a lot of constraints that influence game design. DirectX, as of version 5, has six categories of APIs to support different device categories and functionality: DirectDraw, Direct3D, DirectSound, DirectPlay, DirectInput and DirectSetup.

Figure 6.12 collects all elements of the DirectX Framework and it exposes its hierarchical/layered organization. Starting from the DirectX Foundation or System Services (including all core multimedia services listed above), followed by DirecrtX Media or Application Services (which includes DirectPlay), and finally followed by a set of high level components such as NetMeeting for collaboration, ActiveMovie for video streaming and so on.

Of these elements, the DirectPlay API is of the most interest to us in our discussion of HLA/RTI. The next section will discuss DirectPlay in a little more detail.

DirectPlay DirectPlay is a component of DirectX that provides networking and communication services in a transport independent manner. DirectPlay provides a consistent interface for the developer irrespective of whether the transport mechanism is TCP/IP, IPX or modem. This is achieved using 'service providers', which are layered between DirectPlay and the network. A 'service provider' would exist for each transport medium. Third party developers can create new ones.

A typical DirectPlay gaming scenario would consist of one or more 'sessions'. DirectPlay provides means to create new sessions, list (enumerate) the existing ones and to connect to an existing session. Once a player application connects to a session, it can interact with other player applications in the session. DirectPlay provides methods to move game data among the various participants. Another interesting feature of DirectPlay is it's Lobby object. The Lobby object has the functionality of a real world lobby where players can meet, interact and find the right partner to play against.

Since the look and feel of the DirectPlay interfaces are similar to those of the DirectX components, with a good application design, DirectPlay can be gracefully merged with other DirectX components to create exciting and stimulating multi-player games.

The DirectPlay API The DirectPlay API provides the following services for a networked gaming environment : (The functions described below are just used to illustrate the DirectPlay services and are not an exhaustive set of the DirectPlay API functions!)

- *Session Management Functions* A Direct Play session consists of several applications communicating with each other over the network. The session management functions are used to initiate, control and terminate these interactions.

    - EnumSessions(...) : Lists/Enumerates all the sessions in progress on the network. –
    - Open(...) : Creates a new session or connects to an existing one.
    - Close(...) : Disconnects from a session.

- GetSessionDesc(...) : Obtain the session's current properties.

- *Player Management Functions* The player management funtions are used to manage the players in a session. In addition to creating and destroying players, an application can enumerate the players or retrieve a player's communication capabilities.

  - EnumPlayers(...) : Lists/Enumerates all the players in the session.
  - CreatePlayer(...) : Create a new player.
  - DestroyPlayer(...) : Delete a player.
  - GetPlayerCaps(...) : Get a player's properties (connection speed etc.)

- *Group Management Functions* The group management methods allow applications to create groups of players in a session. Messages can be sent to a group, rather than to one player at a time if the service provider supports multicasting. This is useful to conserve communication-channel bandwidth.

  - EnumGroups(...) : Lists/Enumerates all the groups in the session.

  - CreateGroup(...) : Create a new group.
  - DestroyGroup(...) : Delete a group.
  - EnumGroupPlayers(...) : Lists/Enumerates all the players in a group.
  - AddPlayerToGroup(...) : Adds a new player to a group.
  - DeletePlayerFromGroup(...) : Deletes a player from a group.

- *Message Management Functions* Message management functions are used to route messages among players. Most messages can be defined specific to the application.

  - Send(...) : Sends a message to a player, a group, or all the players in the session. –
  - Receive(...) : Receives a message from the message queue.
  - GetMessageCount(...) : Returns the number of messages wating in the queue.

- *Data Management Functions* DirectPlay lets applications associate data with players and groups. Applications can store two types of information - local and remote. Local data is meant for and is only available to the application that sets it. Remote data is available to all the applications in the session and is used like a shared memory.

  - SetPlayerData(...) : Sets data for a player.
  - GetPlayerData(...) : Gets data for a player.
  - SetGroupData(...) : Sets data for a group.
  - GetGroupData(...) : Gets data for a group.

DirectPlay is based on COM - the Component Object Model and is made up of objects and interfaces based on COM just like the rest of DirectX. All the functions described above are in fact methods of the 'IDirectPlay2' interface of DirectPlay. There are also other DirectPlay library functions used to enumerate the service providers and to create the DirectPlay object itself, before using it's interface.

DirectPlay vs HLA RTI DirectPlay draws comparisons with the HLA RTI due to the common approach that both take in providing an abstract layer to the network. Although DirectPlay was designed with features specific to gaming, there are several similarities between the HLA RTI and DirectPlay.

A DirectPlay session is analogous to federation execution in HLA, with each application (federate) interacting with one another through a communication medium. Both DirectPlay and RTI provide interfaces and methods to manage sessions/federations. The DirectPlay player management functions are similar to the RTI object management functions for creating and destroying players within a session. The notable difference is that DirectPlay does not provide time synchronization features, and there is no concept of 'declaring' an object's attributes, 'interests' and 'intentions' like in the RTI declaration management service. One of the reasons for this is that a higher framework model doesn't govern DirectPlay as the HLA governs RTI. Another reason is that commercial gaming never required these features. Game developers could decide on their own set of protocols specific to their application.

6.4.3 XML Scripting and Agents

Synergy between high level communication models of DirectPlay and RTI discussed above is in fact symptomatic for the current generation of competing but in fact increasingly overlapping and redundant software models. Two other similar event models based on publish/subscribe techniques are offered by JavaBeans and COM+ and were already discussed in

previous Sections (X and Y, respectively). We can therefore expect that the currently competing event-driven schemes will gradually converge on the Pragmatic Object Web.

However, it is worthwhile to note here that none of the current event frameworks listed above comes with a specific event format. Unlike in the previous generation strictly specified event models such as XEvents in X Windows or PDUs in DIS, today's events often revert back to the lower level formats, with the transmitted information being custom encoded as textual strings or numerical buffers at the sender side and then custom decoded at the receiver side. In a sense, the whole concept of remote object was aiming at replacing such custom codecs by more organized and standardized object proxies and method stubs/skeletons. However, while objects communicate by methods, modern components usually prefer to exchange events in a minimally committant publish/subscribe mode, mediated by low level data buffers, reserving thereby the maximal flexibility for the plug-and-play multi-platform, multi-vendor interoperability.

Here is where we see XML to fit perfectly well and to provide universal scripting capabilities complementary to the converging distributed object technologies. In retrospect, having reviewed several distributed object and component approaches, we can summarize the interoperability lessons we learned so far in terms of the following subsequent abstraction layers.

Network Protocols At the network protocol level, we are facining a few competing candidates such as: IIOP of CORBA, DCE RPC of DCOM, JRMP of RMI, and HTTP of the Web. None intends to go away in the near future, hence our concept of multi-protocol JWORB as an ecomony solution for POW middleware node.

Distributed Objects At the distributed object level, the differences between Java, CORBA and COM are often hidden at the user/programmatic level and/or they have tendency to disappear as various models first collide and then converge. Hence, distributed objects are becoming a viable programming technology but they are most adequate for reasonably well understood computationally demanding application domains where the time investment in building the object models is granted by the customer needs.

Components, Events and Scripting However, dynamic distributed computing domains such as gaming or electronic commerce require more flexible, adaptable and modularizable approaches based on plug-and-play components, events and scripted communications. To put it differently, it is possible to merge COM, CORBA and Java objects in a simple large scale application, but it is usually easier to package both domains as reusable coarse components, linked via adaptable XML scripts.

XML Agents The use of XML as interoperability language for distributed event frameworks seems to naturally generalize and extend towards the next, yet higher level of agent based computing. Indeed, XML is naturally readable and writable both by humans and machines (unlike say human-oriented HTML or traditional machine-compiler-oriented Java or JavaScript). Next, any XML event thrown by one Internet component, can be easily caught and further handled by another component or agent, be it a database, middleware filter or user level browser. This facilitates the explosion of middleware filters that will effectively result in a fat multi-tier middleware as in Fig 1.1.

6.4.4 POW VM Architecture

We summarize here our POW VM architecture as we understand it by the end of summer 98. As demonstrated during the analysis of various technology domains addressed in this document, we live in an interesting and difficult time of aggressive competition between major computer vendors trying to grasp control over some global computing paradigms. Our bet is that no single model will win in the near term but we also need a practical approach to proceed and succeed by turning the ongoing software wars to our advantage. Hence our concept of the Pragmatic Object Web and its trial prototype implementation in terms of a mesh of JWORB servers.

The overall concept of JWORB as a single multi-protocol server that is capable to handle each of the object models that comprise POW was already discussed in Section 5.1. Here we summarize POW VM, defined operationally as a node of a (potentially world-wide) distributed environment that follows the POW methodology and hence it enables interoperability between the emergent family of global solutions for Web Computing such as Millenium, JINI, WebBroker etc. A natural POW VM architecture is therefore given by JWORB,

dressed by a set of core services that facilitate such interoperability.

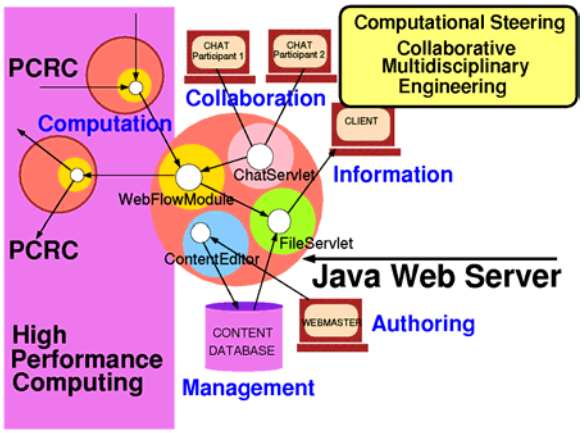

Figure 6.28: *Early WebVM architecture, based on the Jigsaw Java Web Server from W3C.*

Our earlier effort towards specifying such set of services, pursued in the context of 100% pure Java and based on servlets managed by pure Java Web Servers (such as W3C Jigsaw or Javasoft Jeeves) is illustrated in Figure 6.28. Using Jigsaw Resources and then Jeeves Servlets we prototyped what we considered a minimal set of such entities, sufficient to handle Web content serving and authoring, database management, synchronous and asynchronous collaboration, and database management. Some 3 years ago, several of these servlets were still following custom design. Today, in a broader POW framework we can inherit, extend and mix-and-match several standard services from CORBA, COM, Java and WOM when buidling the core POW VM. One example of such a constructive mixture is the use of CORBA services and OMA software engineering practices when building advanced HTTP-NG services.

Also, JWORB based POW VM architecture neturally enables experimentation with colliding, merging and converging standards. For example, as seen from the discussion of XML RPC in Section X, efforts in this area are based on not-yet-standardized extensions of the HTTP protocol. Hence the only way right now to become an early player in this new area of XML based Web Computing is to have control over and be able to experiment with a prototype Web server implementation code. New concepts in the area of merging XML with CORBA and COM RPCs such as WIDL or WebBroker, discussed in Section X, can be also naturally tested and integrated in the JWORB framework. (Indeed, without JWORB one would need the installation and maintenance support for four different network servers – CORBA, COM, Java and HTTP – typically coming from four vendors with the usual vendor-specific idiosynchracies and major software redundancy across the individual server technologies).

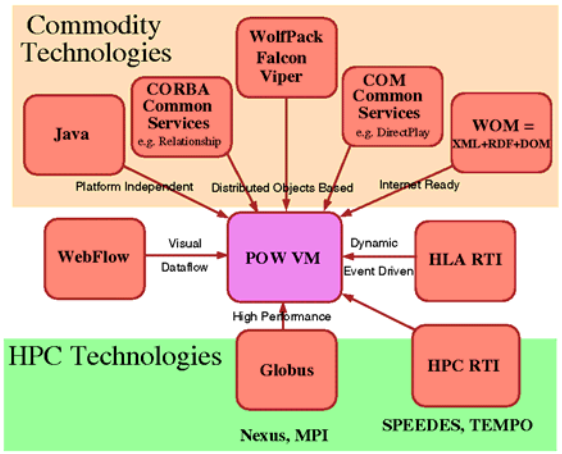

Figure 6.29: *Current input / enabling technologies towards the POW VM under construction.*

Hence we build our POW VM node as JWORB on top of JVM with some core CORBA and/or COM services, extended by a code XML toolset (including parsers, DOM, XSL processors etc.). In the component and event model domain, we are obviously supporting JavaBeans and COM+ components – as POW stakeholders – but while waiting for Sun and Microsoft to decide who is smarter in court, we also intend to support HLA componentware, represented by the Federation Object Model, and the associated event (or interaction) model given by the Run-Time Infrastructure. Since FOM object attributes and interaction parameters are typeless and require custom encoding/decoding, we are extending HLA towards WebHLA as discussed earlier in this Section by mapping FED to DTD formats and by using XML scripting for HLA communication. This way, we end up with an elegant overall model for POW VM which:

- augments HLA/RTI by the universal XML scripting based on new Web/Commodity technologies;
- augments XML based Web Computing by a powerful event-driven mechanism of HLA/RTI;

- complements both by integrating Web with Distributed Object Technologies of CORBA, Java and COM.

Diagram in Figure 6.29 summarized input used in the POW VM design, coming from various computational domains. As discussed previously in Section X, High Performance is provided by suitable encapsulated modules such as Globus for general purpose Metacomputing , SPEEDES for high performance event-driven simulations etc.

6.5 Summary

We presented here WebHLA - an interactive 3-tier programming and training environment based on:

- DMSO HLA architecture and our JWORB based Object Web RTI implementation in the middleware,
- commodity front-ends (such as Web or Microsoft Windows) and
- customer specific backend technologies (ranging from relational databases to modeling and simulation modules to commodity clusters etc.)

DMSO HLA is a new DoD-wide standard for modeling and simulation that will likely have impact also for the enterprise computing via the OMG standardization process. HLA focus is on middleware support to enable interoperability between existing simulations whereas the user front-ends or simulation/database backends are left open for the individual simulation developers. WebHLA is filling these gaps by combining the HLA/RTI middleware standards with the new Web/commodity standards such as VBA or DirectX for the front-ends or ASP/OLEDB for the database backends. The resulting system can be customized and adapted for a variety of application domains, ranging from the DoD M&S to resource management for distributed computing to authoring or runtinme support for multi-player gaming environments.

Status: Current early prototype of WebHLA includes the following components:

- MS Office front-ends such as Excel customized via VBA scripting towards suitable authoring tools;
- DirectX based real-time multimedia graphics front-ends for multi-user gaming;
- Object Web RTI based middleware integration;
- Active Server Pages based support for dynamic HTML generation from NT relational databases such as Access or SQL Server;
- XML support for configuration management and universal scripting support;
- WebFlow based visual dataflow authoring tools.

Applications: Initial applications of WebHLA could be likely in the custom software domain with the core WebHLA technologies providing the base programming framework / skeleton. Current NPAC application for DoD, FMS Training Space falls into such category – it offers an interactive multi-user training tools for advanced M&S technologies such as SPEEDES, ModSAF, HLA etc. Another potential business application domain for WebHLA is Simulation Based Acquisition. Virtual Prototyping Environment for advanced manufacturing are already operated by Boeing or Automotives and the approach is now being advocated for by the DoD as well. WebHLA appears to offer the optimal cost effective implementation infrastructure for such systems as it integrates DoD and commodity standards.

7. Java Grande and High Performance Java

7.1 Roles of Java in Technical Computing

In this section, we focus on the role of the Java language in large scale computing. In Figure 7.1, we picture three possible roles for Java corresponding to its use in the client, middle-tier or backend tiers of Figure 2.1, respectively. The use of Java at the client is well understood and in spite of the battle between Microsoft and Netscape, seems likely to flourish. We have discussed at length the use of Java in the server tier in the previous sections and noted that Java plays a role both as an object model and as the language to build servers like JWORB which implement competing distributed object paradigms. This is perhaps the dominant use of Java in the Intranet software industry and Java is attractive here because it is a very productive programming environment. C++ currently gives higher performance as shown in Figures 5.2 and 5.4 but Java servers are already very robust and new generations of Java compilers are expected to give excellent performance. In this section, we will largely focus on the third role of Java – that as a basic programming environment. These sections are built on the results of four workshops held in Syracuse (Dec. 96), Las Vegas (June 97), Palo Alto (February 98) and Southampton (England, September 98) [X].

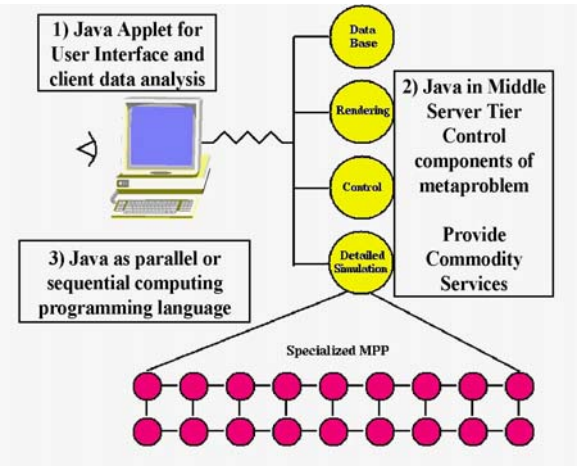

Figure 7.1: *Three Roles of Java in HPCC*

7.2 Why Explore Java as a Technical Computing Language?

Why would one use Java to code one's favorite scientific code? One can cite the usual features, collected below.

The Java Language has several good design features. It is in particular secure, safe (with respect to bugs), object-oriented, and familiar (to C, C++ and even Fortran programmers).

Java has a very good set of libraries covering everything from commerce, multimedia, images to math functions (see for instance NIST's library under development [X]). These frameworks can embody both real software (i.e. fully coded methods) but also interfaces which define standards that lead to uniform easier to use uniform environment where different vendors can focus on the best implementation of a particular service without using their own irrelevant proprietary names. It is curious that the Fortran community never agreed on the names of basic numerical libraries – we should spend our creative energy on the best coding of the best FFT algorithm and not on the subroutine calling sequence.

Java has best available electronic and paper training and support resources. There are in particular over 1000 books in Java. Industry is aggressively developing for Java the best integrated program development environments. Java naturally integrated with network and universal machine supports powerful "write once-run anywhere" model.

There is a large and growing trained labor force. Java is already being adopted in many entry-level college programming courses. In fact, it seems likely that in the future, students entering college will increasingly have Java expertise. They will learn about it from their excursion on the Internet while it should grow in interest as the language used to teach programming at middle and high schools. (Note NPAC's Java Academy [X]) which was very successfully taught each Saturday in the depth of a bleak Syracuse winter to a group of middle and high school students and teachers). Java's natural graphical (applet) view makes it a very social language whose value is more obvious to a beginning programmer than C++ or Pascal. Of course, the Web is the only real exposure to computers for many children, and the only languages, to which they are typically exposed today, are Java, JavaScript, and Perl. We find it difficult to believe that entering college students, fresh from their Java classes, will be willing to accept Fortran, which will appear quite primitive in contrast. C++ as a more complicated systems-building language may well be a natural progression, but although quite heavily used, C++ has limitations as a language for simulation. In particular, it is hard for C++ to achieve good performance on even sequential and parallel code, and we expect Java not to have these problems if the process described in sec. 7.3 is successful.

As well as these laudable positive reasons, we can also compare Java with the "competition" which is Fortran or C++ for serious technical computing. Note the following points.

Fortran77 has excellent compilers, good user base but will not be taught broadly and clearly limited in capabilities; in particular not object oriented. It appears that although Fortran90 and HPF have useful features, they have not "taken off" and it seems likely that they will not "make it". Five years ago, it looked as though C++ could become language of choice in complex scientific codes (perhaps with Fortran as inner core and especially when irregular data structures could not be easily expressed in Fortran). However this movement appears stalled – partly because this trend was halted by a growing interest in Java and users are awaiting events. C will remain widely used as a simple elegant language but object oriented techniques seem essential in large software systems and so the key competition appears to lie between C++, Fortran90 and Java. The C++ language is complex and splintered with no agreement on standards for libraries and parallelism. This is partly because its use in Grande applications is too small to motivate standards and partly due to the prevailing culture.

So we argue that although existing large-scale codes are written in Fortran C and C++, the associated unattractive and comparatively unproductive programming environment handicaps developers. Current languages and tools are sufficient but it does not seem likely that one can easily greatly improve on existing environments without a radically new approach. We suggest that it will easier to try to build an attractive technical computing environment around Java rather than the existing languages.

We can list some additional reasons why we might be more successful in Java than previous Fortran or C++ based programming environments.

- Java has some natural advantages due its internet base with threads and distributed computing built in.

- Java is a young language and we can take steps now to avoid unproductive proliferation of libraries and parallel constructs.

- We could be third (Fortran, C++, and now Java) time lucky.

- Java has the expressivity and object oriented advantages of C++ combined with performance levels of C and Fortran.

- It can use Java's clear advantages in building user interfaces as an entrée into other aspects of large-scale programming.

There are some serious problems to be solved in using Java in technical computing and now we turn to discuss these.

7.3 Java Grande

First we need to define a Grande application as any sort of large-scale or technical commercial or academic problem. Thus it subsumes areas such as:

- High Performance Network Computing or HPDC (High Performance Distributed Computing);
- Scientific and Engineering Computation;
- (Distributed) Modeling and Simulation (as discussed in Section 6);
- Parallel and Distributed Computing;
- Data Intensive Computing;
- Communication and Computing Intensive Commercial and Academic Applications;
- High Performance Computing and Communication (HPCC);
- Computational Grids.

We adopted this offbeat nomenclature, as it was hard to find a "conventional name" that doesn't get misunderstood by some community. Now Java Grande is the application of Java to Grande applications; Grandecomputers are of course compute engines used to execute Grande codes and the adjective can be used in other related ways.

The Java Grande forum [X] was set up to enhance the possibility that one can build around Java a better Grande application environment than is available through Fortran or C++. We described in the previous section why this might be possible and the Forum's sole goal is sponsor community activities designed to realize the "best ever Grande programming environment". The Forum products include recommendations and community actions that could lead to necessary changes to Java or establishment of standards (frameworks) for "Grande" libraries and services. We have had three meetings in March, May and August 1998 while we are now planning a public discussion of our initial conclusions at SC98 in Orlando for November98. The current status is given at our home page [X] while the NPAC resource [X] has more personal broader collection. The Forum is interacting in two rather different dimensions. In the near term, we need to work with the computing mainstream and Sun to discuss a few key changes in Java to allow it to be a complete efficient Grande Programming Language. This includes the floating-point processing, complex type and RMI performance issues described later.

Secondly, the Forum needs to work within the Grande community to encourage and initiate those activities that will lead to standards in such areas as numeric libraries and the Seamless Computing Interface. We suggest that the Grande community has unnecessarily handicapped progress by having as much creativity in the interfacing of its artifacts as in the essential algorithms. As we illustrate in the next section for databases, sometimes all can benefit if one agrees to standard which initially handicap particular and perhaps even the best implementations.

The Forum is set up currently with two major working groups. The *Numerics* working group is led by Ron Boisvert and Roldan Pozo from NIST and is centered on Java as a language for mathematics. Issues studied include:

- Changes in Java controversial handling of floating point which currently has goal of reproducible results but this leads to non-optimal accuracy.
- Support for efficient Complex types or classes.
- Lightweight classes and Operator overloading – this is a natural way of enabling efficient implementation of complex as a class but can also be applied in other circumstances.
- "Fortran rectangular multidimensional arrays" – Java's current multi-subscript language construct gives "arrays of arrays" which often do not lead to efficient code.
- High quality math libraries with agreed interfaces – examples are FFT, Matrix algebra, and Transcendental functions.

Performance and expressivity and their tradeoff underlie these proposed enhancements. As discussed in the four workshops on Java for Science and Engineering computation (Refs and URL's incl Europar)[X], the goal is Java compiler's that obtain comparable performance to those for C or Fortran. Marc Snir has given a very clear analysis [X] of the different issues that inhibit the performance of Java on classic array-based scientific codes. Industry efforts are mainly focussed on Just in Time compilers (JIT) which support the critical applet and servlet models of computation.

However traditional native machine specific compilers are possible and will surely be useful. It will be interesting to compare their performance with the best JIT's and see if and for what application any performance degradation for servlets and applets outweighs the convenience of their mobile dynamic portable computing model. A related issue is if the Java language version of a code has any more information for a native or JIT compiler than the VM (or Java bytecode) representation. Initial studies suggest that the VM and language versions of the code can be compiled with comparable performance.

Difficulties in compiling Java [X] include quite surprising points such as Java's rich exception framework that could restrict compiler optimizations. Users would need to avoid complex exception handlers in performance critical portions of a code. An important feature of Java is the lack of pointers and their absence, of course, allows significantly more optimization for both sequential and especially parallel codes. In general with some possible restrictions on programming style, we expect Java compilers to be competitive with the best Fortran and C compilers. Note that we can also expect a set of high performance ``native class'' libraries to be produced that can be downloaded and accessed by applets to improve performance in the usual areas one builds scientific libraries.

The charter of the second working group led by Dennis Gannon and Denis Caromel (INRIA, France), includes issues associated with coarse grain distributed scientific objects; distributed and parallel computing, *concurrency* support and *applications*. The detailed agenda includes:

- Performance of RMI or "remote method invocation" which is the attractive Java distributed object model.
- Performance of Java runtime (the virtual machine VM) with lots of threads, I/O, and large memory use.
- Parallel Computing interfaces including Java MPI binding and higher level interfaces such as that in HPJava discussed in section 7.6.
- Development of a framework for a universal Java Seamless interface to computing resources as discussed in the next section.
- Development of Grande Application benchmarks. This overlaps the activities of the first working group, which has already started an interesting numeric kernel collection [X] based on the ideas pioneered in the Java version of LinPack [X].

Both working groups have made substantial progress in the last few months with initial reports including key issues we need to bring up with Sun in both the *Numerics* and RMI performance areas. We need broad community involvement in critiquing our proposals, collecting Java Grande benchmarks, and defining standard classes and libraries. We hope to get good participation in a set of workshops on the seamless computing framework. We also need applications that will stress Java and Java runtime (the VM) with large applications – we need to find those weak links of the VM which lead to performance problems?

Note that enterprise Intranets will lead to some such scaling tests but there are some features that will only appear with Grande problems. European involvement in the Grande Forum has been excellent so far with Denis Caromel, Vladimir Getov, and Michael Phillipsen as major contributors. Organizationally NAG and the Edinburgh Supercomputer Center are members.

7.4 Java Seamless Computing Framework or CORBA Facility for Computation

Thinking about computing abstractly, we see that at a high level, it involves a collection of entities – computers, programs and datasets -- which can all be considered as objects with properties and methods. It is thus attractive to consider establishing a uniform terminology to allow definition of interoperable modular tools and procedures (services). Three major areas that could benefit from this standard definition for computer related distributed objects are seamless interfaces, metacomputing and performance modeling.

We term a seamless interface as some sort of user interface allowing the invocation of a given program on any one of multiple backend computers. Metacomputing refers to the more complex situation where several linked modules are run synchronously on multiple geographically computing resources. Sophisticated performance analysis systems need this type of distributed object structure to allow structured specification of the performance characteristics of computing resources and of the resource needs of job modules. The seamless interface is perhaps the simplest case; metacomputing needs in addition specification of the linkage between jobs while the performance estimation application of course requires additional properties specifying performance characteristics. However these three application areas have a set of core properties that are in common.

We are starting the community activity needed to establish first requirements and then explore a possible standard interface. The initial workshop will be co-sponsored by the Java Grande Forum and the Globus metacomputing group. We can set up standards in many ways – it can be an XML specification; a COM or CORBA IDL or a set of Java Interfaces and methods. These different representations can easily co-exist as different external specification of the same object properties.

In the following we will focus on the Java implementation as there is a very natural use of the seamless computing interface as the target of Web-based computing interfaces using Java applets on the client user front end. If we can agree on the seamless framework, then we can allow multiple developers of different front ends customized to different user needs to each access the same set of back end computers.

In Java frameworks and libraries are related concepts. Frameworks have a mix of interfaces (agreed syntax for methods and properties but allowing multiple implementations) and coded classes. On the other hand, libraries are always essentially fully instantiated. Well known Java frameworks cover the enterprise, commerce, 2D and 3D scene specification, multimedia and security areas. A Java framework is a set of agreed Java calls (as discussed above these are mainly Interfaces and not methods) to capabilities expressed in implementation neutral form. Then drivers convert these general calls to vendor specific implementation of the service This framework Java code can either be all on client (2-tier) or on client and middle tier (3 or 4 tier).

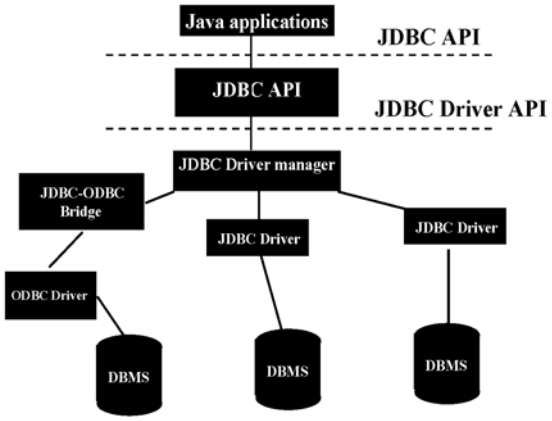

Figure 7.2: *Architecture of JDBC Database Connectivity Framework*

Let us briefly discuss one well-known example framework – that of JDBC (Java Database Connectivity) which is a universal interface to all relational databases. Hence it has some similarities to the seamless interface, which as proposed is the universal interface to all computers (as opposed to just databases running on computers). The architecture shown in Fig. 7.2 has an application JDBC API as well as the capability of linking it different managers for the different database types.

This can be implemented in a multi-tier fashion with the Java JDBC calls either on client, middle-tier server or like Oracle's PL/SQL most efficiently of all on the database back end. We show some of these different implementations on fig 7.3.

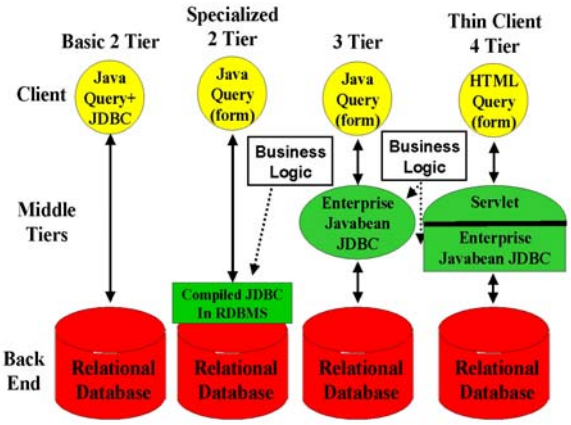

Figure 7.3: *Multi Tier Implementations of JDBC*

Note that adoption of JDBC was not without cost for the database vendors because it immediately implied that proprietary solutions such as PL/SQL were of less importance. However essentially all vendors must however support JDBC because of competitive pressure. Note that the resultant "seamless" interface increases the ease of use of databases and so increases the overall market. Thus the loss of proprietary advantages has positive side effects. In the resultant standards based environment, vendors compete for a larger market with compliant modules, which compete on basis of issues like performance and cost. Similarly the proposed seamless computing framework will allow vendors to compete on either the user front end (GUI) or back end services with the JSCF framework providing universal linkage. The JSCF framework could be implemented at the backend as a set of drivers, which relate the generic Java framework interfaces to particular software (e.g. invocation of a compiler) on particular machines. One would include JSCF support as a requirement in system purchases and so encourage vendors to implement support for the standard.

Setting up a community framework such as JSCF requires reaching agreement by "suitable interested parties" on what are the services and then what are the interfaces for a given service. This process as with JDBC and Oracle's PL/SQL can lead to difficult choices by some organizations. We had already started to accumulate possible services and their features [X] for a workshop on seamless interfaces at Reading England, September 97. In our initial planning we intend to include existing ideas from metacomputing projects such as Condor Globus Legion and from performance estimation projects PACE POEMS and PetaSIM [X]. Obviously existing seamless interface projects such as WebSubmit (NIST) [X} and UNICORE (German Supercomputer consortium) [X] need to be included.

Possible Services in a Java Seamless Computing Framework include:

- Grande Resource discovery, allocation and scheduling where the recently announced JINI technology (see Section 4.1) from Sun looks attractive
- Compiling, Executing, Specification of features needed for execution optimization. This includes parameters needed by MPI/HPF decompositions such as number of processors
- Resource Management and Scheduling jobs as in Codine [X] or LSF [X] or commercial NT environments
- Accounting where one could use features from the rapidly developing Web commerce technology?
- Authentication, and Security which is especially hard in metacomputing where one naturally links computer resources with several different management policies. Here again Internet commerce and its associated public key infrastructure will give useful building blocks.
- Sharing, Accessing and Storing into File Systems
- Data and Performance Visualization Interface
- Performance measurement and recording (cf: Pablo SDDF [X])
- Interfaces for Programming Tools
- Debuggers
- Computational Steering and Interpreted Execution interfaces
- Libraries including names in generalized Math class which the Java Grande is defining in its *numerics* working group
- Module linkage model for metaproblems (multidisciplinary applications) although this could be "outside scope". Earlier we showed how the Java AWT event model as used in the JavaBean framework (see Figure 3.4) could be used for this

The preliminary list of services given above will surely be refined and elaborated with further study. They need to be integrated into an object model for distributed computers and problems which will then have these services as methods and further properties.

7.5 Parallelism in Java

In order to discuss parallelism in Java, we divide the forms of parallelism seen in applications into four broad categories.

Modest Grain Size Functional Parallelism : e we are thinking of the type of parallelism used when computation and I/O operation are overlapped as exploited extensively by web browsers but can also be seen in possible overlap of message traffic with computation in the Java plus MPI parallel programming paradigm of sec. 7.7. This parallelism is built into the Java language with threads but has to be added explicitly with (thread) libraries for Fortran and C++. In the browsers, one typically sees the different components (multiple images, applets etc.) processed by different threads concurrently.

Object Parallelism : This is quite natural for C++ or Java where the latter can use the applet mechanism to portably represent and invoke concurrently objects. We have already discussed implicitly this form of parallelism in section 6 for distributed military simulations that use large-scale object based models where we have suggested the WebHLA framework. Note that one could include "metaproblems" (described in category 3 below) in this category for there is a continuum of objects used in HLA and similar distributed simulations spanning fine to coarse grain sizes.

Metaproblems: This is the parallelism in applications that are made up of several different sub-problems which themselves may be sequential or parallel. We have already discussed in the earlier sections, the power of Java in this case for overall coarse grain software integration or what is essentially equivalent, the linkage of distributed (coarse grain) objects. This category is seen in the use of Java in CORBA and web servers in the middle tier and is explicitly discussed in sec. 3, which describes linkage of Java modules using WebFlow for multidisciplinary applications.

Data Parallelism: Here we refer to natural large-scale parallelism found from parallel updates of grid-points, particles and other basic components in scientific computations. Such parallelism is supported in Fortran (or C) by either high-level data parallel HPF or at a lower level Fortran plus message passing (MPI). Java does not have any built in parallelism of this type, but at least the lack of pointers means that natural parallelism is less likely to get obscured than in C or C++. There seems no reason why Java cannot be extended to high level data parallel form (HPJava) in a similar way to Fortran (HPF) or C++ (HPC++). We describe in sec. 7.6, an effort at NPAC [REF38], which is focussing on a high level SPMD rather than the traditional HPF style approach to data parallel Java. Data parallelism can be supported using threads on shared memory machines as pioneered at Indiana [REF] whereas in distributed memory machines, explicit message passing must be used at the low level. This leads to the hybrid model of fig 7.4. One advantage of distributed shared memory architectures is their support of a single Java VM and a uniform programming model. Comparing with the discussion of category 2 above, we see that threads can of course we used to support multiple forms of relatively fine grain parallelism. Message passing is clearly satisfactory for Java as the language naturally supports inter-program communication, and as described in sec. 7.7, the standard capabilities of high-performance message passing are being implemented for Java. One can see the linkage of the natural distributed computing message-passing models built into Java with the specialized MPI/PVM style systems in parallel computing.

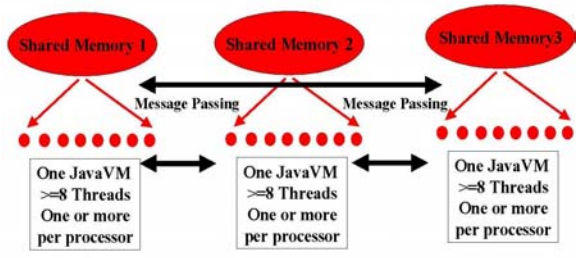

Figure 7.4: *Hybrid Java Model for Parallelism*

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

7.6 HPspmd and HPJava: Pragmatic Data Parallelism

Data parallelism is obviously very important in High Performance scientific computing and currently explicit user specified parallelism and message-passing libraries are the dominant approach to scaling data parallelism. Much research has gone into more attractive higher level programming models such as HPF. Currently this has promising features and some success with the Grande user community. However the complexity of the compiler has delayed deployment and prevented timely implementation of critical capabilities. Considering a rapidly moving field such as Java, now does seem the time to consider approaches requiring complex compilers with a lengthy development cycle. Further experience ha shown that library based approaches to Grande programming are quite successful – one hides the

complexity of message passing by invoking parallel libraries from an SPMD environment. Thus we introduced what we call the *HPspmd programming model*, where we provide some of the HPF distributed array capabilities through a library interface rather than through a compiler analysis of either loops or Fortran90 syntax as in HPF [X]. The library calls support communication and arithmetic and include manipulation of "ghost cells" used in many regular problems. This concept is designed to have base technology implementation complexity, and ease of use lying in between those of MPI (straightforward to develop the MPI routines but low level user interface) and HPF (high level elegant environment but with the need for a major compiler development activity). Note *HPspmd* programs can intermix pure MPI and in HPF through its extrinsic interface with the *HPspmd* library calls. Although *HPspmd* can be applied to any language, we have designed a prototype Java environment that we call HPJava [X]. HPJava is built on top of MPI and comes with a full featured Java interface to MPI mentioned in the next section. As well as MPI for low-level message passing, HPJava uses the library Adib built at NPAC to support HPF for regular collective operations. One could add further capabilities such as Global Arrays for 1-sided access to remote data, and CHAOS for irregular collective operations. We compare the expression of a simple block decomposed vector B and array with four processors in HPF and HPJava in Table 1.

Typical HPJava Specification of Distributed Arrays	Corresponding HPF Specification of the same distributed Arrays
Procs p = new Procs1(4) ; Range x = new BlockRange(100,p.dim(0)); float [[,*]] a = new float [[x, 100]] on p ; **float [[]] b = new float [[x]] on p ;**	!HPF$ PROCESSOR P(4) !HPF$ DISTRIBUTE T(BLOCK) ONTO P REAL A(100,100) !HPF$ ALIGN A(:,*) WITH T(:) REAL B(100) !HPF$ ALIGN B(:) WITH T(:)

Table 1 *Comparison of HPF and HPJava Specification of two arrays B(100), A(100,100) spread in one dimension block fashion over 4 processors*

In the following Table 2, we show how red-black iteration on a two dimensional array can be expressed in HPJava. Note that the [[]] notation is used to signal a distributed array. We translate the special syntax used in HPJava, into conventional Java plus library calls and then invoke your favorite Java compiler.

```
Procs p = new Procs2(NP, NP) ;
on(p) {
  Range x = new BlockRange(N, p.dim(0), 1) ;   // ghost width 1
  Range y = new BlockRange(N, p.dim(1), 1) ;   // …
  float [[,]] u = new float [[x, y]] ;
  for(int parity = 0 ; parity < 2 ; parity++) { //  red and black
    Adlib.writeHalo(u, widths) ; // Communicate Ghost Cells

    overall(i = x [1 : N - 2])
      overall(j = y [1 + (x.idx(i) + parity) % 2 : N - 2 : 2])
        u [i, j] = 0.25 * (u [i - 1, j] + u [i + 1, j] +
                           u [i, j - 1] + u [i, j + 1]) ;
  }
}
```

Table 2 *Red-Black Interation in HPJava*

In Fig. 7.5, we give some preliminary HPJava performance numbers for a simple Jacobi iteration where HPJava outperforms Java due the HPJava translators use of efficient one dimensional arrays to represent the distributed [[]] syntax in Table 1. Note that communication in HPJava is quite efficient as its uses native calls to C++ libraries to handle array operations.

7.7 Java links to MPI

Naturally one can implement an MPI linkage for Java and this has been implemented by NPAC (mpiJava in [24] and [25]) and a Westminster College London group ([28] and [31]). Further Mississippi State has announced the availability of a commercial version JMPI [29]. Although there are Fortran C and C++ defined bindings to MPI [27], there is no formal definition of Java binding to MPI. Hence different approaches have been pursued. The Westminister College version automatically links C version of MPI to Java Native Interface (JNI).

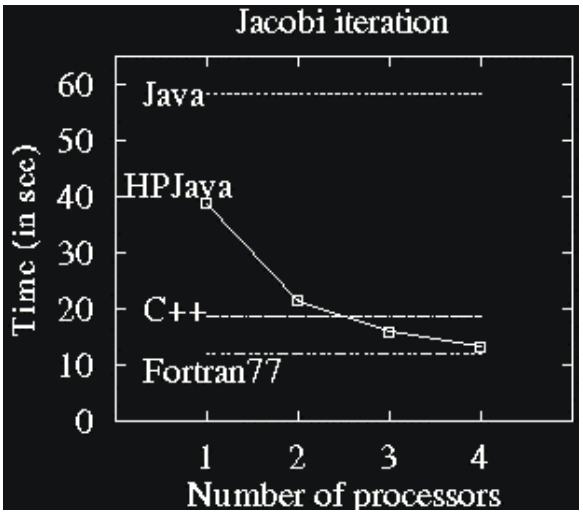

Figure 7.5: *Preliminary Performance of HPJava*

On the other although still using JNI, the NPAC version mpiJava "optimizes" the Java link based on C++ MPI standard by exploiting several features of the object oriented structure of Java. The NPAC version could be further extended to use serialization and pass any array of Java Objects. The Mississippi State version goes one step further with a version even more tuned to Java.

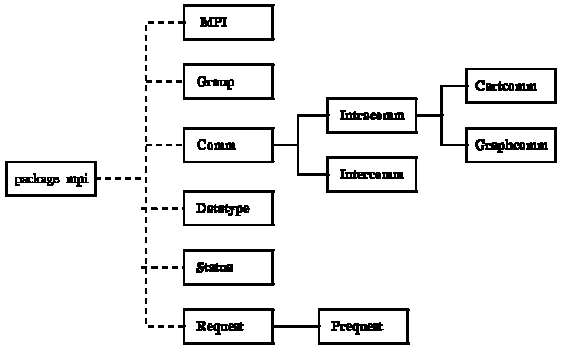

Figure 7.6: Class Structure of mpiJava

In the remainder we give a few more details of mpiJava which is a fully featured Java interface to MPI 1.1 but as described above using an Object-oriented API based on MPI 2 standard C++ interface. One interesting feature is the comprehensive test suite translated from IBM MPI suite and which was used to test all the myriad of MPI functions. JavaMPI is available for Solaris, Windows NT and other platforms. Fig 7.6 defines the class structure of mpiJava, which is freely available for download from [24]. One can illustrate the MPI Java binding with the simple method invocation to send an array of characters to all processors from the mpiJava call:

```
MPI.COMM_WORLD.Send(message,0,
message.length, MPI.CHAR, 1, 99) ;
```

The initial implementation has been benchmarked with results shown in fig 7.7 and 7.8 for both shared memory and distributed memory implementations on multiprocessor PC and Sun Solaris UNIX platforms. We used the WPI NT version of MPI [32] and the well-known MPICH for UNIX. The figures compare C and Java implementations and we see that not surprisingly the extra overhead of the Java is most important in the Shared Memory mode where the latency is smallest anyway. In shared memory, the zero byte message transfer time increases from 70 μs to 160 μs for NT and 150 μs to 375 μs for UNIX. In distributed memory mode, the increase is from 620 μs to 690 μs for NT and 680 μs to 960 μs for UNIX. Obviously these Java measurements are quite ephemeral as the compiler technology is rapidly improving. However even now the Java MPI binding appears to have acceptable performance.

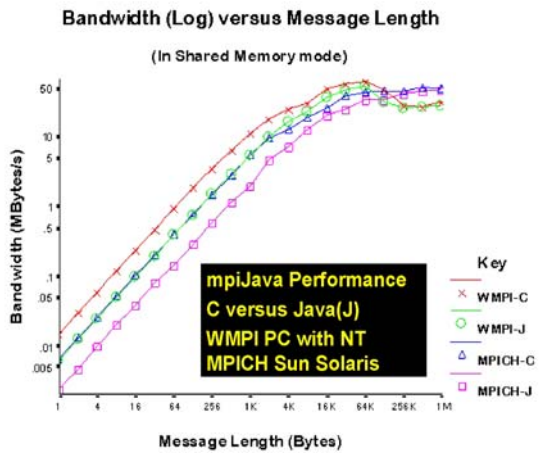

Figure 7.6: *Performance of Shared Memory mpiJava for C and Java on UNIX and NT*

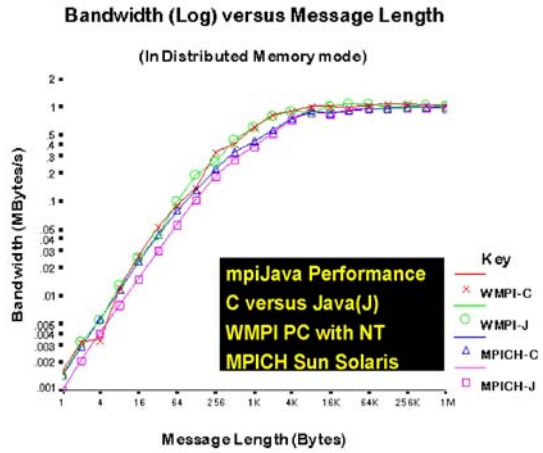

Figure 7.7: *Performance of Distributed Memory mpiJava for C and Java on UNIX and NT*

8. HPcc and Parallel Computing

Most of the discussion in this paper has been devoted to the use of commodity technologies for computational grids or the field that is sometimes termed HPDC (High Performance Distributed Computing). However as we explored in the last subsection in the special context of Java, we believe that one can also use commodity technologies to build parallel computing environments, which combine both high functionality and high performance. As usual the functionality comes from inheriting commodity services and the challenge is to combine these with high performance. We assert that this approach will lead to sustainable attractive programming environments. In metacomputing, the HPcc approaches of sections 3, 5 and 6 leads to novel high level environments for there are few pre-existing high level metacomputing and distributed simulation environments. In parallel computing, HPcc offers an alternative to other approaches which could be advantageous as HPcc leads to uniform vendor neutral, productive and sustainable (lower maintenance costs) programming environment.
First compare the two views of a parallel computer in Figs. 8.1 and 8.2.

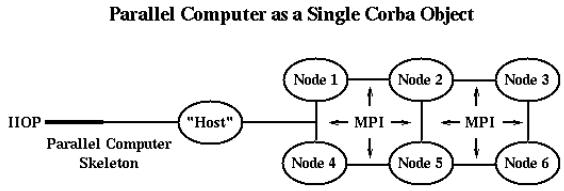

Figure 8.1: *A Parallel Computer viewed as a single CORBA object in a classic "Host-node computing model". Logically the "Host" is at tier-2 and the nodes are at tier 3. The Physical architecture could differ from the logical architecture.*

In the above figure, we see a simple multi-tier view with commodity protocols (HTTP, RMI, COM or the IIOP pictured) used to access the parallel computer as a single entity. This entity (object) delivers high performance in an obvious way by running classic HPCC technologies (such as HPF, PVM or the pictured MPI) in the third tier. This is model pictured in Fig 1.1 and 9.1 where the parallel machine is viewed as a single back end service. The seamless interface of sec 7.4 would again interface to this a single computer with its parallel architecture reflected in the value of certain properties such as the number of nodes. This could be considered a representation of the "host-node" model of parallel programming. Note that in figs. 8.1 and 8.2, we draw various nodes and the host as separate entities. These represent logically distinct functions but the physical implementation need not reflect the distinct services. In particular, two or more capabilities can be implemented on the same sequential or SMP system.
In Fig. 8.1, we are not exploiting the distributed computing support of commodity technologies for parallel programming. However in Fig. 8.2, we view the parallel computer as a distributed computer with a particularly fast network and integrated architecture.

In Fig. 8.2, each node of the parallel computer runs a Corba ORB, Web Server or equivalent commodity server. Now commodity protocols can operate both internally and externally to the parallel machine. This allows a particularly powerful environment where one can uniformly address the full range of commodity and high performance services. Further tools such the visual environment of sec. 6.3 can now be applied to parallel as well as distributed computing. Obviously one should be concerned that this flexibility has been accompanied by a reduction

in communication performance from that of Fig. 8.1. Indeed most messaging protocols such as RMI, IIOP and HTTP have unacceptable performance for most parallel computing applications. However we can use the ideas of sec. 3.2 to obtain good performance with a suitable binding of MPI and PVM to the commodity protocols. In Fig. 8.3, we redraw Fig. 2.3 in a fashion to demonstrate the analogy to Fig 8.2.

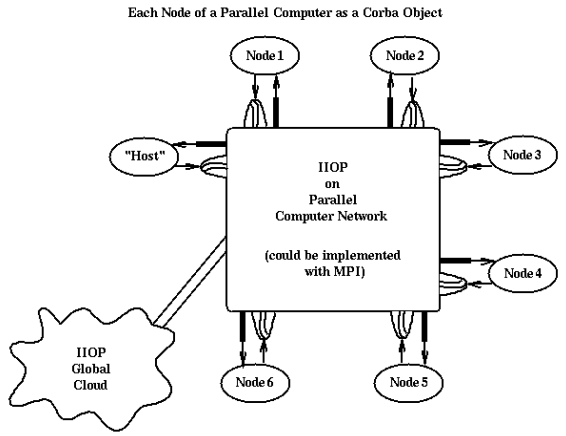

Figure 8.2: *Each node of a parallel computer instantiated as a Corba object. The "Host" is logically a separate Corba object but could of course be instantiated on the same computer as one or more of the nodes. Using the protocol bridge of Fig. 8.4, one could address objects using Corba with local parallel computing nodes invoking MPI and remote accesses using Corba where its functionality (access to very many services) is valuable.*

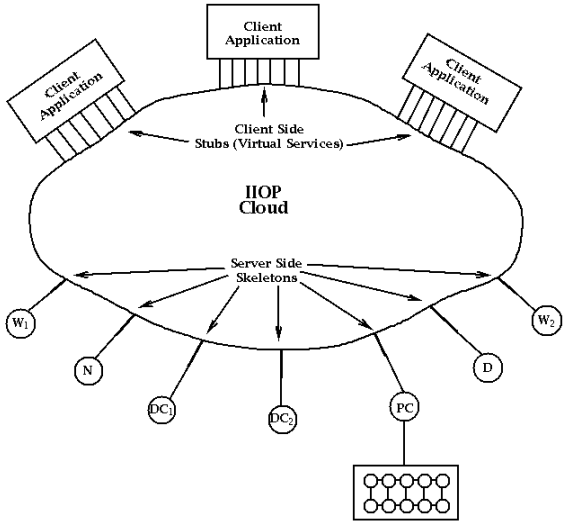

Figure 8.3: *Pure Corba architecture for the heterogeneous DcciS services of Fig. 2. There is a similar Java version of this using RMI and JDBC with of course the linked application being restricted to Java code. Corba and the analogous COM solution are cross-language solutions.*

In Fig. 8.4, we extend the previous figure to show an approach to high performance, which uses a separation between messaging interface and implementation. The bridge shown in this figure, allows a given invocation syntax to support several messaging services with different performance-functionality tradeoffs.

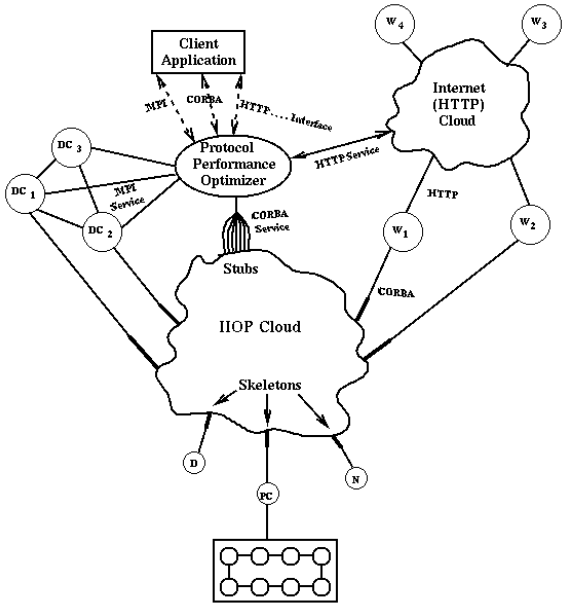

Figure 8.4: *A message optimization bridge allows MPI (or equivalently Nexus or PVM) and commodity technologies to coexist with a seamless user interface.*

In principle, each service can be accessed by any applicable protocol. For instance a Web Server or database can be accessed by HTTP or Corba; a network server or distributed computing resource supports HTTP Corba or MPI. Of course one can substitute equivalent commodity (RMI, COM) or HPCC technologies (PVM, PVMPI, Nexus) in the above discussion. Note that there are two ways of linking MPI and Corba. Firstly there is the MPI function call that actually calls a Corba stub; secondly a Corba invocation can be trapped and replaced by an optimized MPI implementation. One could for instance extend the MPI communicator field to indicate a preferred protocol implementation and this could be an interesting capability of the Java-MPI linkage of sec. 7.7. This preference can be set using the mechanism of sec. 3.2. As discussed in the Java Grande forum, there are important research issues in efficient object serialization needed for a high performance implementation of Fig. 8.4.

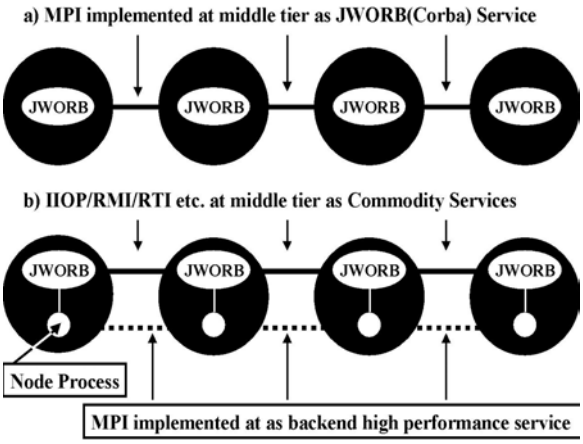

Figure 8.5: *(a) Middle Tier and (b) High Performance backend implementations of MPI in HPcc*

There are two versions of parallel processing in HPcc when we adopt the view of running middle tier servers on each processor of an MPP. The simplest is shown in Fig. 8.5(a) where we implement all message traffic at the middle tier level so as in our current RTI implementation, we use base JWORB or equivalent services for MPI. In Fig. 8.5(b), we use the hybrid approach of sec.3 where each node of the parallel runs both a commodity server and a "native high performance MPI" interacting with the typical node programs. The commodity server is providing Object Web services in the same spirit that UNIX or NT on each node of most current MPP's provide operating system services. The network of JWORB servers in Fig. 8.5(b) provide an important management layer -- using perhaps HLA as discussed in sec. 6.3 -- as here we do not need high performance but rather the rich functionality of the middle tier.

9. Conclusions: A Multi Tier Grande Computing System

So suppose you think HPcc and/or Java Grande is interesting. How does this impact your implementation of a computing environment? The situation with Java Grande is perhaps clearest as now is not the time to deploy it except on clients and servers; we should experiment with and evaluate the use of Java as a scientific programming language.In more detail we summarize situation as:

- Don't need to rewrite existing codes in Java!
- Instead use Java freely at client and middle (server or "gateway") tier and wrap existing codes as CORBA or Java distributed objects. This will give you all the advantages of Java for distributed objects
- Conduct suitable experiments in using Java in complete Grande applications
- Make certain your interests are represented in Java Grande Forum
- Retrain your staff in Java Web and distributed object technologies
- Put "High Performance Grande Forum compliant" Java support into your RFP's for hardware and software

Now suppose you wish to put together a computing environment that builds in some of the ideas described here. We can expand figure 1.1 to a more explicit Grande computer application in figure 9.1 which illustrates the richness of the use of the middle tier. Here we call the middle tier a Gateway as it naturally acts as this to the set of available special purpose backend systems such as databases and massively parallel computers. The component called "Gateway Control" in Fig. 9.1 is intended to implement the seamless interface described in sec 7.4. The remaining middle tier components in Fig. 9.1 are servers (CORBA, servlet etc. in general sense of pragmatic object web) either implementing directly or invoking proxies to the backend for the labeled services. We see in this figure multiple supercomputers in the backend -- one doing CFD simulation of airflow; another structural analysis while in more detail you have linear algebra servers (NetSolve [X] from Tennessee); Optimization servers (NEOS [X] from Argonne); image processing filters (using perhaps Khoros [X]); databases (here a particularly rich example is the NCSA Biology workbench [X]); visualization systems (from low end PC systems to AVS [X] to virtual environments in CAVEs [X]).

All these services are linked together and to commodity services such as collaborative information systems in the sea of middle tier. The example shown is roughly aimed at multi-disciplinary applications to design of vehicles but other areas would produce a qualitatively similar situation.

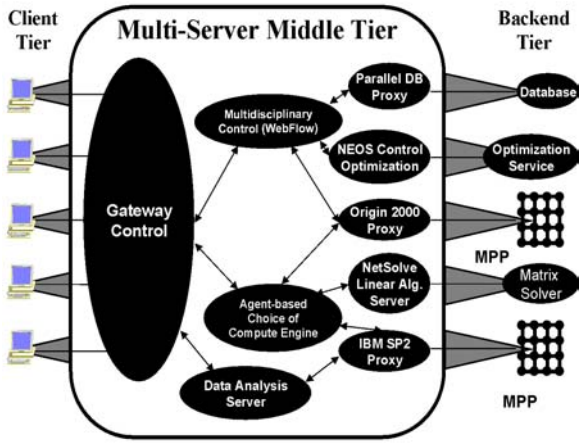

Figure 9.1: *Typical HPcc implementation of a complex high performance multidisciplinary application.with the middle tier naturally implemented on the Gateway machine*

We can enumerate capabilities of middle tier as illustrated above and discussed throughout this paper – especially in sec. 3.

- Seamless Interface in "Gateway Control" -- an Enterprise Javabean which processes input from user's Java Applet interface and maps user generic commands to those on specific machine as discussed in sec. 7.4
- Accounting, Security, Compiling Interface, Seamless Tools Interface, global data and file system interface
- Resource management of heterogeneous MPP backend (linked to seamless interface)
- Database and Object Brokers; Network servers like NetSolve and NEOS
- Uses agents (as used in NetSolve) to determine optimal execution platform
- Collaboration Servers including Habanero [X], Tango [X], Lotus Notes [X] etc.
- Visualization Servers
- "Business Logic" to map user data view (e.g. objects) to persistent store (e.g. Oracle database) and simulation engine (MPP) preferred format

Now we suggest that the logical architecture of Fig. 9.1 could naturally imply a physical architecture as the hardware and software tradeoffs of the middle tier are of course different from those of a backend resource. Roughly you can say that middle tier is naturally supported by a modern business enterprise system – typically a shared memory SMP.

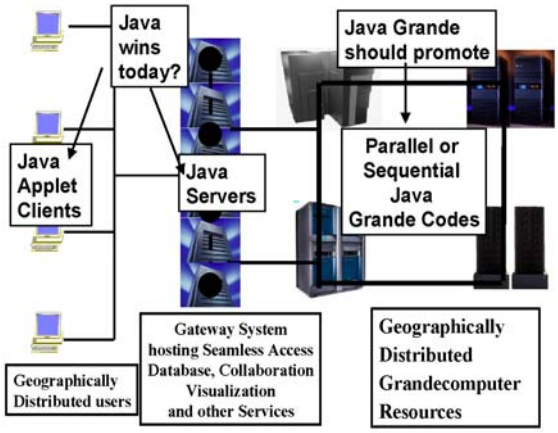

Figure 9.2: *The Hybrid Gateway Architecture implemented with separate middle tier, client and backend hardware.*

Several subsystems and applications will run on the Gateway or middle tier.

- The 90% of users who only need HPCC occasionally!
- Most of a Command and Control application.
- Several FMS (forces modeling) and IMT(Integrated modeling and testing) applications using the WebHLA of sec. 6.
- Some I/O and database intensive applications. Note business enterprise systems are natural hosts for commercial databases.
- High value services with modest computational needs e.g. grid generation and other pre-processing, data manipulation and other post-processing.
- Video Servers for training and education.
- Design and planning tools.
- "Glue" such as WebFlow servers for multidisciplinary interactions.
- Control of metacomputing applications.
- Capabilities such as the new JINI Java resource registration and discovery service.

Although these are simple concepts, we note that the structure of figs. 9.1 and 9.2 are not seen in many Grande computing environments and the need for Gateway middle tier systems is not reflected in many solicitations for center hardware and software. We suggest that it will be fruitful to plan and support such hybrid Grande computing environments.

10. Acknowledgements

This work is partially supported by the DoD High Performance Computing Modernization Program (HPCMP) ARL and CEWES Major Shared Resource Centers (MSRCs) through the Forces Modeling and Simulation (FMS) Programming Environment and Training (PET) programs based on contracts DAHC94-96-C-0010 with Raytheon E-Systems at ARL and DAHC94-96-C-0002 with Nichols Research Corporation at CEWES.

We are grateful to Bob Wasilausky, the FMS Computational Technology Area (CTA) Lead for his interest in our work and the overall guidance.

Tom Haupt leads NPAC projects summarized in Section 3.3 and aimed at WebFlow technology transfer to the HPC community. Bryan Carpenter leads NPAC HPJava project. Mark Baker collaborates with NPAC and leads the seamless computing effort in U.K.

Several Syracuse University students contributed to the development activities discussed in the paper. OMBuilder was started by Balaji Natarajan and continued by Sachin Shanbhag. Bruno Goveas constructed WebFlow interface to Jager. Sunil Jos, Mahesh Rengaswamy and Abbasaheb Yadev contributed to the FMS Training Space. Samuel Jos contributed to the CareWeb project. Tom Pulikal contributed to the Data Mining project. Subhash Nair leads the DirectX / DirectPlay based simulation front-end development. Zeynep Odcikin Ozdemir leads the parallel CMS project.

JavaGrande team includes:

11. References

[1] 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 , June '97, in *Concurrency: Practice and Experience.*

[2] R. Orfali and D. Harkey, Client/Server Programming with Java and CORBA , 2nd Edition, Wiley 1998.

[3] G. C. Fox, W. Furmanski, H. T. Ozdemir and S. Pallickara, Building Distributed Systems for the Pragmatic Object Web, Wiley 1998 book in progress.

[4] G. C. Fox, W. Furmanski, B. Goveas, B. Natarajan and S. Shanbhag, WebFlow based Visual Authoring Tools for HLA Applications, in Proceedings of ITEA'98, July 98, Aberdeen, MD.

[5] G. C. Fox, W. Furmanski, S. Nair and Z. Odcikin Ozdemir, Microsoft DirectPlay meets DMSO RTI for Virtual Prototyping in HPC T&E Environments , in Proceeding of ITEA'98, July 98, Aberdeen, MD.

[6] G. C. Fox, W. Furmanski and H. T. Ozdemir, Object Web (Java/CORBA) based RTI to support Metacomputing M&S , in Proceedings of ITEA'98, July 98, Aberdeen, MD.

[7] G. C. Fox, W. Furmanski and T. Pulikal, Evaluating New Transparent Persistence Commodity Models: JDBC, CORBA PPS and OLEDB for HPC T&E Databases , in Proceedings of ITEA'98, July 98, Aberdeen, MD.

[8] 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/

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

[10] 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

[11] 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).

[12] 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.

[13] [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/

[14] D. Bernholdt, G. Fox and W. 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/

[15] 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

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

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

[18] LEGION, *http://www.cs.virginia.edu/~legion/*

[19] E. Akarsu, G. Fox, W. Furmanski and T. 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

[20] I. Foster and C. Kessleman, Globus Metacomputing Toolkit, http://www.globus.org

[21] 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.

[22] Bryan Carpenter, Yuh-Jye Chang, Geoffrey Fox, Donald Leskiw, and Xiaoming Li. Experiments with HPJava. Concurrency: Practice and Experience, 9(6):633, 1997.

[23] Bryan Carpenter, Yuh-Jye Chang, Geoffrey Fox, and Xiaoming Li. Java as a language for scientific parallel programming. In 10th International Workshop on Languages and Compilers for Parallel Computing, volume 1366 of Lecture Notes in Computer Science, pages 340--354, 1997.

[24] javaMPI Home Page http://www.npac.syr.edu/projects/pcrc/mpiJava

[25] Mark Baker, Bryan Carpenter, Geoffrey Fox, Sung Hoon Ko and Xinying Li. *mpiJava: A Java MPI Interface* in Java Grande workshop at Europar, September 2-3, 1998, Southampton.

[26] Adam J. Ferrari. JPVM: Network parallel computing in Java. In ACM 1998 Workshop on Java for High-Performance Network Computing. Palo Alto, February 1998, Concurrency: Practice and Experience, 1998. To appear.

[27] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. University of Tennessee, Knoxville, TN, June 1995. http://www.mcs.anl.gov/mpi.

[28] Vladimir Getov, Susan Flynn-Hummel, and Sava Mintchev. High-Performance parallel programming in Java: Exploiting native libraries. InACM 1998 Workshop on Java for High-Performance Network Computing. Palo Alto, February 1998, Concurrency: Practice and Experience, 1998. To appear.

[29] George Crawford III, Yoginder Dandass, and Anthony Skjellum. The jmpi commercial message passing environment and specification:Requirements, design, motivations, strategies, and target users. http://www.mpi-softtech.com/publications.

[30] Sava Mintchev and Vladimir Getov. Towards portable message passing in Java: Binding mpi. Technical Report TR-CSPE-07, University of Westminster, School of Computer Science, Harrow Campus, July 1997.

[31] Narendar Yalamanchilli and William Cohen. Communication performance of Java based parallel virtual machines. In ACM 1998 Workshop on Java for High-Performance Network Computing. Palo Alto, February 1998, Concurrency: Practice and Experience, 1998. To appear.

[32] WMPI - http://dsg.dei.uc.pt/w32mpi

[33] The OMG DARPA Workshop on Compositional Architectures

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

[35] The JacORB Object Request Broker

[36] The OmniOrb Object Request Broker

[37] FMS Training Space

[38] WebFlow Home Page http://osprey7.npac.syr.edu:1998/iwt98/products

[39] W3C HomePage http://www.w3c.org for XML/RDF specifications

[40] UML, http://www.rational.com/uml

[41] PCRC, http://www.npac.syr.edu/projects/pcrc/

[42] POOMA, http://www.acl.lanl.gov/pooma/main.html

12 Glossary

Active Server Pages: see ASP.

ActiveX: marketing name for a collection of Microsoft technologies related to dynamic Internet programming. ActiveX builds on top of COM and extends the previous OLE controls based Automation, restricted to MS Office applications, to the Internet domain. In particular, ActiveX control, ofen viewed as a competitive construct a JavaBean, is a COM component that can be downloaded and installed in the Internet Explorer container.

ADO: *Abstract Data Objects* - a COM library that offers high level API on top of OLEDB – Microsoft transparent persistency layer.

Applet: an application interface where referencing (perhaps by a mouse click) a remote application as a hyperlink to a *server* causes it to be downloaded and run on the client.

ASP: *Active Server Pages* – Microsoft server-side technology that facilitates creation of dynamic HTML pages. An ASP file, marked as such by the .asp extension, is being automatically preprocessed by the Microsoft Internet Information Server before it is sent to the Web client as an (dynamically modified) HTML page. All text contained between <% … %> brackets is interpreted as Visual Basic Script and executed by the server, the surrounding brackets are ignored, and the resulting textual HTML output is inlined in the server-side-include style, replacing the code the generated it.

Attribute: that part of an IDL interface that is similar to a public class field or C++ data member. The idltojava compiler maps an OMG IDL attribute to accessor and modifier methods in the Java programming language. CORBA attributes correspond closely to JavaBeans properties.

Avatar: geometry within a VRML scene that represents the user. The location of the avatar corresponds to the users viewing position.

AVS: *Advanced Visualization System* – a visual dataflow authoring and runtine system, developed originally by Stardent Computer and now continued by AVS, Inc., popular in high-end research labs and typically used for advanced visualization tasks, often coupled with high performance computing kernels.

AWT: *Abstract Windowing Toolkit* - Java package distributed by Javasoft as part of JDK that offers support for platform-independent windowing and GUI programming; for example, Java AWT objects are rendered using X/Motif on UNIX screens and using Win32 on Windows.

Behaviour: a general term in VRML 2.0 to animate objects and to specify interactions among objects through output events, input events. routes, sensors and scripts.

BOA: *Basic Object adapter* – a component of CORBA model, responsible for activating and sustaing objects, identified by a CORBA broker and served by CORBA servers. BOA design is now being replaced by a new, more scalable design of POA.

Callback: a mechanism by which a remote object sever, invokes a method on a client (the client reference needs to be registered with the server). This allows for asynchronous notification of events as opposed to polling by the client

CareWeb: Web based healthcare community network, including patient record database, collaboratory diagnosis and medical education support. The system was prototyped by NPAC and is currently productized by Translet.

CDR: *Common Data Representation* **-** CDR is tailored to the data types supported in the CORBA Interface Definition Language (IDL), handles inter-platform issues such as byte ordering (big-endian / little-endian)

CGI: *Common-Gateway-Interface* – a method for communication between a browser and a server for processing user input through a script and generating output that is sent back to the user. This script is usually placed in the cgi-bin directory of the server.

CHSSI: *Common HPC Software Support Infrastructure* **–** one of three main programmatic thrusts of the DoD HPC Modernization Program (the other two being represented by Grand Challenges and PET), focused on core software infrastructure for specialized Computational Technology Areas such as Computational Fluid Dynamics, FMS etc.

Client: any code which invokes an operation on a distributed object. A client might itself be a CORBA object, or it might be a non-object-oriented program, but while invoking a method on a CORBA object, it is said to be acting as client.

Client Stub: a Java programming language class generated by idltojava and used transparently by the client ORB during object invocation. The remote object reference held by the client points to the client stub. This stub is specific to the IDL interface from which it was generated, and it contains the information needed for the client to invoke a method on the CORBA object that was defined in the IDL interface.

CLSID: *Class Indetifier* **–** a unique very long integer, generated automatically by the system and used as class identifier by the Microsoft COM model.

CMS: *Comprehensive Mine Simulator* **–** an advanced DIS simulation of minefields, their components and individual mines, cooperating with ModSAF terrain and vehicle simulation modules; CMS is developed by the countermine R&D team at the Nigh Vision Lab, Ft. Belvoir, VA.

CODE: *Computationally Oriented Display Environment* **-** a visual parallel programming system, developed by Jim Browne at the University of Texas, that users to compose sequential programs into a parallel one. The parallel program is a directed graph, where data flows on arcs connecting the nodes representing the sequential programs. The sequential programs may be written in any language, and CODE will produce parallel programs for a variety of architectures, as its model is architecture-independent.
http://www.cs.utexas.edu/users/code/

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

COM+: COM+ will provide a framework that will allow developers to build transaction-based distributed applications from COM components, managed from a single point of administration. COM+ offers higher level, more user friendly API's in which COM+ classes are handled in a similar way as the regular (local) C++ classes and the whole formalism shares several design features with the Java language model.

Component : an object which, can be modified and reused without prior knowledge of the Implementation details.

Component Model : the latest tidal wave in Object Oriented Programming which facilitates the reuse of objects without exposing any implementation details whatsoever to the end-user or programmer.

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

Computational Grid: a recent term used by the HPCC community to describe large scale distributed computing which draws on analogies with electrical power grids as enabling a revolution

Controls: Microsoft technology term for paramaterized and embeddable software entiries; this approach started from OLE controls that enabled inter-application automation within the MS Office suite, and evolved towards ActiveX controls, acting as Internet Explorer plug-ins.

CORBA: Common Object Request Broker Architecture. An approach to cross-platform cross-language distributed object developed by a broad industry group, the OMG. CORBA specifies basic services (such as naming, trading, persistence) the protocol IIOP used by communicating ORBS, and is developing higher level facilities which are object architectures for specialized domains such as banking. (fig. 6)

CORBA Facility: CORBA Facilities, occupy the conceptual void between the enabling technology defined by CORBA and Object Services, and the application-specific services, that the OMA labels "Application Objects".

CORBA Horizontal Facility: these Facilities are shared by many or most systems. There are four major sets of these facilities: User Interface, Information Management, Systems Management and Task Management.

CORBA Vertical Facility: supports domain-specific tasks that are associated with vertical market segments.

CORBA Services: while the ORB specifies a systems grammar, Object Services represent its most basic vocabulary; the essential interfaces needed to create an object, intoduce it into its environment, use and modify its features , and so forth. These services, bundled with every ORB, constitute the basic enabling technology of an OMA-compliant software system.

Crossware: Netscape term for the next generation software, based on multi-tier Web architecture and IIOP software bus and JavaBeans based publish-subscribe connectivity.

Cumulvs: visual front-end and PVM based communication library developed at Oak Ridge National Lab that supports computational steering and coordinating data displays from multiple viewers in a distributed computation.

DARP: *Data Analysis and Rapid Prototyping* – Java applet based front-end for HPF applications developed by Tom Haupt at NPAC that adds HPF interpreter capabilities to commercial HPF compilers and adds interacting scripting and rapid prototyping suppor to high performance software development environments.

DataChannel RIO: a two-way Corporate Portal infrastructure for corporate Intranets and Extranets making them easier to use, integrate, manage, and support.

Data Mining: describes the search and extraction of unexpected information from large databases. In a database of credit card transactions, conventional database search will generate monthly statements for each customer. Data mining will discover using ingenious algorithms, a linked set of records corresponding to fraudulent activity.

DcciS: *Distributed commodity computing and information System* - a software environment of unprecedented quality and functionality produced by industry and the loosely organized worldwide collection of (freeware) programmers.

DCE: the OSF Distributed Computing Environment is a comprehensive, integrated set of services that supports the development, use and maintenance of distributed applications. It provides a uniform set of services, anywhere in the network, enabling applications to utilize the power of a heterogeneous network of computers, http://www.osf.org/dce

DirectPlay: a component of Microsoft DirectX multimedia architecture, offering a high level, network protocol independent API for multiuser networking applications such as online multiplayer gaming.

DirectX: new high performance COM based multimedia architecture from Microsoft, capable of exploiting both HAL and HEL kernel layers; includes several modules such as DirectDraw for 2D sprites, Direct3D for 3D rendering, DirectInput for mouse and joystick control, DirectSound for 3D sound, and DirectPlay for the multiplayer game networking.

DIS: Distributed Interactive Simulation - original framework developed in such projects as SIMNET, to support FMS and IMT applications. HLA and RTI are superceding DIS.

Distributed Computing: the use of networked heterogeneous computers to solve a single problem. The nodes (individual computers) are typically tightly coupled.

DLL: *Dynamically Linked Library* – a software library module that links dynamically i.e. during runtime to the main program executable module and hence it can be shared rather than replicated among multiple concurrently executing programs.

DMSO: *Defense Modeling and Simulation Office* – a DoD agency responsible for advancing modeling and simulation technologies of relevance for the warfighter. Major DMSO efforts include HLA, CMMS and more recently CEDRIS and SBA.

DOM: Document Object Model - an application programming interface (API) for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated. With the Document Object Model, programmers can build documents, navigate their structure, and add, modify, or delete elements and content.

DOT: *Distributed Object Technologies* – a spectrum of remotely accessible object technologies, recently exploding on the Internet. Currently most popular models include Java, CORBA, COM and WOM; see also POW for a possible integration framework.

DSI: *Defense Simulation Internet* – a dedicated DoD network to allow simulation systems at distant sites to engage each other. Hence, DSI is a dedicated WAN that is logically separated from the Internet traffic and it facilitates wide-area federations of LAN based simulations. Local area networks tied to DSI are connected via a T-1 line which allows high speed transfers of data.

DTD: *Document Type Definition* - usually a file (or several files to be used together) which contains a formal definition of a particular type of document. This sets out what names can be used for elements, where they may occur, and how they all fit together. DTD's let processors automatically parse a document and identify where every element comes and how they relate to each other.

EJB: *Enterprise Java Beans* - a cross - platform component architecture for the development and deployment of multi-tier, distributed, scalable, object-oriented Java applications.

Enterprise JavaBeans: see EJB.

Event: a software artifact used to represent some temporal characteristics of an executing program that require special processing or communication with other modules or programs.

Exception: an Event generated when the program encouters unexpected conditions and cannot continue execution but the termination mode is to be decided by the user or some other program.

Falcon: code name for Microsoft Message Queue Server, a new capability offered by Windows NT Enterprise Editon that guarantees reliable message delivery in distributed environment.

FMS: Forces Modeling and Simulation - a major class of DoD (Department of Defense) applications, which are often termed "war-games". FMS either involves virtual time event driven simulations or real-time simulations involving people and military vehicles or instruments in the loop. Typically FMS uses a geographically distributed set of computers to simulate a geographically distributed application.

Front Page: popular HTML editor and Web site authoring package by Microsoft, based on the previous product by Verrmeer Software.

GET Method: one of the request methods of the HTTP protocol with one argument given by a URL specified by the client; the server responds by returning the content of this URL to the requesting client.

GIOP: *General Inter-ORB Protocol* – a wire protocol for inter-ORB communication, introduced by CORBA to facilitate interoperability between independent ORB vendors. The most popular version of GIOP is the IIOP protocol.

Globus: a low level toolkit for building high performance distributed computing applications, developed by Ian Foster and Argonne and Ian Foster at Caltech/USC. Globus uses Nexus as the core communication layer and it offers a set of services such as Directory, Security, Resource Allocation and some Quality of Service support.

HAL: *Hardware Abstraction Layer* – a loadable kernel-mode module in Windows operating environment that provides the low-level interface to the hardware platform. Internal kernel components of Windows as well as user-written device drivers

maintain partability by calling the HAL routines rather then accessing directly the underlying hardware.

HEL: *Hardware Emulation Layer* – a Windows capability to support software emulators for hardware devices. From the user level API perspective, HEL and HAL offer identical interfaces. HEL is useful for testing the use of new hardware devices which are still under development – this use is anticipated especially in the area of interactive multimedia where vendors are expected to advertise new special devices via their downloadable HEL emulators.

HeNCE: *Heterogeneous Network Computing Environment* - an X-window based software environment designed to assist scientists in developing parallel programs that run on a network of computers. HeNCE provides the programmer with a high level abstraction for specifying parallelism. HeNCE is based on a parallel programming paradigm where an application program can be described by a graph. HeNCE graphs are variants of directed acyclic graphs, or DAGS. Nodes of the graph represent subroutines and the arcs represent data dependencies. Individual nodes are executed under PVM. HeNCE is composed of integrated graphical tools for creating, compiling, executing, and analyzing HeNCE programs.

HLA: High Level Architecture – distributed object based simulation interoperability architecture by DMSO, adopted as a DoD-wide technical standard and currently proposed by DMSO as a CORBA standard.

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

HPCC: *High Performance Computing and Communication* - originally a formal federal initiative but even after this ended in 1996, this term is used to describe the field devoted to solving large-scale problems with powerful computers and networks.

HPDC: *High Performance Distributed Computing* - The use of distributed networked computers to achieve high performance on a single problem, that is the computers are coordinated and synchronized to achieve a common goal.

HPF: *High Performance Fortran* - a language specification published in 1993 by experts in compiler writing and parallel computation, the aim of which is to define a set of directives which allow a Fortran 90 program to run effeciently on distributed memory machines.

HPJava: an initiative by ARPA Parallel Compiler Runtime Consortium to address the use of Java for high performance language compilers and interpreters.

HTTP: Hyper Text Transport Mechanism – a stateless transport protocol allowing control information and data to be transmitted between web clients and servers.

IDL: *Interface Description Language* – a special language used to specify API for distributed object or components that admit multi-language bindings. Both CORBA and OSF DCE offer (incompatible) IDL frameworks. DCE IDL was accepted by Microsoft, extended and renamed as MIDL (Microsoft IDL). See also WIDL for new XML based initiatives towards IDL specification.

IIOP: Internet Inter Orb Protocol - a stateful protocol allowing CORBA ORB's to communicate with each other, and transfer both the request for a desired service and the returned result.

IMPORT: *Integrated Persistent Objects and Relations Technology* – a high level fifth generation language (5GL)that offers built-in callback and publish / subscribe high level synchronization and routing constructs that are compiled or translated to low level discrete-event based simulation engines. IMPORT is expected to facilitate development of large scale, complex simulations with large number of tightly interacting objects. Current sequential IMPORT implementation uses Quick Threads and the Parallel IMPORT based on SPEEDES in under development as one of the FMS projects within the DoD HPC Modernization Program.

IMS: Instructional Management System – a suite of tools and educational standards under development by EDUCOM, based on core distributed object technologies of CORBA, COM, Java and Web, and

aimed at enabling teachers, learners, software developers, content providers, and other parties involved in the learning process to create, manage access and share the use of online learning materials and environments.

IMT: Integrated Modeling and Testing - a major class of DoD applications aimed at detailed simulation of the capabilities of military subsystems such as a new tank. IMT simulations tend to be similar to FMS but include more detail for the component being tested. IMT can involve real systems in the loop, and is of growing importance as DoD cuts back on expensive physical tests. Civilian analogues are car crash simulations used in the automobile industry and computational fluid dynamics simulations substituting for wind tunnels in the aeronautics industry.

Introspection: for those who are queasy about the idea of enforced naming conventions, explicit information about a class can be provided using the BeanInfo class. When a RAD Tool wants to find out about a JavaBean, it asks with the Introspector class by name, and if the matching BeanInfo is found the tool uses the names of the properties, events and methods defined inside that pre-packaged class.

Invocation: the process of performing a method call on a CORBA object, which can be done without knowledge of the object's location on the network. Static invocation, which uses a client stub for the invocation and a server skeleton for the service being invoked, is used when the interface of the object is known at compile time. If the interface is not known at compile time, dynamic invocation must be used.

IOR: *Interoperable Object Reference* - an ORB vendor independent representation of remote reference to a CORBA object.

IP: *Internet Protocol* - the network layer communication protocol used in the DARPA Internet. IP is responsible for host-to-host addressing and routing, packet forwarding, and packet fragmentation and reassembly.

I-WAY: an experimental high-performance ATM network linking dozens of the country's fastest computers and advanced visualization environments.

JacORB: an object request broker written in Java - a partial implementation of OMG's CORBA standard. JacORB is free and is published under GNU public Library License

Java: an object-oriented interpreted programming language frum Sun, suitable for Web development due to the built-in portable support for mobility, networking, multithreading and GUI

Java API and Frameworks: a collection of packages including Java foundation classes and offering Java langauge interface to Java virtual machine (JavaVM) A Java Framework such as JDBC defines the standard Java methods and interfaces for accessing a particular service such as databases, commerce, graphics, multimedia etc. Such frameworks are determined by consensus in each area and are mirrored in CORBA as Facilities

Java Applet API: applet package in the Java API, the first in the long series of new APIs.

Jager: a simple video game type HLA application, ,including a user-operated cannon and a set of enemy ships, and distributed by DMSO as part of their RTI release.

Java 1.0: the first official release of JavaVM and the associated Software Development Kit (SDK), published in early 1996 and offering all core Java packages such as *java*, *awt*, *net*, *io* and *util*.

Java 1.1: the first major upgrade of JavaVM and SDK, published in 1997 and adding functionality in several areas such as new, publish / subscribe based Event model, new AWT package, JDBC, JavaBeans and RMI.

Java 1.2: coming second major upgrade of JavaVM and SDK, expected in late 1998 or early 1999 and adding functionality in areas such as JavaIDL (support for CORBA), Java3D, JavaMedia and others.

Java Grande: a Java forum, formed by representatives from high performance labs, lead by Geoffrey Fox, and aimed at enabling Java support for scientific and engineering computing areas, currently dominated by Fortran, C and C++.

Java3D: Java API that offers high level platform independent support for 3D graphics at the level similar to SGI Open Inventor.

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

Java Blend: a development product from Sun that uses JavaTM technology to greatly simplify the complex process of building business applications that can access any database. Java Blend contains a tool and a runtime environment that simplifies the integration of Java objects with enterprise data.

Java Card: the Java Card specifications enable Java technology to run on smart cards and other devices with limited memory. The Java Card API allows applications written for one Java Card-enabled smart card platform, to run on any other Java Card-enabled platform.

Java Enterprise API: this offers connectivity to enterprise databases via the JDBC package, and to distributed objects with Java IDL interface to CORBA and via native JavaBeans model

Java IDL: this API provides support for defining remote CORBA interfaces in the IDL interface definition language, an industry standard by OMG. Java IDL includes an IDL-to-Java compiler and a lightweight ORB that supports IIOP.

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

JFC: *Java Foundation Classes* - this is core to the Java platform. The JFC extend the original Abstract Windowing Toolkit (AWT), by adding a comprehensive set of graphical user interface class libraries.

Java Grande: application of Java to Grande applications; Grandecomputers are of course compute engines used to execute Grande codes and the adjective can be used in other related ways

Java JDK: *Java Development Kit* - Java Development Kit, including Java API for the language foundation classes and the base software development tools such as Java compiler and the native class interface generator. Current release is JDK 1.1.5.

Java Media API: support for real-time interactive multimedia including Sound, Animation, 2D, 3D, Java Telephony, Speech and Collaboration.

Java Object Serialization: allows Java programs to serialize objects into a stream of bytes that can be later used to build equivalent objects for the same or remote JavaVM.

JavaOM: *Java Object Model* - our name, constructed for the purpose of this presentation, to refer to a mix and blend of Java technologies involved in distributed object computing; hence, JavaOM includes RMI, JavaIDL, JINI and JavaSpaces.

JavaOS: Java Operating System - highly compact operating system designed to run Java applications directly on microprocessors in anything from net computers to pagers.

Java RMI: *Remote Method Invocation* - support for creating objects whose methods can be invoked from another virtual machine, analogous to a remote procedure call (RPC). This can be looked upon as a native ORB.

JavaScript: Java-like but typeless and truly interpreted scripted language developed by Netscape and used for dynamic HTML programming both at the client (Navigator) and server (LiverWire) sides.

Java Security API: a framework for developers to easily and securely include security functionality in their applets and applications. Functionality encompasses including cryptography, digital signatures, encryption and authentication. .

Java Servlet API: support for building customized server solutions in terms of servlets - small executable programs that users upload to run on networks or servers.

JavaSpaces: distributed persistence and data exchange mechanisms for code written in the Java programming language. Facilitates the development of Systems that use flow of data to implement distributed algorithms.

Java Standard Extension API: a standardized framework for extending the core Java API by domain specific packages.

JavaVM: an abstract computer that runs compiled Java programs. Implemented in C as an interpreter of the opcode generated by Java compiler.

Java WorkShop: Java CASE toolkit from SunSoft, includes Visual Java support for visual software engineering.

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

JDCE: *Java Distributed Collaborative Environment* - a RMI/CORBA based collaboration environment developed at NPAC.

Jfactory: visual interface builder for Java applications and applets; supports click-and-drag authoring, automated code generation and user level extensibility.

Jigsaw: Web server written entirely in Java, recently published by the WWW Consortium, and to be used as the experimentation platform with mobile code, scripting languages, object-oriented Web etc.

JINI: Sun's new effort towards a more complete distributed system, aimed at the emerging market of networked consumer electronic devices. The overall goal is to turn the network into a flexible, easily administered tool on which human or computational clients can find resources efficiently.

JIT: *Just-In-Time Compiler* - an Interpreter which, "compiles on the fly" by "remembering" the native machine code produced when bytecodes are first seen. This can produce major (at least a factor of 10) improvement in performance of Java Applets when used in a browser and when code is used more than once as it is in any iteration. Symantec produced the first well known JIT for Java under Windows95.

JNDI: Java Naming and Directory Interface - another product from JavaSoft which allows developers to deliver Java applications with unified access to multiple naming and directory services across the enterprise.

JRMP: *Java Remote Method Protocol* - internal protocol used by Java Virtual Machine to support RMI. A version of RMI based on open IIOP protocol of CORBA will be also provided in the coming releases of JDK.

JSDA: the Java Shared Data API (JSDA) defines a multipoint data delivery service for use in support of highly interactive, collaborative, multimedia applications.

JWORB: *Java Web Object Request Broker* - a multi-protocol network server written in Java. Currently, JWORB supports HTTP and IIOP protocols, i.e. it can act as a Web server and as a CORBA broker or server. Base architecture of JWORB can be represented as a protocol detector plus a collection of dedicated servers for the individual protocols.

Khoros: object-based, UNIX software that provides a broad range of functionality in a variety of domains such as image processing, medical imaging, signal processing, and scientific visualization. The Khoros visual programming environment assists quick prototyping, without requiring a single line of code to be written. Khoros' rich set of CASE tools work with this visual programming environment to create a powerful integration environment that moves legacy code and customized software into a consistent, standards-based framework with ease. Khoros system was developed by University of New Mexico and is now commercialized by Khoral Research, Inc.

LCU: *Language Connect University* - another Web/Oracle service constructed by Translet Inc. for the distance education community. Marketed by the Syracuse Language Systems as an Internet extension of their successful CD-ROM based multimedia courses for several foreign languages, LCU offers a spectrum of collaboratory and management tools for students and faculty of a Virtual University.

Legion: an object-based, meta-systems software project led by Andrew Grimshaw at the University of Virginia. This scalable system is founded on solid principles, guided by in-house work in object-oriented parallel processing, distributed computing, security, and distributed computing systems. Legion system addresses key issues such as scalability, programming ease, fault tolerance, security, site autonomy, etc. Legion is designed to support large degrees of parallelism in application code and manage the complexities of the physical system for the user.
http://legion.virginia.edu/

Little Language: an application or domain specific macro language, typically used to set application specific variables, control execution modules, specify data or display configurations etc.

LiveWire: JavaScript based server-side scripting technology developed by Netscape; offers scripted support for server-side-include operations including Web linked databaseas; the corresponding Microsoft approach is given by the Active Server Pages technology.

Lotus Notes: a major commercial software system supporting aspects of collaboration but at its heart a web-linked document database providing the type of tools also seen in systems such as Cold Fusion for producing with high level tools general Web Interfaces to database material.

Marshalling: an intermediate step of remote method execution in which the method invoking machine marshalls the method arguments, i.e. converts them to a serialized byte stream, sends it to the network, and the method executing machine unmarshalls the steam and passes the arguments to the local method for execution.

MBONE: *Multicast Backbone* – a virtual network based on IETF specification and used by several hundred researchers for developing protocols and applications for group communication on the Internet.

Metacomputing: a computing paradigm involving a spectrum of wide-area distributed high performance systems connected via high bandwidth networks.

Metadata: a set of data constructs specifying the structure of the data used by a given system. A system including a metadata component is often referred to as self-describing or introspecting.

Metaproblems: advanced, complex computational and communication problems that require the use or map naturally on metacomputing environments. Examples include modeling world climate or world economy, large scale wargaming, crisis mangement, Internet virtual communities.

Microsoft Access: low end single user affordable SQL database with a set of form based tools offered by Microsoft as part of the MS Office environment.

Microsoft ActiveVRML: Microsoft proposal for VRML 2.0 that lost with the Moving Worlds proposal from SGI and partners but is being nevertheless implemented by Microsoft.

Microsoft ActiveX: previous OLE framework, now reformulated as a content development and media integration environment for the Internet. Includes tools for visual programming and integration.

Microsoft ActiveX Conferencing: a suite of technologies that enable real-time, multiparty, multimedia communication over the Internet.

Microsoft ActiveX Controls: formerly known as OLE controls, are components (or objects) that can act in a similar way as Netscape plug-ins or Java applets in the Microsoft software/database domain.

Microsoft ActiveX Development Kit: includes, an evolving collection of technologies that Microsoft is experimenting with and introducing to facilitate development of Internet applications and content.

Microsoft DirectPlay: a component of DirectX that provides networking and communication services in a transport independent manner. DirectPlay provides a consistent interface for the developer irrespective of whether the transport mechanism is TCP/IP, IPX or modem.

Microsoft Direct X: a set of APIs for developing powerful graphics, sound and network play applications. The biggest incentive that DirectX offers developers is a consistent interface to devices across different hardware platforms.

Microsoft Jakarta: codename for JavaVM for Windows and visual Java development tools from Microsoft .

Microsoft Internet Explorer: a Web browser from Microsoft, competing with Netscape and offering a

set of extensions such as a) COM based plug-in capabilities referred to as ActiveX; b) support for DirectX graphics (currently via DirectAnimation, soon via XML based Chromeffects); c) support for push technology via the XML based CDF.

Microsoft NetMeeting: a collaboratory system from Microsoft, supporting point-to-point video and audio teleconferencing, chat, whiteboard and application sharing; coming NT version 5 includes built-in extended version of NetMeeting with ActiveMovie based multi-party video streaming support.

Millenium: next generation distributed operating system, currently being addressed by Microsoft Research. Key design guidelines are: total rethinking of existing paradigms, aggressive abstraction, seamless Web computing support. Some Millenium subsystems include: Borg, Coign and Continuum.

ModSAF: *Modular Semi-Automated Forces* – a large scale DIS simulation system for the Army developed by STRICOM and supporting a broad range of simulation functions and databases; includes support for terrain, vehicle, troops, their communication, aggregation, autonomous behavior, battlefield operations etc. ModSAF is written in C, packaged as ~500 libraries of a total size of 1.5 millions lines of C code.

MPI: *Message Passing Interface* - the parallel programming community organized an effort to standardize the communication subroutine libraries used for programming on massively parallel computers such as Intel's Paragon, Cray's T3D, as well as networks of workstations. MPI not only unifies within a common framework programs written in a variety of exiting (and currently incompatible) parallel languages but allows for future portability of programs between machines.

MIME: *Multipurpose Internet Mail Extension* - the format used in sending multimedia messages between *Web Clients* and *Servers* that is borrowed from that defined for electronic mail.

MPP: *Massively Parallel Processing* – a high performance computing technology based on large number tightly coupled processing units, performing concurrently the dedicated components of the whole computation and communicating through an optimized, high bandwidth communication mesh.

MTS: *Microsoft Transaction Server* – a middleware technology from Microsoft (code name Viper) represented as a COM server that offers management services for other, finer grain COM components; services include multithreading, security, transaction support.

NCSA Habanero: a framework for constructing distributed collaborative software tools. Using Habanero, programs that were designed for a single-user can be relatively easily recast as multi-user collaborative tools.

NDR: Network Data Representation – a byte stream format used by the OSF DCE RPC, adopted as the networking base for Microsoft DCOM.

Netscape Client Pull: Netscape extension of HTML which, allows servers to put the Navigator in an active pull mode, i.e. to instruct the Navigator to reload an indicated URL after a specified timeout.

Netscape Community System: Web based community network environment, including support for bulletin boards, real-time chat sessions, electronic mail, and private discussion groups.

Netscape Communications Server: high-performance server software that enables organizations to publish rich hypermedia documents (standard Netscape Web server before the current series of new server products).

Netscape Cookies: a "cookie" is a small piece of information which a web server (via a CGI script) can store with a web browser and later read back from that browser. This is useful for having the browser remember some specific information across several pages.

Netscape Enterprise Server: high performance, secure WWW server for HTML delivery, content management and portable access to database management systems.

Netscape JRI: an abstract interface to Java services that allows application programs (Java service consumers) to be decoupled from the Java runtime internals (Java service producers).

Netscape LDAP: the Lightweight Directory Access Protocol (LDAP) is Netscape's strategic directory protocol. It defines a simple mechanism for Internet clients to query and manage an arbitrary database of hierarchical attribute/value pairs over a TCP/IP connection.

Nexus: a simple point-to-point channel based high performance communication library, used as the core communication layer of the Globus metacomputing toolkit.

NPAC: *Northeast Parallel Architectures Center* at Syracuse University is a research and development center led by Geoffrey Fox and focused on modern computation technologies for large-scale information and simulation applications.

NPAC WebWisdom: a system for Web based distance education, including courses on Web technologies and electronic presentation tools.

Object Reference: a construct containing the information needed to specify an object within an ORB. An object reference is used in method invocations to locate a CORBA object. Object references are the CORBA object equivalent to programming language-specific object pointers. They may be obtained from a factory object or from the Naming Service.

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

OLE: *Object Linking and Embedding* – Microsoft component technology that was initially motivated by the requirements of the object level interoperability between MS Office documents, and then evolved towards the COM component, now followed by all Microsoft products.

OLEDB: Microsoft COM interface that offers uniform transparent persistence abstraction on top of heterogeneous datastores, including flat file systems, relational and object databases.

OMA: *Object Management Architecture* – overall architecture for distreibuted objects, specified by OMG; includes object request brokers, common aobje3ct services, horizontal (generic) and vertical (domain specific) facilities.

OMG: Object Management Group - an organization of over 700 companies that is developing CORBA through a process of call for proposals and development of consensus standards.

OmniORB2: omniORB2 is a robust, high-performance CORBA 2 ORB, developed by Olivetti Research Laboratory.

Oracle: leading vendor of relational database management systems for the UNIX/Solaris platform.

OrbixWeb: a Java Object Request Broker from IONA technologies Inc.

ORBlet: a small ORB, operating inside a Web browser and turning it into a CORBA client or server. The concept was introduced by Netscape and implemented in terms of Visigenic technology (VisiBroker). An alternative to a resident ORBlet as in Netscape is a Java ORB downloadable as an applet by any Java-capable Web browser.

ORB: Object Request Broker - used in both clients and servers in CORBA to enable the remote access to objects. ORB's are available from many vendors and communicate via the IIOP protocol.

ORPC: *Object Remote Procedure Call* – Microsoft extension of DCE RPC, used for DCOM communication protocol.

OWRTI: *Object Web Run-Time Infrastructure* **-** an implementation of DMSO RTI 1.3 in Java as a set of CORBA objects; one such implementation was recenbtly constructed at NPAC on top of the JWORB environment.

Parallel Computing: a computational paradigm involving several processing units, operating concurrently and communicating the intermediate results.

PCRC: *Parallel Compiler Runtime Consortium* – a multi-university DARPA program led by NPAC with the goal of extracting and coordinating common infrastructure components present in several parallel compiler projects.

Perl: a very useful interpreted language developed originally for sophisticated UNIX Systems and Document handling. Very common language for building Web enhancements in CGI scripts.

Perl5: Perl5 has object-oriented features and many useful new features Many uses of PERL are being supplanted by Java.

Persistence.com

PL/SQL: a procedural programming language similar to C or Fortran but with built-in SQL commands, offered by Oracle for building versatile and dynamic programmatic interfaces to the Oracle database server.

POA: *Portable Object Adapter* **-** POA is far more scalable and portable than its predecessor BOA. POA supports objects with persistent identities besides providing a transparent object activation model.

POS: *Persistent Object Service* **–** initial specification of Persistence Service in CORBA which didn't gain broader accepance among competing RDBMS vendors; new ongoing initiatives in the area of CORBA databases include: Persistent State Service and Business Object Facility.

POST Method: one of the request formats of the HTTP protocol, used to send some data from client to the server; typical use includes sending the content of a fill-out form to the server with the associated URL that points to a CGI script to be used to process the form content.

POW: *Pragmatic Object Web* **-** in the real world, we have four competing middleware technologies: Java, CORBA, COM and WOM, and each of them offers or attempts at a different solution for the universal middleware data representation. POW provides an efficient integration framework for several major software trends.

POWVM: *Pragmatic Object Web Virtual Machine* **-** a mesh of JWORB servers running on various platforms and supported with a suite of services that facilitate formation of distributed computational surfaces, collaboratory spaces or federations of simulation modules.

PSS: *Persistent State Service* - a new initiative by the OMG, aimed at specifying a uniform persistence layer on top of variety of datastores, including relational and object databases.

PVM: *Parallel Virtual Machine* - a software system that enables a collection of heterogeneous computers to be used as a coherent and flexible concurrent computational resource. The PVM system transparently handles message routing, data conversion for incompatible architectures, and other tasks that are necessary for operation in a heterogeneous, network environment.

Python: an object-oriented interpreted language developed by Guido van Rossum , initially as a component of the Amoeba operating system, and more recently gaining some popularity for Internet programming. Some people prefer Python than Java since Python interpreter code is in public domain.

Rational Rose: a visual tool for object-oriented software analysis and design, developed by Rational and based on the UML standard.

RDF: Resource Definition Format - designed to provide an infrastructure to support metadata across many web-based activities. RDF is the result of a number of metadata communities bringing together their needs to provide a robust and flexible architecture for supporting metadata on the Internet and WWW.

Reflection: because of Java's late binding features, a Java .class file contains the class's symbol information and method signatures, and can be scanned by a development tool to gather information about the class.

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

RPC: *Remote Procedure Call* – a communication technique in which a program on one machine executes a procedure on another machine, using some suitable marshalling method to communicate procedure arguments and return results. RPC environments were difficult to maintain and they are being now replaced by the distributed object technologies which offer a natural encapsulation framework for the remote method bookkeeping.

RTI: Run Time Infrastructure - Run time defined to support HLA compliant simulations including "federated" interoperable simulations.

RTP: a thin protocol providing support for applications with real-time properties, including timing reconstruction, loss detection, security and content identification.

Sandbox: Java applets run in a womb provided by the web browser that offers them services, and prevents them from doing anything naughty such as file i/o or talking to strangers (servers other than the one the applet was loaded from). The analogy of

applets being like children lead to calling the environment they run in the "sandbox". See womb, applet.

SBA: *Simulation Based Acquisition* – a new trend in the DoD modeling and simulation with the ultimate goal of providing virtual prototyping, test and evaluation tools for the full product development and acquisition cycle. Virtual prototyping environments are already successfully used by Boeing and the Automotive industry. Emergent DoD SBA environments are based on open standards such as HLA, CMMS, CEDRIS etc. and are likely to enforce cooperation and interoperability with industry.

Serialization: a data conversion and communication technique in which a non-sequential software element such as a method, attribute or the whole object is represented as a sequence of bytes, sent to remote location, and converted there back to a non-serial internal representation.

Server: this is usually a program which listens to a specific port, and accepts (processes) invocations from objects at remote locations.

Server Skeleton: a public abstract class generated by the idltojava compiler that provides the ORB with information it needs in dispatching method invocations to the servant object(s). A server skeleton, like a client stub, is specific to the IDL interface from which it is generated. A server skeleton is the server side analog to a client stub, and these two classes are used by ORBs in static invocation.

Servlet: a component of the Java Web Server (formerly known as Jeeves) that facilitates creation of executable programs on the server side of an HTTP connection.

SET: *Secure Electronic Transaction* - a single technical standard for safeguarding payment card purchases made over open networks. SET specification, developed by Visa and MasterCard, includes digital certificates - a way of verifying the actual cardholder is making the purchase - and will provide financial institutions, merchants, and vendors with a new and safe way of doing transactions on the emerging electronic commerce marketplace.

SGML: *Standard Genralized Markup Language* - the international standard (ISO 8879:1986) for defining descriptions of the structure and content of different types of electronic document. Essentially, SGML is a method for creating interchangeable, structured documents.

SIW: *Simulation Interoperability Workshop* – bi-annual worksop in the area of advanced DoD Modeling and Simulation, organized by SISO (Simulation Internoperability Standard Organization) and focused on hot topics targeted by DMSO such as HLA, CEDRIS or SBA.

SOAP: *Simple Object Access Protocol* – new XML based protocol by Microsoft, initially focused on portable object level communication between various Windows platforms but also recommended for other distributed object technologies such as CORBA or Java.

Socket: tool for network communications, specifies end-points of a communication. Sockets identify the origin and destination of messages.

SPEEDES: *Synchronous Parallel Environment for Emulation and Discrete Event Simulation* – a parallel simulation kernel, supporting optimistic (rollbackable) synchronization based on the Breathing Time Warp algorithm. Individual nodes of a parallel SPEEDES simulation advance independently in time, taking controllable risk that some of their computation will need to be discarded / rolled back, and then they synchronize periodically by collective computation of the Global Virtual Time.

SQL: Structured Query Language - the language most commonly used to create database queries.

SSL: *Secure Socket Layer* - developed by Netscape and used as their base security enforcing technology.

Stringified Object Reference: an object reference that has been converted to a string so that it may be stored on disk in a text file (or stored in some other manner). Such strings should be treated as opaque because they are ORB-implementation independent. Standard object_to_string and string_to_object methods on org.omg.CORBA.Object make stringified references available to all CORBA Objects.

Synchronized: Java's Abstraction of a mutual exclusion lock. This prevents multiple threads from accessing the same piece of data, thus avoiding deadlocks.

Syracuse Language Systems: a software compnay in Syracuse, NY area, developing CD-ROM multimedia products for learning foreign languages; SLS collaborates with NPAC spin-off Translet in the area of database and middleware support for virtual university management.

TAO: C++ ORB that is performance optimized for real time applications.

TCP: *Transmission Control Protocol* **-** a connection-oriented protocol used in the DARPA Internet. TCP provides for reliable transfer of data, as well as out-of-band indication of urgent data.

TEMPO/Thema: a parallel simulation engine based on optimistic time management strategy similar to Time Warp under development by SAIC; this sytem shares common features with SPEEDES but it is currently supported only for shared memory architectures.

UDA: *Universal Data Access* **–** Microsoft approach to data access and management based on OLEDB transparent persistency layer.

UML: *Uniform Modeling Language* **–** new visual language for object-oriented development by Rational, adopted as standard by OMG. Sample applications include Rational Rose by Rational, Visual Modeller by Microsoft and Together/J by Object International.

VBA: *Visual Basic for Applications* **–** a VB scripting framework for Microsoft applications, offering VB API for COM interfaces, published by a given application.

VIA: *Virtual Interface Architecture* **–** high performance messaging architecture for applications that reside on servers connected into a SAN (System Area Network). VIA achieves top performance since it bypasses OS kernel during the communication process. VIA was developed jointly by Microsoft, Intel and Compaq.

VAT: public domain real-time Internet audio conferencing tool from LBNL .

VIC: public domain real-time Internet video conferencing tool from LBNL.

Viper: code name for Microsoft Transaction Server (MTS).

Visual Basic: programming language (VB), script (VBS) and macro language (VBA) developed by Microsoft and used for variety of purposes such as GUI scripting, Web server side scripting, application specific little/macro languages etc.

Visual Java: Visual Java authoring support in the recent release of Java Workshop, a Java CASE toolkit from SunSoft.

VM: *Virtual Machine* **–** a software program emulating a hardware processing unit, typically organized as an interpreter of a virtual assember code.

VRML 1.0: *Virtual reality Markup Language 1.0* **-** a standard network protocol for 3D scene description based on a subset of the ASCII Open Inventor format.

VRML 2.0: extends VRML 1.0 towards interactive 3D worlds by adding Sensor nodes that detect user events, routes that connect user events with the scene graph nodes, and Script nodes that describe interactive behavior of the scene graph objects using a scripting language such as Java, JavaScript, Tcl etc.

VRMLScript Custom script for VRML 2.0 under consideration by the VRML forum.

W3C: *World Wide Web Consortium* **-** the W3C was founded in October 1994 to lead the World Wide Web to its full potential by developing common protocols that promote its evolution and ensure its interoperability. We are an international industry consortium, jointly hosted by the Massachusetts Institute of Technology Laboratory for Computer Science [MIT/LCS] in the United States; the [INRIA] in Europe; and the Keio University Shonan Fujisawa Campus in Japan.

Wargame2000: a large scale simulation of space warfare based on SPEEDES under development by Joint National Test Facility at Colorado Springs, CO.

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

WebCORBA: recent initiative by W3C to integrate CORBA with the Web.

WebFlow: a high-level programming environment developed by NPAC - that addresses visual componentware programming issues and offers a user friendly visual graph authoring metaphor for seamless composition of world-wide distributed high performance dataflow applications from reusable computational modules.

WIDL: *Web Interface Definition Language* **- e**nables automation of all interactions with HTML/XML documents and forms, to provide a general method of representing request/response interactions over standard Web protocols, and to allow the Web to be utilized as a universal integration platform.

WebHLA: *Web High Level Architecture* **-** HLA standard specification leaves several implementation decisions open and to be made by the application developers - this often leaves developers of new simulations without enough guidance. In WebHLA, we fill this gap by using the emergent standards of Web based distributed computing – we call it *Pragmatic Object Web* - that integrate Java, CORBA, COM and W3C WOM models for distributed componentware.

Web linked Database: a set of interactive Web pages offering dynamic interfaces to relational databases in terms of HTML forms or Java applets and one of several middleware techniques such as CGI, JDBC, ColdFusion, LiveWire, ASP etc.

WebObjects: Web server extension technology from NeXT, including WebScript scripting language, a collection of productivity tools, and interfaces to database management systems.

WebRTI: see OWRTI.

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

WebWindows: our term used to refer to Web based operating environment under collective development by the Web community. DcciS and Pragmatic Object Web discussed here can be viewed as specific current instances of this concept.

Wolfpack: code name for the Microsoft Cluster Server.

WOM: *Web Object Model* **-** 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.

XML: eXxtensible Markup Language - is a simple and very flexible language based on SGML. Although originally envisaged to meet the challenges involved in large-scale publishing, XML is set to play an increasingly important role in the markup of a wide variety of data on the Web.