From Javabean to Database Web Servers in the Pragmatic Object Web

Geoffrey C. Fox, Wojtek Furmanski and Hasan T. Ozdemir
Northeast Parallel Architectures Center, Syracuse University, Syracuse NY
{gcf, furm, timucin}@npac.syr.edu

1: Introduction

We believe that industry and the loosely organized worldwide collection of commercial, academic and freeware programmers is developing a remarkable new software environment of unprecedented quality and functionality. We can call this DcciS - Distributed commodity computing and information System. Perhaps a little while ago it was clear what a web server is but today as DcciS grows in sophistication, this is not so obvious. Originally web servers largely served documents (HTML, JPEG, VRML etc.) but now their functionality and role in a distributed system is potentially much richer. Always web servers had the CGI (Common Gateway Interface) mechanism, which allowed them to invoke arbitrary programs and act as a link to general services. However DcciS has generalized this to the invocation of distributed objects with cleaner interfaces as exemplified by CORBA's IDL. In general we see a rapid merging of web and distributed object technologies to produce a powerful infrastructure which one can term the object web. This development joins many previously disjoint communities and indeed there are several approaches, which must be reconciled before the structure of the object web becomes clear. In particular there are four significant commodity object technologies -- CORBA, COM(DCOM), Javabeans as well as the efforts of W3C (World Wide Web Consortium) to develop a Web Object Model (WOM). Each of these approaches has important strengths and we can expect commercial standards to incorporate aspects of all four. A recent OMG/DARPA workshop on compositional software architectures [7] 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 complete solution.

As consensus and technical progress leads to the merged standards, we refer to the ongoing confusing amalgam as the "Pragmatic Object Web". In spite of the confusion we believe that the power of the current technologies is so great that they can already build systems which offer impressive capabilities and therefore the "Pragmatic Object Web" will often be the best choice when building large scale distributed systems. We illustrate these ideas with a description of some NPAC activities built around the use of these commodity systems for large scale high performance computing (HPCC). As described elsewhere [1,8,12], we believe that HPCC can benefit greatly by leveraging the impressive DcciS software infrastructure. Here we focus on the use of a novel Java Web Server JWORB [9] which combines different object web capabilities (in particular directly it serves as both a classic web HTTP and CORBA server). We describe its initial performance characteristics and use in distributed computing projects with a visual front-end Webflow [2-5].

2: Commodity Technologies and their use in Multi-Tier Systems

The last four 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 WOM, COM and Javabeans to be the core distributed object technology.

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

In each commodity technology area, we have impressive and rapidly improving software artifacts. As examples, we have at the lower level the collection of standards and tools such as HTML, HTTP, MIME, IIOP, CGI, Java, JavaScript, Javabeans, CORBA, COM, ActiveX, VRML, new powerful object brokers (ORB's), dynamic Java servers and clients including applets and servlets. The new W3C base technologies include 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.

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

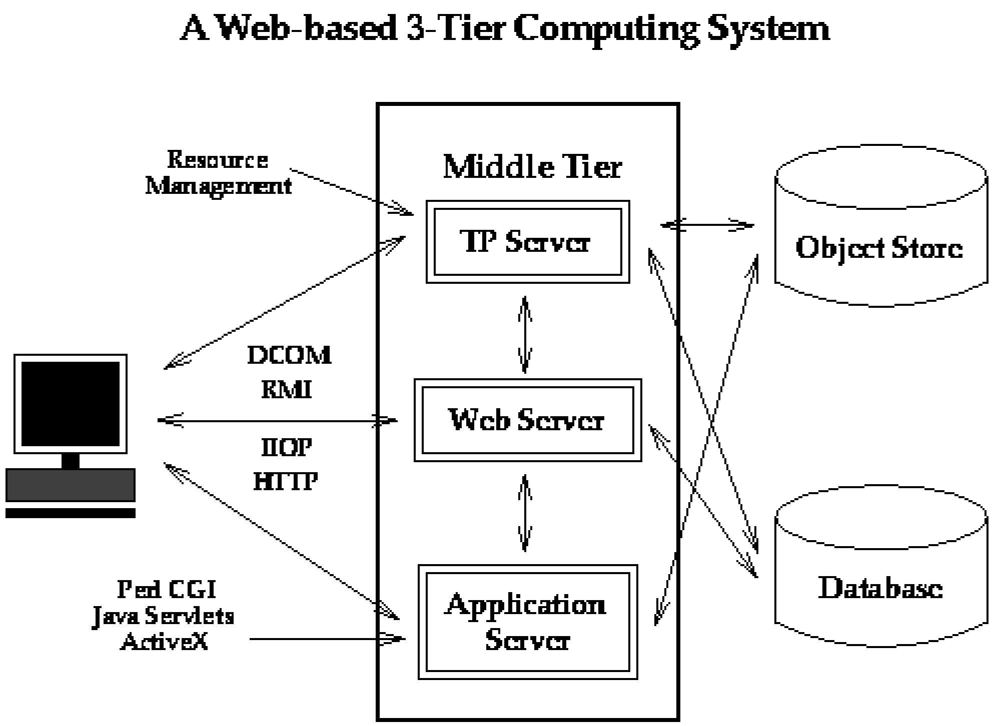

Fig. 1: Industry 3-tier view of enterprise Computing

We start with a common modern industry view of commodity computing with the three tiers shown in fig 1. Here we have customizable client and middle tier systems accessing "traditional" back end services such as relational and object databases. A set of standard interfaces allows a rich set of custom applications to be built with appropriate client and middleware software. As indicated on figure, both these two layers can use web technology such as Java and Javabeans, distributed objects with CORBA and standard interfaces such as JDBC (Java Database Connectivity). There are of course no rigid solutions and one can get "traditional" client server solutions by collapsing two of the layers together. For instance with database access, one gets a two tier solution by either incorporating custom code into the "thick" client or in analogy to Oracle's PL/SQL, compile the customized database access code for better performance and incorporate the compiled code with the back end server. The latter like the general 3-tier solution, supports "thin" clients such as the currently popular network computer.

As discussed in the previous section, 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 which are absorbed by the web in the simplest way:

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

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

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

As an example consider the evolution of networked databases. Originally these were client-server with a proprietary network access protocols shown in fig. 2 below.

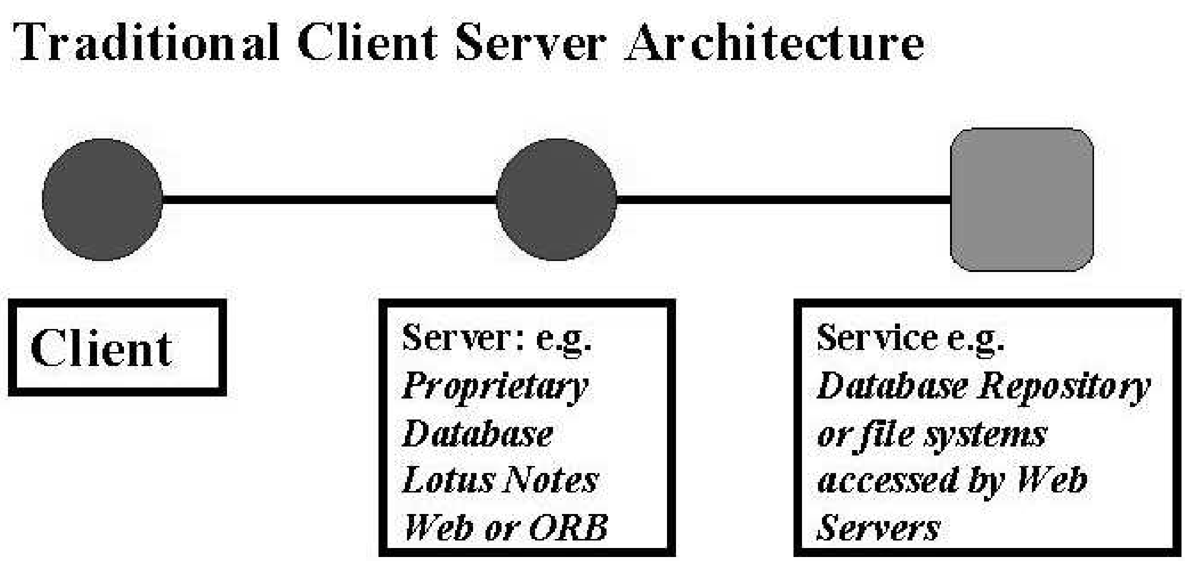

Fig. 2: Classic Client Server Architecture

Web linked databases and enterprise Intranets naturally produced a three tier distributed service model illustrated in fig. 3, 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. Note fig 3 shows how middle tier business logic is responsible for mapping a user-side object model into the available repository – here a relational database.

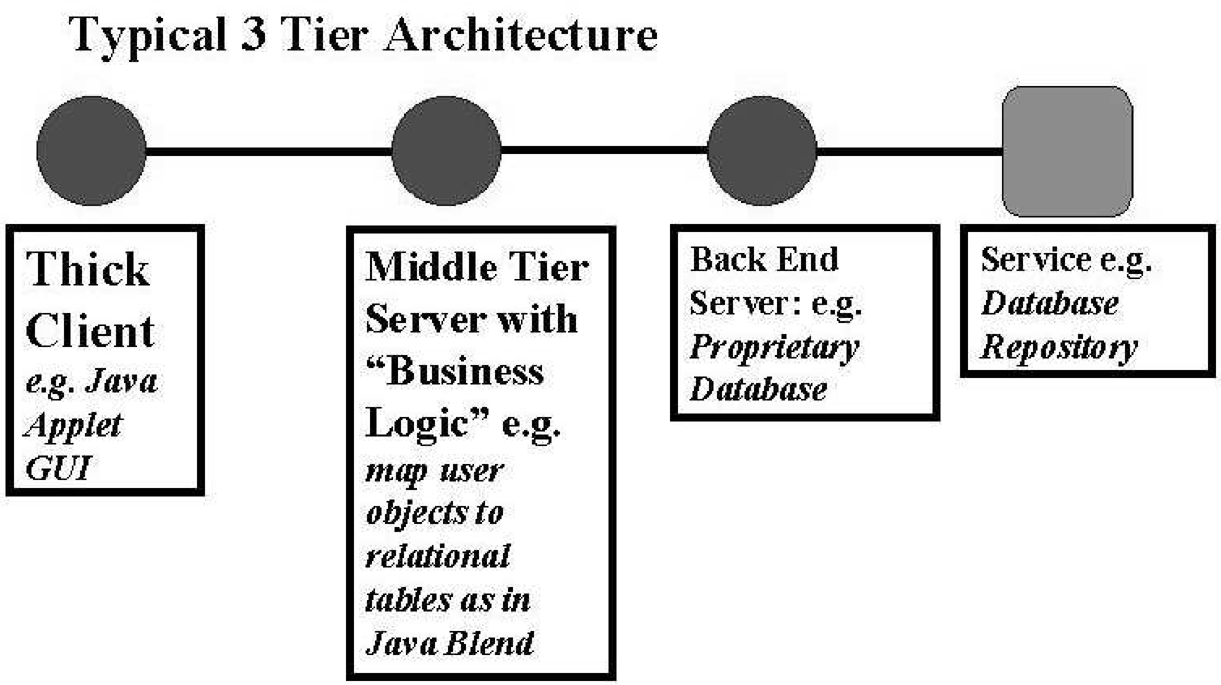

Fig. 3: 3 Tier model for a web-linked (object) database

As shown in fig. 4, we see today the "Pragmatic Object Web" 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.

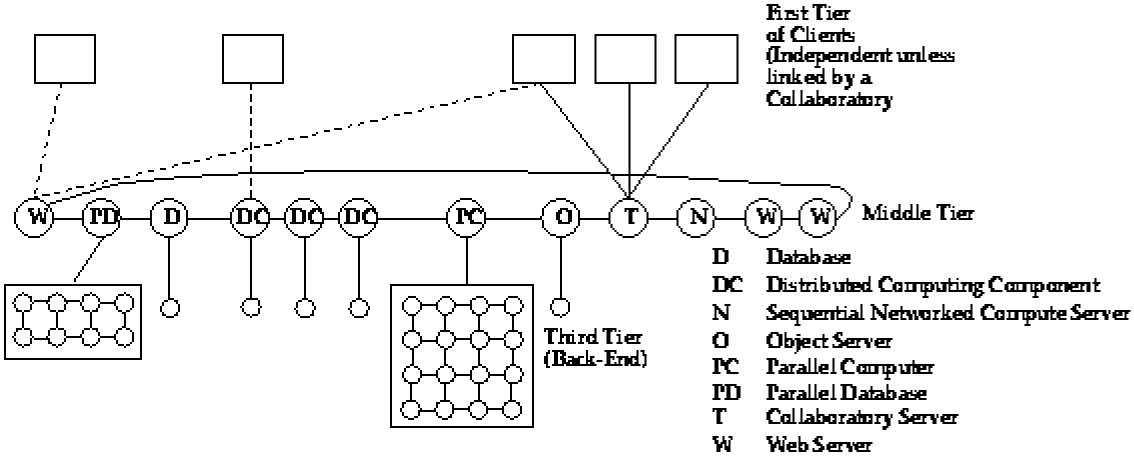

Fig. 4: Today's Heterogeneous Interoperating Hybrid Server Architecture. NPAC's HPCC strategy involves adding to this system, high performance in the third tier.

In fact, the middle tier is a "server-server" model with multiple servers cooperating to solve a single problem. One example is the 4 tier architecture designed to support "thin clients" shown in fig. 5.

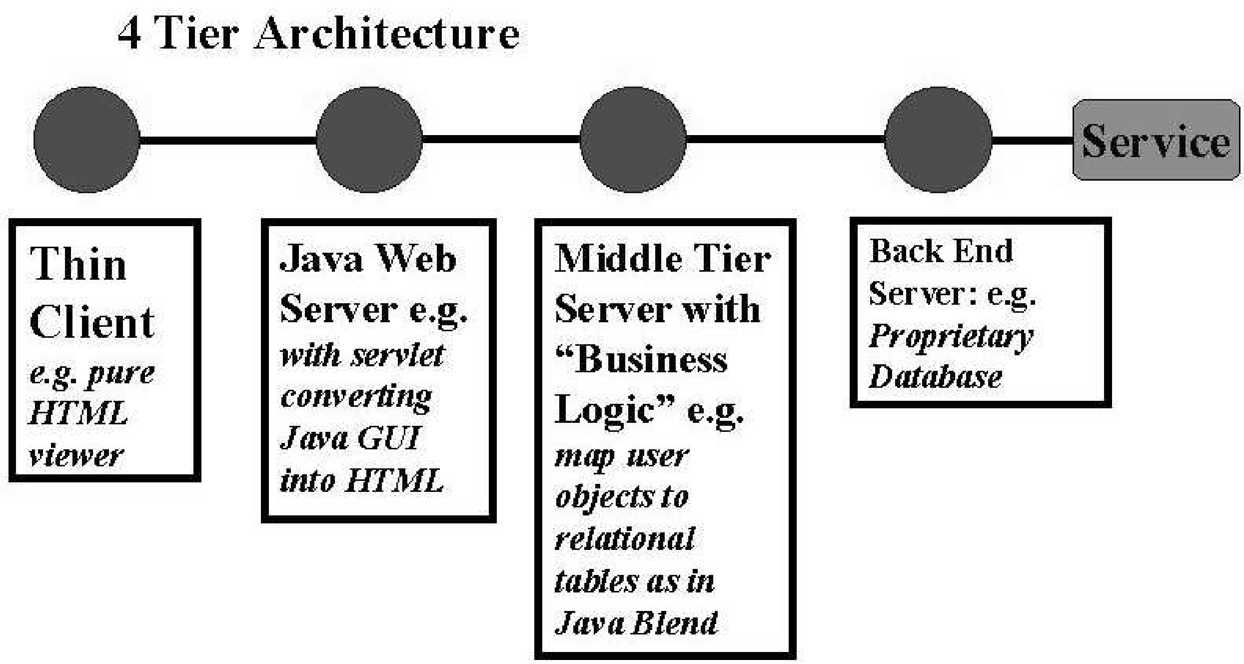

Fig. 5: A 4 tier architecture

Further suppose there is a small object broker (a so-called ORBlet) in each browser as in Netscape's current systems. We then get a “sea” of middle tier servers connected to client and backend systems which themselves can have servers. The resultant system shown in fig. 6 shows several web and other servers which are the fixed resources which are linked by data or program (as in agents) flow to execute a given task.

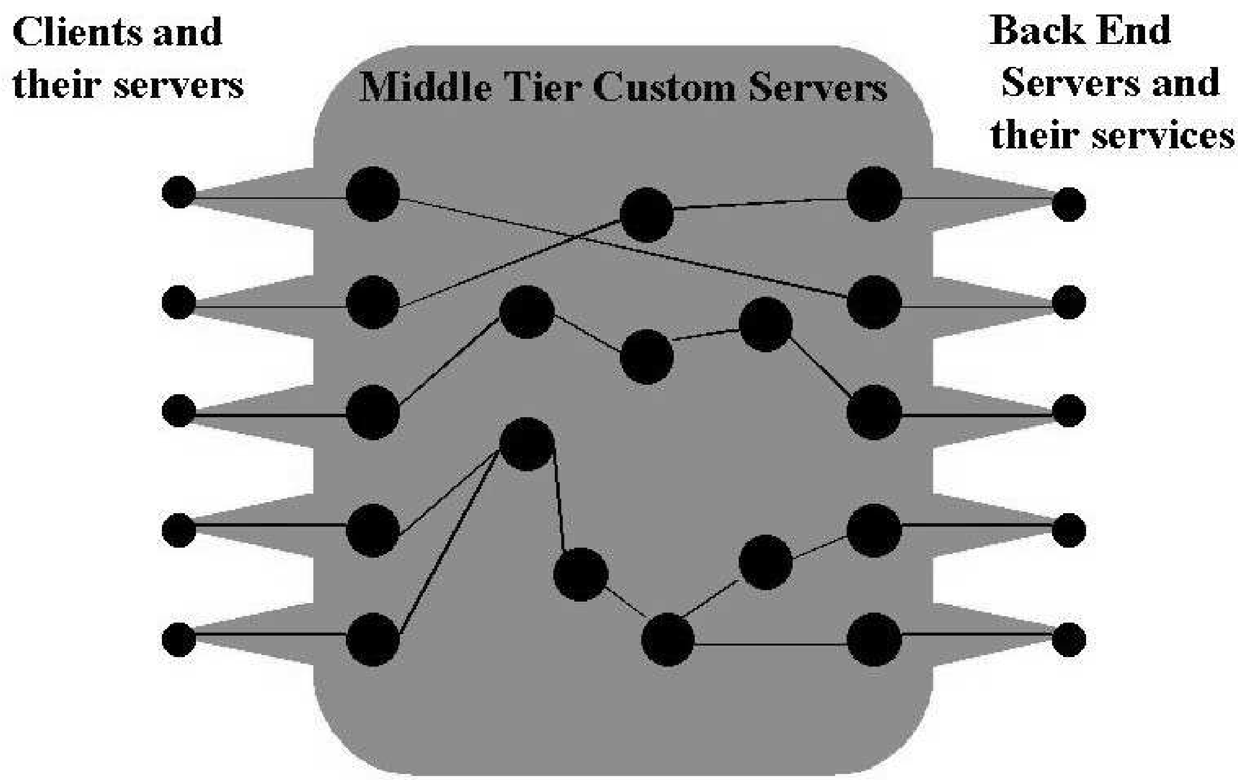

Fig 6: A sea of web and object servers linking client side personal servers to specialized back-end servers.

Thus our generalized object web servers are the key building blocks. We hypothesize that it will be very helpful to build systems so that a given server can be a truly pragmatic server and be able to link all the approaches together in a single resource. Then we can link components from the different communities and build the Pragmatic object web by picking and choosing the best of each approach. This is the motivation behind JWORB, which is described below.

3: WebFlow Current State

We first describe a system WebFlow that we have developed over the last three years and which illustrates well the new opportunities offered by the object web. WebFlow supports a 3-tier distributed visual dataflow computing model. Layers 1(client) and 2 (middle-tier)are written in Java and provided as part of the release together with a set of trial/demo modules. Layer 3 is left open for further specifications and refinements which allows us now to wrap any backend code as WebFlow module and link it to its tier 2 Java proxy via a customized socket connection. WebFlow middleware is given by a mesh of Web servers written in Java and hence offering a natural computational extensibility model via the dynamic properties of the underlying JavaVM. In '96, we analyzed two natural standard candidates in this area: Jeeves server by JavaSoft (later renamed as Java Web Server (JWS)) and Jigsaw server by the World-Wide Web Consortium.

We found light-weight Servlets in Jeeves to offer more attractive dynamic extensibility model than more heavy-weight Resources in Jigsaw and we selected Jeeves/JWS as a base of WebFlow middleware. Of course other web servers now support servlets and our current WebFlow work uses a modern Apache web server.

Each JWS node of WebFlow manages its portion of a distributed computation in terms of three management servlets: Session Manager, Module Manager and Connection Manager. Each Session Manager exports the externally visible URL as a WebFlow entry point and it controls the concurrent user sessions on this server. Connection Manager handles the module connection requests. Module Manager is responsible for loading the WebFlow modules to the Java VM and controling module life cycle (init, run, destroy).

WebFlow Module is a Java Object, which implements webflow.backend.Module interface. This interface contains three methods: a) init() - called after the module is created by the Module Manager; b) run() - called by the Module Manager when the module is fully instantiated, connected and ready to start the regular operation as a distributed dataflow node; c) destroy() - called when the Module Manager needs to stop the execution and to release the module object.

WebFlow front-end shown in figs. 7 and 8 is given by a Java applet, served by any of the JWS Session Managers and offering the visual interactive tool for dataflow authoring. In the current prototype, we based our front-end on the GEF (Graph Editing Framework) package from UCI [6] and we suitably extended it by building the URL and socket based communication between the applet and the JWS Session Manager servlet.

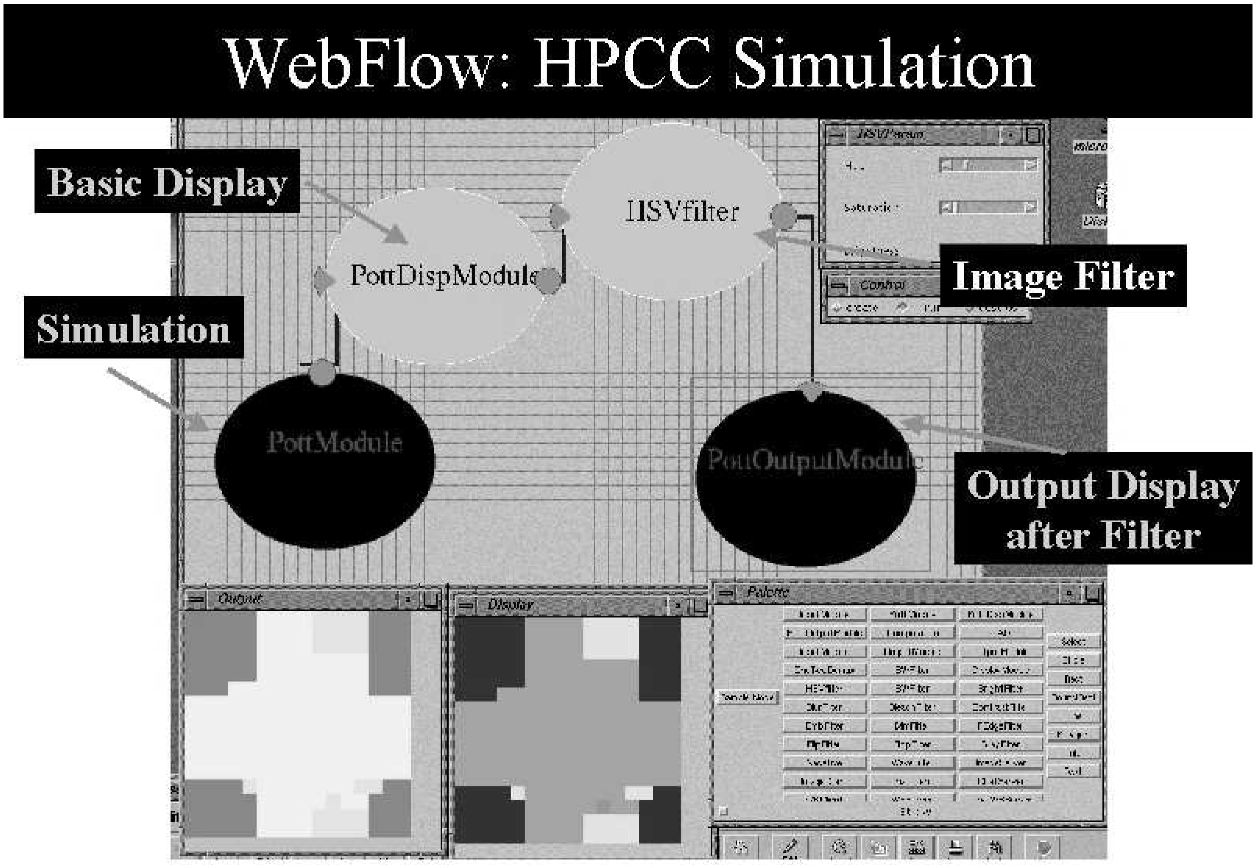

Fig.7: Sample WebFlow Application: HPCC simulation (Potts spin system) is wrapped as a WebFlow module and its real-time output stream is passed to the Display and Image Filter Modules which enable real time control and fine-tuning of the visualization display.

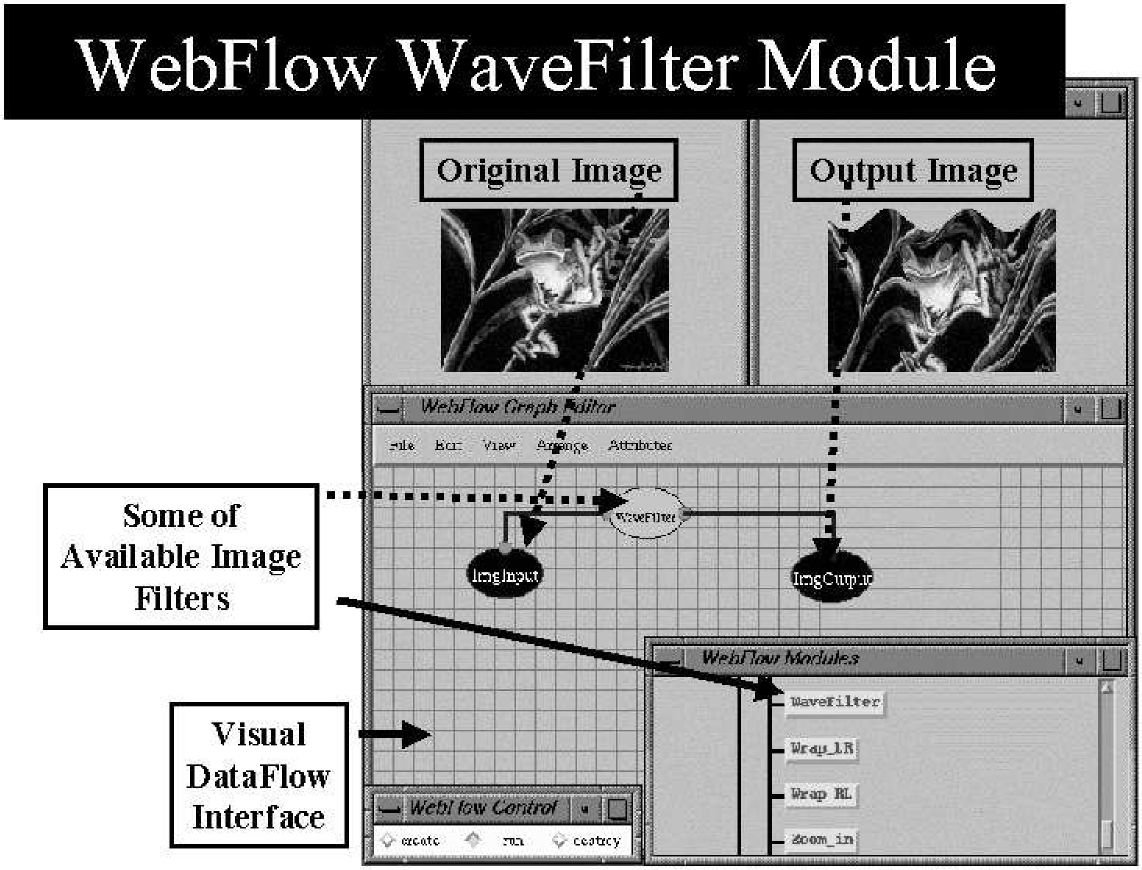

Fig.8: A simple (pure Java) Image Filter module (Wave Filter). Image input, output and filter are represented as visual icons in the WebFlow editor applet. Modules/icons can be selected from a tree-structured palette/navigator.

So far, we have developed HPCC tools representing a set of simple proof-of-the-concept backend modules testing various aspects of the system including:

a) Selected HPCC/HPF simulations (fig. 7);

b) AVS-style library of image processing filters (fig. 8);

c) Simple collaboratory sessions;

d) Interfaces to SciVis packages (fig. 9) [21];

e) Dynamic generation of breakpoint modules during a visual debugging session of an HPF application using the NPAC DARP (Data Analysis and Rapid Prototyping) [10] package support for the HPF environment and demonstrated at Supercomputing'97 [11].

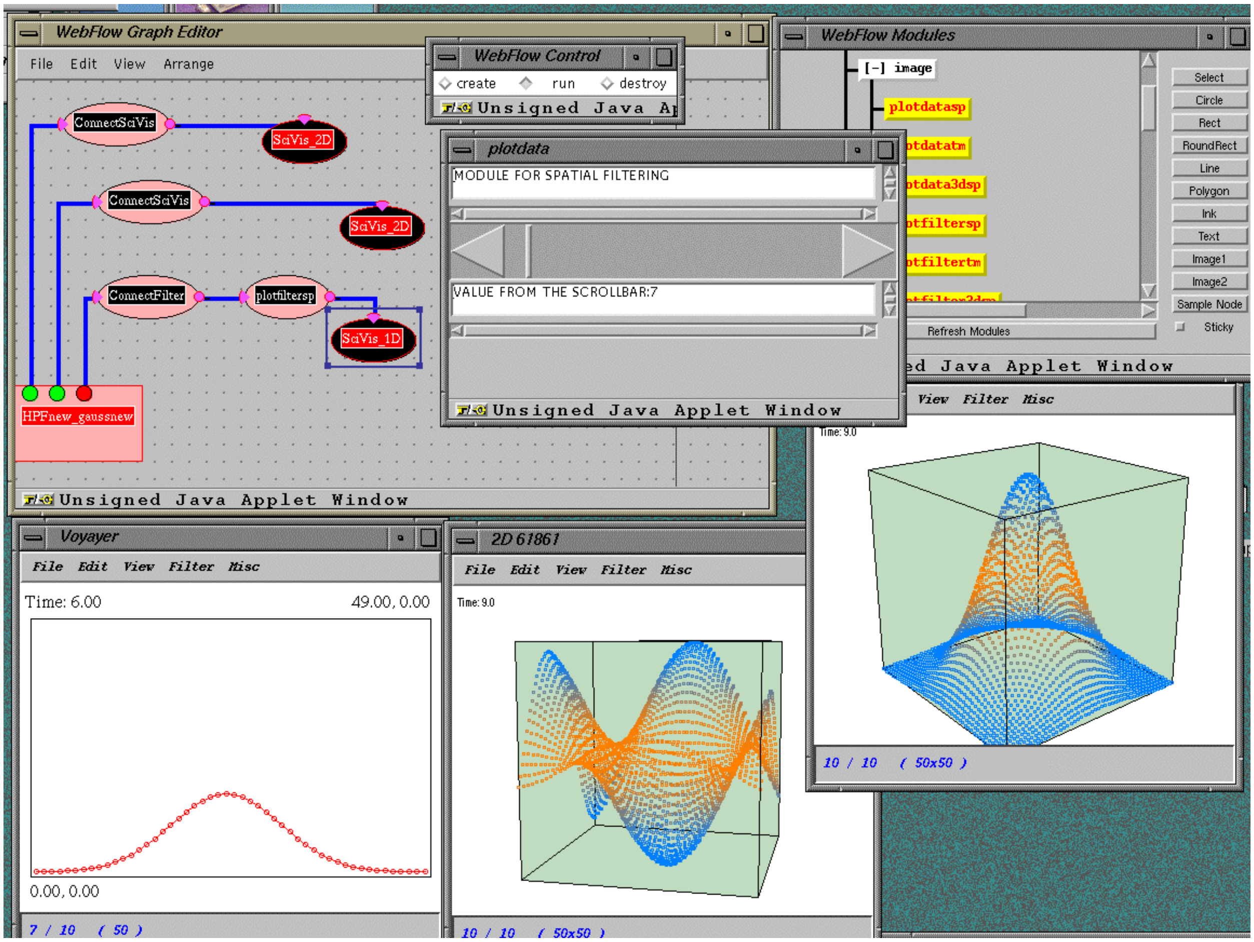

Fig.9: WebFlow demonstration at Supercomputing '97. HPCC simulation (Binary Black Holes) with the DARP based scripting/debugging support is wrapped as WebFlow module with variable number outputs, specified interactively and represeting visual debugging probes. Real-time data streams extracted this way from the simulation is passed (via optional filters) to suitable visualization modules that wrap and control NPAC SciVis display windows.

4: Towards Object Web Based WebFlow

In the new WebFlow design, we adopt the integrative Pragmatic Object Web methodology i.e. we setup a multiple-standards based framework in which the best assets of various approaches cumulate and cooperate rather than competing. We start the design from 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 metacomputing middleware that we wish to develop. 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 domains which therefore translates into the request of integrating the CORBA and Web technologies. We implement it by building a Java server which handles multiple network protocols and includes support both for HTTP and IIOP. This can be easily done as IIOP requests are distinguished by the 'GIOP' magic word whereas HTTP requests start from the corresponding 'GET', 'POST' etc. string identifiers.

We called such server JWORB (Java Web Object Request Broker) since it can act both as Java Web Server and as ORB for the Java objects. It can also act as a CORBA client or server for Java objects. Unlike in the WebFlow prototype where we used the JWS from JavaSoft, we decided to implement both HTTP and IIOP support of WORB from scratch. The reason is that neither Jigsaw (which is huge and still evolving) nor Jeeves/JWS (which evolved from core Java / JDK candidate to a commercial product) turned out to be useful for our purposes. Further, we believe that Web servers nowadays should be implemented using solid CORBA services. In the CORBA sector, we have some free Java ORB support such as OrbixWeb, VisiBroker for Java or JavaIDL but none comes with the source release. There is also an evolving GNU Java ORB called JacORB which however started before JavaIDL mapping was standardized, it already grew to a large volume and is now being slowly converted to the current OMG standards. Therefore, we decided to develop our own Java ORB sector as well and to make it fully compliant with the source level components of the coming JavaIDL (such as the org.omg.CORBA package).

With the JWORB based middleware, we can now address both the back-end and front-end layers of WebFlow in a uniform and elegant way using the IIOP protocol. Back-end components will be typically packaged as C/C++ CORBA servers (possibly wrapping some Fortran codes) and represented by the corresponding Java proxies managed by JWORB. The emergent CORBA components standard under development by OMG offers a natural model for the WebFlow middleware encapsulation of such proxies, mapped directly into JavaBeans at the JWORB level. Front-end remains fully factorized as in the current WebFlow prototype and can be given by any visual authoring package with a suitable 'bean box' functionality.

4.1: Front End: Commodity Tools for Visual Authoring

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). UML offers a broad spectrum of visual authoring graph topologies, some initial constructs for parallel processing and an extensibility model. We are currently analyzing this new standard from the perspective of adopting it as a visual model for new WebFlow front-end.

4.2: Middleware: Java Web Object Request Broker (JWORB)

JWORB is a multi-protocol extensible server written in Java. The base server has HTTP and IIOP protocol support. It can serve documents as an HTTP Web Server and it handles the IIOP connections as an Object Request Broker. As an HTTP server, JWORB supports both the Servlet and CGI mechanisms. Any servlet developed with Java Servlet API can run with JWORB.

Since JWORB design is Object Oriented, it is very easy to add other protocols. As JWORB starts up, it looks at configuration files to figure out which protocols it is capable of handling and it loads the necessary protocol classes for each protocol. If we want to add a new protocol, we need to implement a few abstract classes defined for the protocol object and to register this protocol implementation in the configuration file.

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 claims that it can serve this connection, then this protocol handler deals with this connection.

After the core JWORB server accepts a connection, it asks for a worker thread from the worker pool, it gives this connection to the worker thread and it returns its accepting state if there are available workers in the thread pool. The thread pool manager prevents JWORB from consuming all resources on the machine and creating a new thread object on each client request.

In the current design of core JWORB, the server process returns to accepting state if there is an available worker thread(s) for next request. Otherwise, it waits for a notification from the thread pool manager. In the next releases, we are planning to define an Event Queue and use CORBA Event Service to keep the request events in the queue for subsequent processing. This approach is very much like handling events in the distributed simulations and in fact our design here is driven by our DoD work on JWORB based HLA/RTI support for DoD Modeling and Simulation.

JWORB provides IIOP support, which is fully compliant with Java/IDL mapping by the OMG. We implemented the server side support and we are currently using idltojava from JavaSoft's JavaIDL as our IDL compiler. **idltojava** will be the standard utility of the next JDK releases.

We tested the performance of server objects 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.

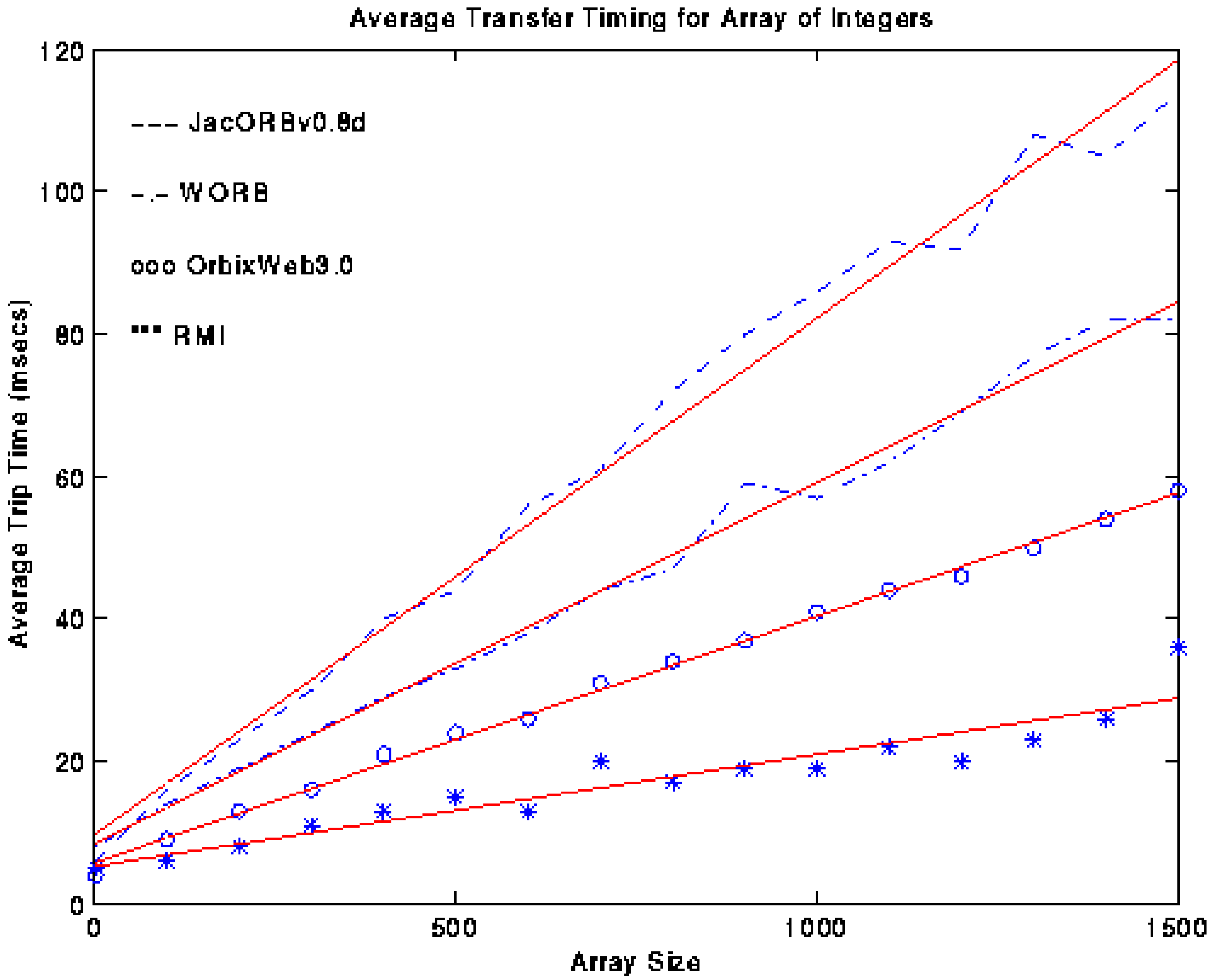

Fig.10: IIOP communication performance for variable size integer array transfer by four Java ORBs: JacORB, JWORB, OrbixWeb and RMI. As seen, initial JWORB performance is reasonable and further optimizations are under way. RMI appears to be faster here than all IIOP based models.

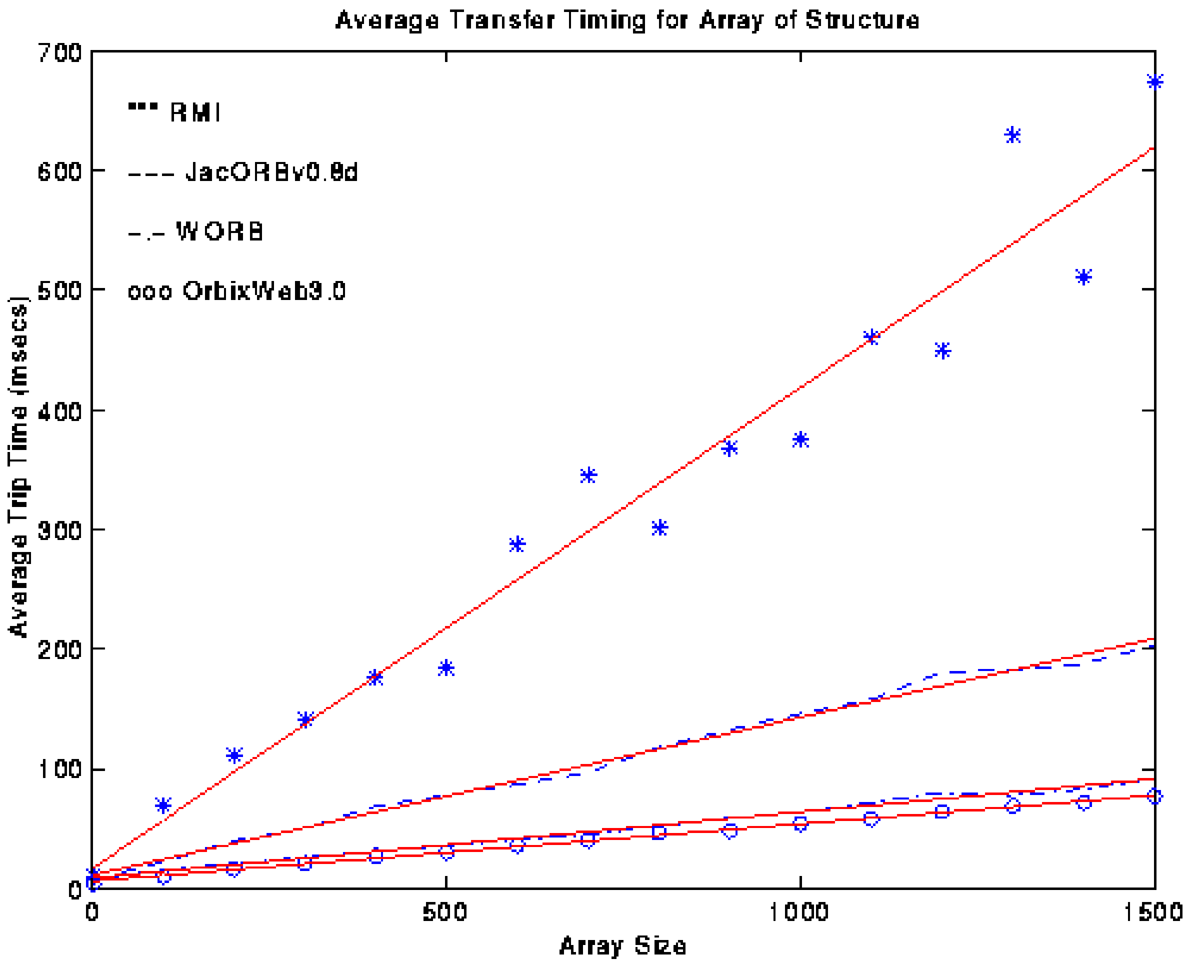

Fig 11: IIOP communication performance for transferring a variable size array of structures by four Java ORBs: JacORB, JWORB, OrbixWeb and RMI. Poor RMI performance is due to the object serialization overhead, absent in the IIOP/CDR protocol.

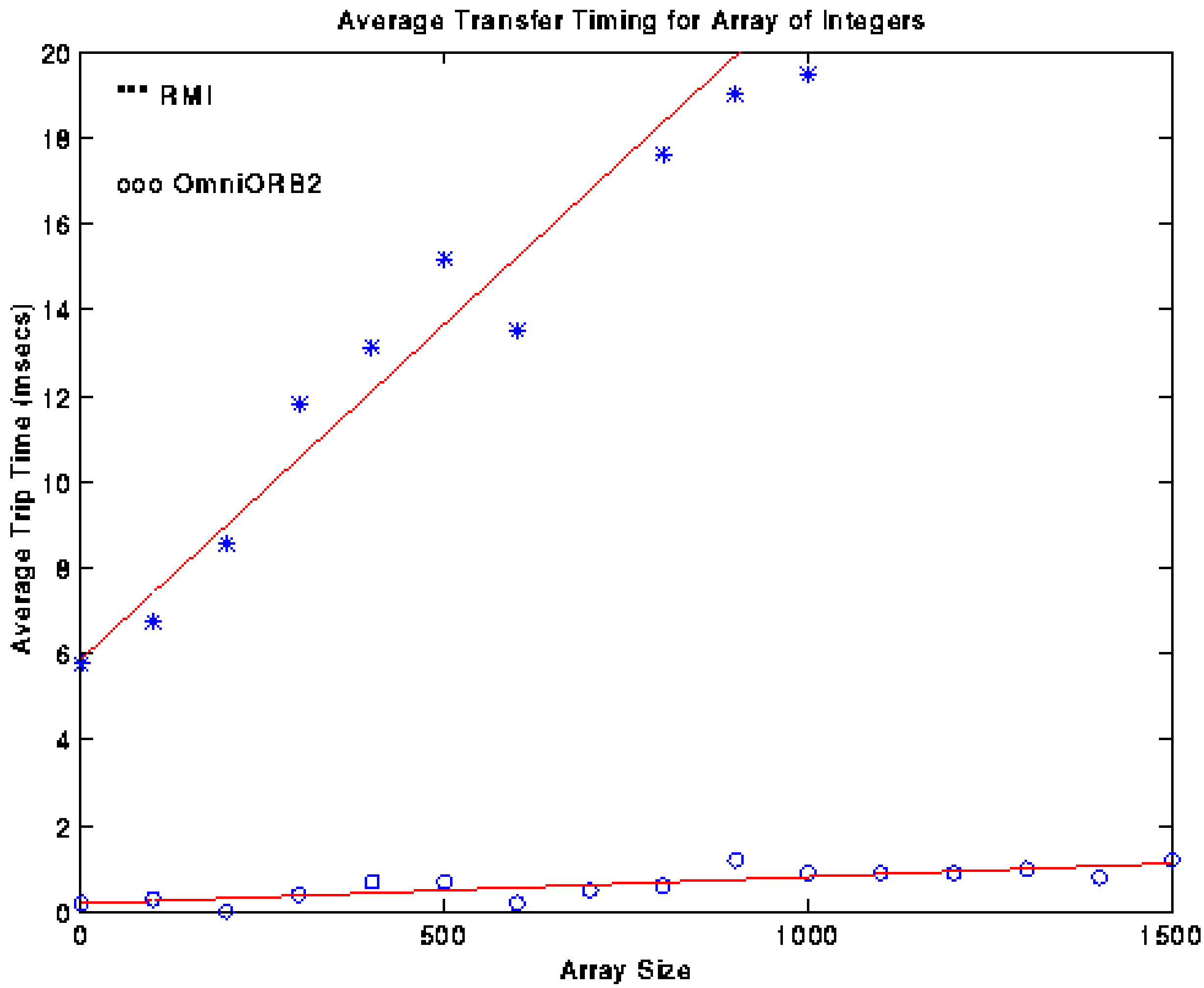

Fig 12: Initial performance comparison of a C++ ORB (omniORB) with the fastest (for integer arrays) Java ORB (RMI). As seen, C++ outperforms Java when passing data between distributed objects by a factor of 20.

The timing results presented above indicate that that JWORB performance is reasonable when compared with other ORBs even though we haven't invested yet much time into optimizing the IIOP communication channel. The ping value for various ORBs is the range of 3-5 msecs which is consistent with the timing values reported in the Orfali and Harkey book [13]. 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 performs poorly for arrays of structure. JacORB uses object serialization also for arrays of primitive types and hence its performance is poor on both figures.

We are currently doing a more detailed performance analysis of various ORBs, including C/C++ ORBs such as omniORB2 or TAO [19] 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 based metacomputing will be based on Globus/Nexus [14] backend (see next Section) and the associated high performance remote I/O communication channels wrapped in terms of C/C++ ORBs (such as omniORB2). However the middleware Java based ORB channels will be used mainly for control, steering, coordination, synchronization, load balancing and other distributed system services. This control layer does not require high bandwidth and it will benefit from the high functionality and quality of service offered by the CORBA model.

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

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

4.3: Back End: HPCC/Globus, HLA/RTI, Legacy Systems

In parallel with designing and prototyping new CORBA based WebFlow, we are also training Syracuse University students on building simple 3-tier metacomputing applications [20] and we start exploring the interfaces of the current WebFlow prototype to a suite of external backend components. As part of the NCSA Alliance, we are working with the NCSA scientists on adapting WebFlow for their HPDC applications in the Quantum Monte Carlo/Nanotechnology domain. We are also working with the Argonne team to explore the WebFlow based visual authoring tools as a possible front-end for a broader family of emergent metacomputing applications based on the Metacomputing Toolkit Globus. As part of the DoD HPC Modernization Program, we are building JWORB based support for RTI (Run-Time Infrastructure) which acts as a distributed object bus for the HLA (High Level Architecture) - a new DoD-wide standard for Modeling and Simulation. We are also adapting WebFlow as a visual simulation tool for HLA federations and we note that WebFlow can be integrated with POOMA [17] as part of the ASCI systems[18]. Finally, we are also exploring WebFlow model a possible integration framework for legacy systems such as heterogeneous distributed RDMS systems of relevance for telemedicine, distance training, virtual prototyping and other Web/commodity based community networks.

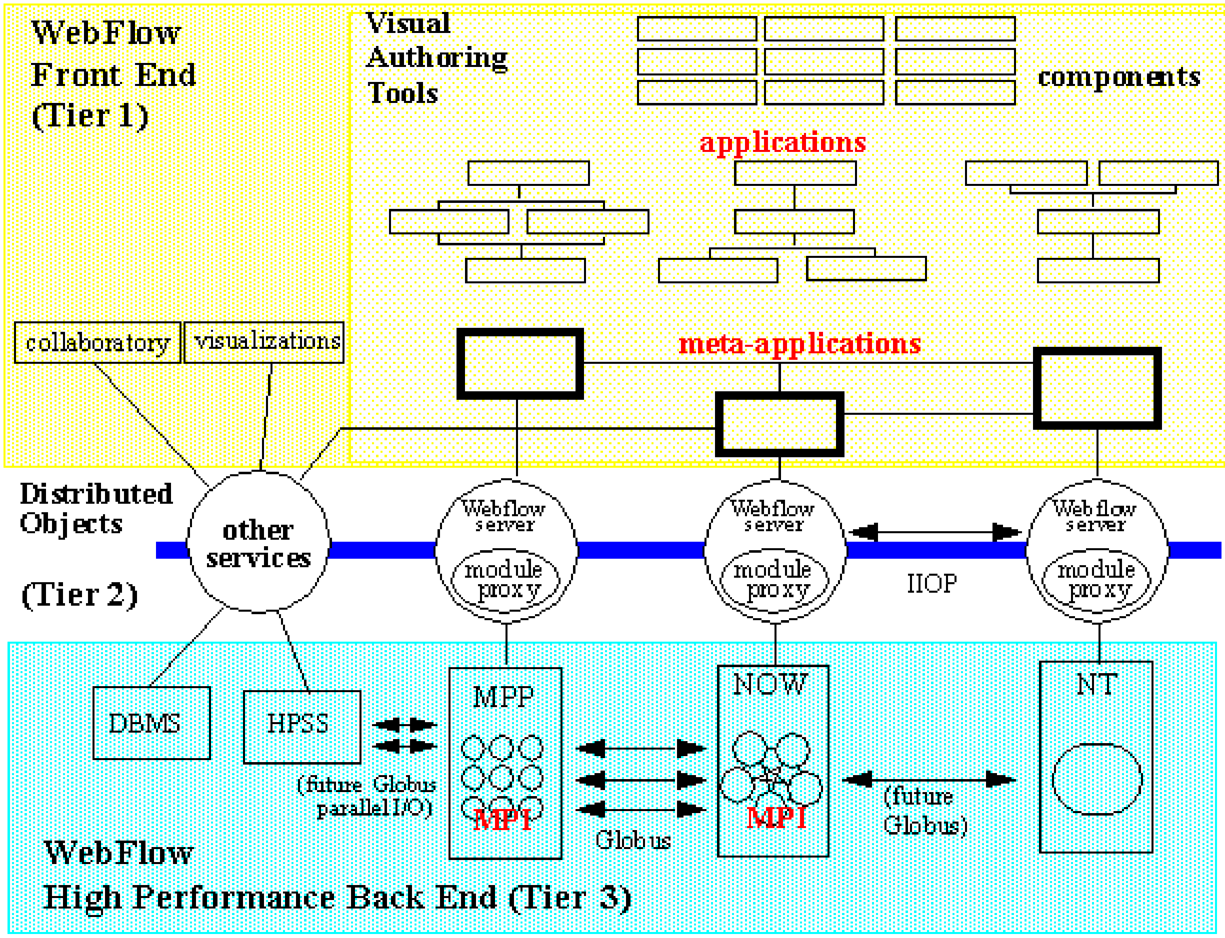

Fig 13: Overall view of the new WebFlow model. UML based front-end in tier 1 is linked to the JWORB based middleware which is linked via suitable module proxies to the metacomputing backend. WebFlow servers in tier 2 are given by JWORB servers dressed by a suitable collection of CORBA services, customized for metacomputing application purposes and supporting security, resource management, load balancing, fault tolerance, session control/journaling etc.

5: Related Work

At the tier-3 level, we already mentioned Globus [14] and we also intend to inspect WebFlow interfaces to the Legion [15] system. Some visual metacomputing issues were also addressed in the related VDCE [16] project at Syracuse. It is plausible that the CORBA based middleware offered by JWORB will allow us to formulate jointly as the HPDC community a CORBA Facility for Metacomputing that would provide users with binding to Globus, Legion, VDCE or other favored metacomputing backends.

At the tier-2 level, we are working with several Java ORBs used for testing purposes and comparative analysis such as OrbixWeb, VisiBroker, and JacORB. In the C++ sector, we view omniORB2 as a useful model and we intend to use it initially to wrap C, C++ and Fortran codes as CORBA components. We are learning from JacORB source when implementing IIOP sector of JWORB and from the Jigsaw and HP-Nexus sources when implementing the HTTP part of JWORB.

Several tier-1 packages were discussed in the text. We currently view UML [22] as the most promising model and we are at the planning stage of the prototype implementation.

We don't know of any other public domain effort similar to our new WebFlow model. One related commercial system is IBM Component Broker which involves IBM Mainframes in the backend, CORBA based middleware (based on IBM SOM model) and variety of commodity front-end options.

6: Acknowledgements

This document is focused on JWORB based middleware of the new WebFlow model. We want to acknowledge here the effort of several other researchers who contributed to the current and/or are contributing to the new version of WebFlow. The early WebFlow prototype was implemented by Vanco Burzevski and Maja Camuseva with the help of Girish Premchandran and Dimple Bhatia who also developed most of the WebFlow modules. Java code for the imaging modules was contributed by Shrideep Palickara. Tom Haupt and Erol Akarsu contributed their DARP technology and helped to develop HPF WebFlow modules for Supercomputing'97. Tom Haupt is now also leading efforts on linking WebFlow with Globus and exploring the initial applications at NCSA and CEWES. Bharat Ananthakrishnan contributed in the initial stage of the JWORB/HTTP development and he explored JavaStudio/WebFlow interface. Mahesh Rengaswamy explores VisualAge and the Rational Rose products as WebFlow interface candidates. Tom Pulikal offers WebFlow system administrative support and explores interfaces to legacy databases (Oracle, Access, SQL Server). Zeynep Ozdemir investigates the role of DCOM and DirectX front-ends and Balaji Natarajan is working on WebFlow based visual simulation tools for distributed simulations/wargaming.

Wojtek Furmanski is grateful to Salim Hariri and his research group for the insight into the HPDC backend issues acquired while participating in the VDCE (Virtual Distributed Computing Environment) project at the CASE Center at Syracuse University, sponsored by Rome Laboratory.

The initial WebFlow prototype was supported by the Department of Energy. Current work on CORBA based WebFlow is supported by the DoD High Performance Modernization Program, National Science Foundation and the industry grants from IBM Watson and Translet, Inc.

References

1. Geoffrey Fox, Wojtek Furmanski, Marina Chen, Claudio Rebbi and Jim Cowie, *WebWork: Integrated Programming Environment Tools for National and Grand Challenges,* http://www.npac.syr.edu/techreports/html/0700/abs-0715.html, March 1995.

2. *WebFlow - a visual programming paradigm for Web/Java based coarse grain distributed computing* – http://www.npac.syr.edu/projects/webspace/doc/sc96/handout/handout.ps

Supercomputing'96 Handout Material

3. *Web based Computing - WebFlow* - http://www.npac.syr.edu/projects/webspace/doc/cewes-mar97/talk/ CEWES talk, March '97

4. *WebFlow - a visual programming paradigm for Web/Java based coarse grain distributed computing*, http://www.npac.syr.edu/projects/webspace/doc/cpande/feb97/feb97.ps February '97, special issue of *Concurrency: Practice and Experience* on Java for Scientific Computing.

5. *WebFlow Project Home Page at NPAC* -- http://osprey7.npac.syr.edu:1998/iwt98/products/webflow/

6. Jason Robbins *GEF: Graph Editing Framework*-- http://www.ics.uci.edu/pub/arch/gef/

7. Craig Thompson, *OMG/DARPA Workshop on Compositional Software Architectures*, http://www.objs.com/workshops/ws9801/ Monterey, CA January 6-8 1998.

8. Dennis Gannon, *Component Architectures for High Performance Distributed Meta-Computing*, http://www.objs.com/workshops/ws9801/papers/paper086.html OMG/DARPA Workshop, January 1998.

9. ***JWORB*** Home Page at NPAC -- http://osprey7.npac.syr.edu:1998/iwt98/projects/worb/

10. E. Akarsu, G. Fox, T. Haupt, *DARP: Data Analysis and Rapid Prototyping Environment for Distributed High Performance Computations* , http://www.npac.syr.edu/projects/hpfi/

11. G. Fox, W. Furmanski and T. Haupt, *SC97 handout: High Performance Commodity Computing (HPcc)* , http://www.npac.syr.edu/users/haupt/SC97/HPccdemos.html

12. G. Fox and W. Furmanski, *HPcc as High Performance Commodity Computing*, http://www.npac.syr.edu/users/gcf/HPcc/HPcc.html

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

14. ***Globus*** Project Home Page, http://www.globus.org/

15. ***Legion*** Project Home Page, http://www.cs.virginia.edu/~legion/

16. ***VDCE*** Project Home Page, http://koshka.cat.syr.edu/projects/vm/

17. ***POOMA*** Project Home Page, http://www.acl.lanl.gov/PoomaFramework/

18. G. Fox, W. Furmanski and T. Haupt, *ASCI WebFlow: High-Level Programming Environment and Visual Authoring Toolkit for High Performance Distributed Computing*, http://www.npac.syr.edu/projects/asci-webflow

19. ***TAO*** Project Home Page, http://www.cs.wustl.edu/~schmidt/TAO.html

20. Tom Studer's ***CPS714*** NPAC Course Final Project, http://osprey7.npac.syr.edu:3768/cps616spring97-docs/cb97tst/CPS714/

21. NPAC ***SciVis*** Project Home Page, http://kopernik.npac.syr.edu:8888/scivis/index.html

22. ***UML*** Home Page, http://www.rational.com/uml