Using WebFlow to Provide a Seamless Access to Remote Resources

Tomasz Haupt, Erol Akarsu and Geoffrey Fox

Notheast Parallel Architecture Center at Syracuse University. 111 College Place Room 3-217, Syracuse, NY 13244-4100

Abstract

In this paper we discuss use of WebFlow for a seamless access to remote resources. We illustrate our ideas with two WebFlow applications that require access to remote resources: the Landscape Management System (LMS) and Quantum Simulations (QS). For LMS we use WebFlow to retrieve data from many different sources as well as to allocate remote computational resources needed to solve the problem at hand. WebFlow transparently for the user controls the necessary data transfer between hosts. Quantum Simulations requires access to HPCC resources and therefore we layered WebFlow on top of Globus metacomputing toolkit. This way WebFlow plays the role of a job broker for Globus. We admit that WebFlow does not comprise a complete solution yet for the seamless access to remote resources. Many issues, notably security, still remain to be solved. We expect to leverage from the recent DATORR (Desktop Access to Remote Resources) initiative of the Java Grande Forum when tackling these issues.

1 Overview of WebFlow

The visual HPDC framework introduced by this project offers an intuitive Web browser based interface and a uniform point of interactive control for a variety of computational modules and applications, running at various labs on different platforms. Our technology goal is to build a high-level user friendly commodity software based visual programming and runtime environment for HPDC.

WebFlow is a particular programming paradigm implemented over a virtual Web accessible metacomputer and given by a dataflow-programming model (other models under experimentation include data parallel, collaboratory and other paradigms). WebFlow application is given by a computational graph visually edited by end-users, using Java applets. Modules are written by module

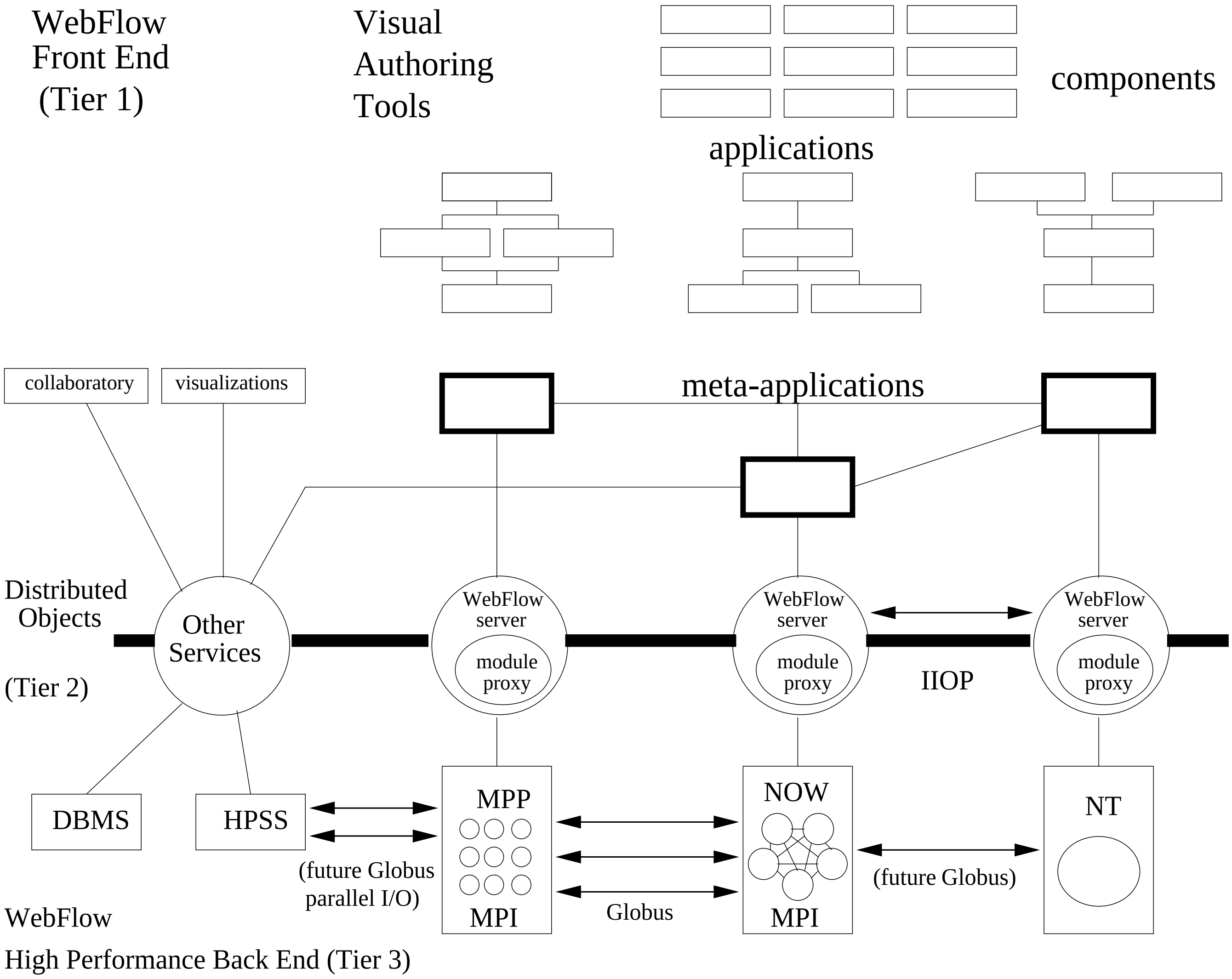

Fig. 1. Three-Tier Architecture of WebFlow

developers, people who have only limited knowledge of the system on which the modules will run. They not need concern themselves with issues such as: allocating and running the modules on various machines, creating connections among the modules, sending and receiving data across these connections, or running several modules concurrently on one machine. The WebFlow system hides these management and coordination functions from the developers, allowing them to concentrate on the modules being developed.

The implementation of WebFlow is described in details in our previous papers[1,2]. Here, we briefly recall that WebFlow is implemented as a modern three-tier system, as shown in (Figure 1):

Tier 1 is a high-level front-end for visual programming, steering, run-time data analysis and visualization, and collaboration built on top of the Web and OO commodity standards. Distributed object-based, scalable, and reusable

Web server and Object broker Middleware forms Tier 2. Back-end services comprise Tier 3. In particular, high performance services are implemented using the metacomputing toolkit of Globus[5].

In our current prototype of WebFlow, the middle-tier is given by a mesh of Java enhanced Web Servers, running servlets that manage and coordinate distributed computation. This management is implemented in terms of the three servlets: Session Manager, Module Manager, and Connection Manager. These servlets are URL addressable and can offer dynamic information about their services and current state. Each of them can also communicate with each other through sockets. Servlets are persistent and application independent.

In spite of the success of the WebFlow project we see that our current implementation suffers form some limitations. Two the most obvious areas of improvement we want to achieve are fault tolerance and security. However, instead of adding complexity to already complex and to large extend custom protocol of exchanging data between the servlets based on low-level sockets, we are reimplementing the WebFlow middle-tier using industry standards distributed object technologies: JavaBeans and CORBA. Our early tests proved that WebFlow can be implemented using Java event model, and propagate the events through networks using CORBA. The details of the design will be described in an upcoming paper.

The backed modules are Java objects that implement WebFlow specific module interface. The interface defines mandatory methods of a module: initialize, run, and destroy. During initialization the newly created object register itself to the module manager, and registers its input and output ports to the connection manager. The run method implements the actual functionality of the module, while the destroy method release resources taken by the module, for example, socket connections are closed. This design matches very accurately our requirements to build from predefined modules an application that follows the dataflow paradigm. In our new design a module is a JavaBean allowing us to generalize response of the modules on events.

The functionality of the module (the run method) can be implemented entirely in Java, and this is a way in which we develop new WebFlow modules. However, existing applications written in a language different than Java can be easily encapsulated in a WebFlow module using either JNI or via classes from the System package. More, the execution of the module can be delegated to an external system capable of resource allocation such as Globus.

In order to run WebFlow over Globus there must be at least one WebFlow node capable of executing Globus commands, such as globusrun. In other words, there must be at least one host on which both Globus and WebFlow server are installed. This host serves as a "bridge" between two domains (Figure

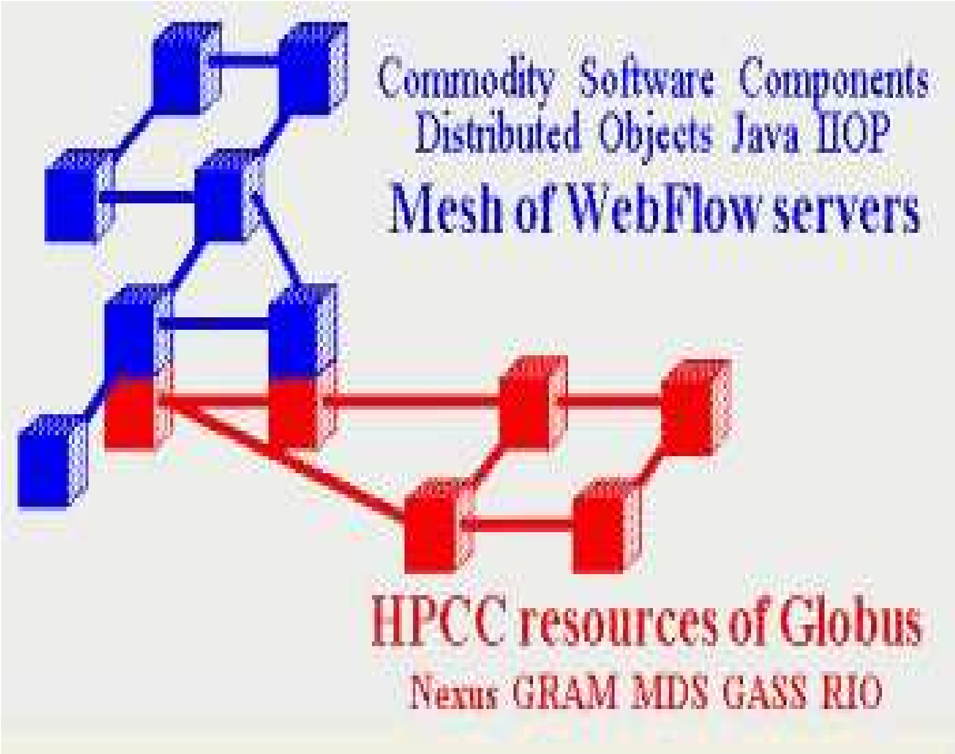

Fig. 2. Bridge between WebFlow and Globus resources (the Grid)

2): a network of WebFlow servers and a network of resources controlled by Globus(the Grid). The jobs that require computational power of massively parallel computers are directed to the Globus domain, while others can be launched on much more modest platforms, such as the user's desktop or even a laptop running Windows NT.

Both Globus and WebFlow gains from this symbiotic coexistence. From the WebFlow perspective, Globus is an optional, high performance (and secure) back-end, while WebFlow serves as a high-level web accessible visual interface and job broker for Globus. Together they cover much wider application domain: Globus adds HPCC world to WebFlow, WebFlow adds commodity software, in particular that available only on Microsoft Windows95/98/NT.

We are aware that providing a remote access to the Globus resources (either via front-end applets or WebFlow server-to-server connections) we may introduce a security hole in the system. We made several experiments to upgrade WebFlow security standards to match these of Globus. However, at this time we have postponed incorporating any security mechanism into WebFlow till we rebuild the middle-tier using CORBA and some widely accepted standards emerge - hopefully defined by DATORR[3].

2 WebFlow Application: Land Management System (LMS)

LMS project was sponsored by the U.S. Army Corps of Engineers Waterways Experiment Station (CEWES) Major Shared Resource Center (MSRC) at Vicksburg, MS, under the DoD HPC Modernization Program, Programming Environment and Training (PET). The first, pilot phase of the project can be described as follows. A decision maker (the end user of the system) wants to evaluate changes in vegetation in some geographical region over a long time period caused by some short term disturbances such as a fire or human's activities. One of the critical parameters of the vegetation model is soil condition at the time of the disturbance. This in turn is dominated by rainfalls that possibly occur at that time. Consequently, the implementation of this project requires:

- Data retrieval from remote sources including DEM (data elevation models) data, land use maps, soil textures, dominating flora species, and their growing characteristics, to name a few. The data are available from many different sources, for example from public services such as USGS web servers, or from proprietary databases. The data come in different formats, and with different spatial resolutions. Without WebFlow, the data must be manually prefetched.
- Data preprocessing to prune and convert the raw data to a format expected by the simulation software. This preprocessing is performed interactively using WMS[4] (Watershed Modeling System) package.
- Execution of two simulation programs: EDYS[4] for vegetation simulation including the disturbances and CASC2D[4] for watershed simulations during rainfalls. The latter results in generating maps of the soil condition after the rainfall. The initial conditions for CASC2D are set by EDYS just before the rainfall event, and the output of CASC2D after the event is used to update parameters of EDYS. We used test data sets for time periods with at least 2 rainfalls. Consequently the data transfer between the two codes had to be performed several times during one simulation. EDYS is not CPU demanding, and it is implemented only for Windows95/98/NT systems. On the other hand, CASC2D is very computationally intensive and typically is run on powerful UNIX compute servers.
- Visualization of the results of the simulation. Again, WMS is used fr this purpose.

The purpose of this project was to demonstrate feasibility of implementing a system that would allow launching and controlling the complete simulation from a networked laptop. We successfully implemented it using WebFlow with WMS and EDYS encapsulated as WebFlow modules running locally on the laptop and CASC2D executed by WebFlow on remote hosts, either at CEWES in Vicksburg, MS or NPAC in Syracuse, NY. We demonstrated the

Fig. 3. Retrieving DEM and Land Used data from USGS.

system using a Pentium II based laptop running Windows NT located in Washington, DC (with CASC2D running at Vicksburg) and in Orlando, FL during Supercomputing '98 (with CASC2D running alternatively at Syracuse and Vicksburg).

For this project we developed a custom front-end that allows the user to interactively select the region of interest by drawing a rectangle on a map, select the data type to be retrieved, launch WMS to preprocess the data and make visualizations, and finally launch the simulation with CASC2D running on a host of choice. An example screendump is given in (Figure 3).

3 WebFlow Application: Quantum Simulations (QS)

QS project is performed within Alliance Team B and its primary purpose is demonstrate feasibility of layering WebFlow on top the Globus metacomputing toolkit. This way WebFlow serves as a job broker for Globus, while Globus (or more precisely, GRAM-keeper) takes responsibility of actual resource allocation, which includes authentication and authorization of the WebFlow user to use computational resources under Globus control.

This application 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 logical structure of the application is shown in (Figure 4).

GAUSSIAN and GAMES are run as Globus jobs on Origin2000 or Convex Exemplar at NCSA, while all file editing and format conversion a performed on the user's desktop.

Unlike LMS, for QS we are using the WebFlow editor as the front-end. The WebFlow editor provides an intuitive environment to visually compose (click-drag-and-drop) a chain of data-flow computations from preexisting modules (as shown in Figure 4). In the edit mode, modules can be added to or removed from the existing network, as well as connections between the modules can be updated. Once created network can be saved (on the server side) to be restored at a later time. The workload can be distributed among several WebFlow

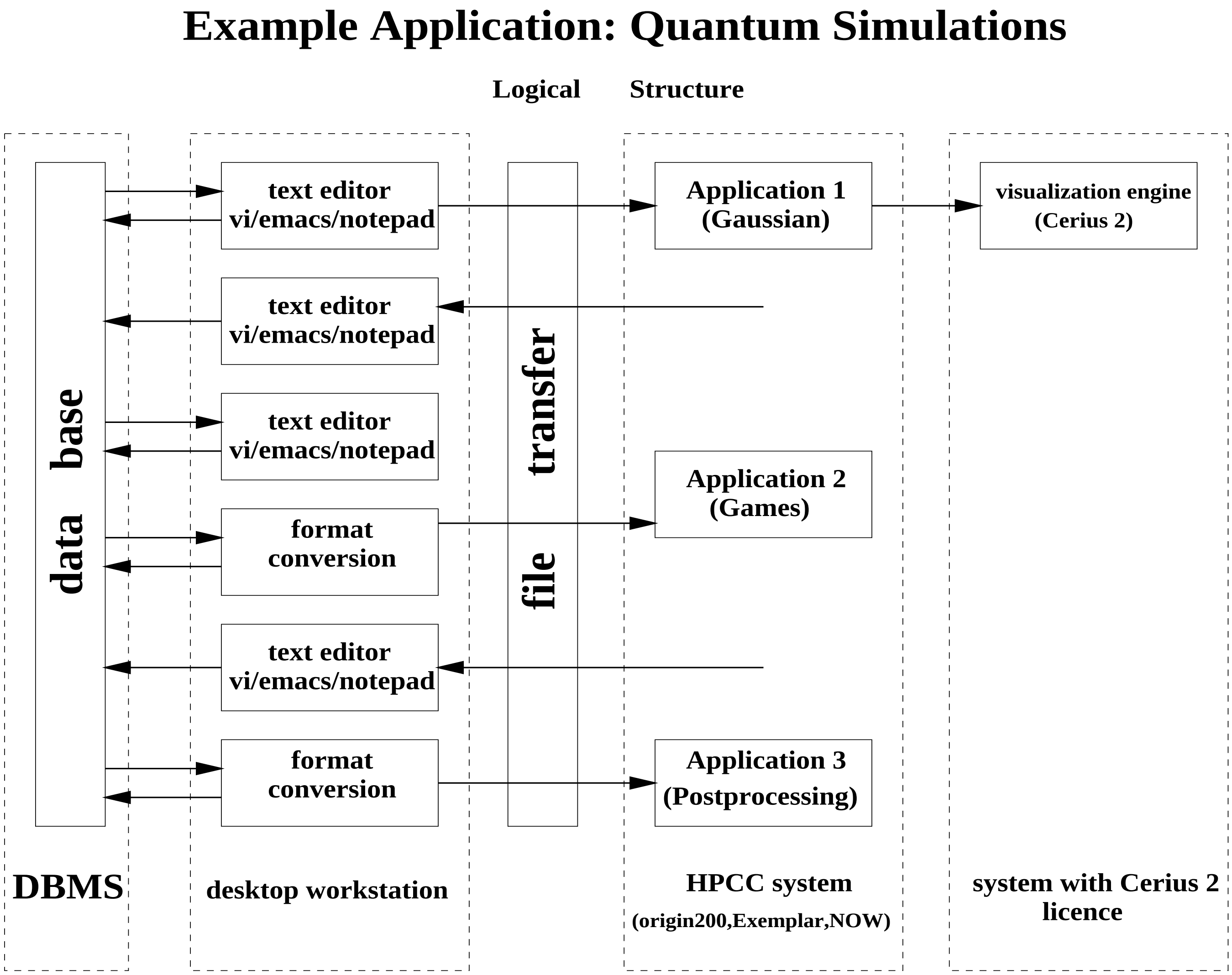

Fig. 4. Logical structure of the QS application

nodes (WebFlow servers) with the interprocessor communications taken care of by the middle-tier services. More, thanks to the interface to the Globus system in the backend, execution of particular modules can be delegated to powerful HPCC systems. In the run mode, the visual representation of the metaaplication is passed to the middle-tier by sending a series of requests (module instantiation, intermodule communications) to the Session Manager.

The control of the module execution is exercised not only by sending relevant data through the module's input ports. Majority of modules we developed so far requires some additional parameters that can be entered via "Module Controls" (similarly to systems such as AVS). The module controls are Java applets displayed in a card panel of the main WebFlow applet. The communication channels (sockets) between the backend implementation of a module and its front-end Module Controls are generated automatically during the instantiation of the module.

Not all applications follow closely the data flow paradigm. Therefore it is necessary to define an interface so that different front-end package can be "plugged in" into the middle-tier, giving the user a chance to use the front-end that best fits the application at hand. Currently we offer visual editors based on GEF[13] and VGJ[14]. In the future, we will add an editor based on the UML[12] standard, and we will provide an API for creating custom ones.

4 Summary

We use WebFlow to provide a seamless access to remote resources for two applications that require it. For LMS we use WebFlow to retrieve data from many different sources as well as to allocate remote computational resources needed to solve the problem at hand. WebFlow transparently for the user controls the necessary data transfer between hosts. Quantum Simulations requires access to HPCC resources and therefore we layered WebFlow on top of Globus metacomputing toolkit. We admit that WebFlow does not comprise a complete solution yet for the seamless access to remote resources. Many issues, notably security, still remain to be solved. We expect to leverage from the recent DATORR (Desktop Access to Remote Resources) initiative of the Java Grande Forum when tackling these issues.

References

[1] Erol Akarsu,Tomasz Haupt, Geoffrey Fox and Wojtek Furmanski, WebFlow-High-Level Programming Environment and Visual Authoring Toolkit for High Performance Distributed Computing, Supercomputing'98, November 1998.

[2] Tomasz Haupt,Erol Akarsu and Geoffrey Fox, Web Based Metacomuting, Special Issue on MetaComputing for the FGCS International Journal on Future Generation Computing Systems.

[3] DATORR(Desktop Access to Remote Resources), http://www.javagrande.org

[4] WMS, EDYS and CASC2D codes has been made available to us by CEWES. EDYS is written by Michael Childress and CASC2D is written by Fred Ogden. http://www.wes.hpc.mil

[5] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit", International Journal of Super computing Applications, 1997. See also Globus Home Page: http://www.globus.org

[6] G. Fox and W. Furmanski, "HPcc as High Performance Comodity Computing", chapter for the "Building National Grid" book by I. Foster and C. Kesselman, http://www.npac.syr.edu/users/gcf/HPcc/HPcc.html

[7] CORBA - OMG Home Page http://www.omg.org

[8] COM Home Page http://www.microsoft.com/com

[9] Enterprise JavaBeans http://java.sun.com/products/ejb/

[10] Quantum Simulations ,
http://www.ncsa.uiuc.edu/Apps/CMP/cmp-homepage.html

[11] Advanced Visualization System, http://www.avs.com/

[12] UML Home Page http://www.rational.com/uml

[13] GEF: The Graph Editing Framework home page:
http://www.ics.uci.edu/pub/arch/gef/

[14] VGJ, Visualizing Graphs with Java home page:
http://www.eng.auburn.edu/department/cse/research/graph_drawing/graph_drawing.html

[15] D. Bhatia, V. Burzevski, M. Camuseva, G. C. Fox, W. Furmanski and G. Premchandran, "WebFlow - a visual programming paradigm for Web/Java based coarse grain distributed computing", Concurrency: Practice and Experience, Vol. 9 (6), pp. 555-577, June 1997. See also WebFlow Home Page at NPAC: http://osprey7.npac.syr.edu:1998/iwt98/products/webflow.

[16] E. Akarsu, G. Fox, T. Haupt, DARP: Data Analysis and Rapid Prototyping Environment for Distribute High Performance Computations, Home Page http://www.npac.syr.edu/projects/hpfi/

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

[18] JWORB Project
Home Page http://osprey7.npac.syr.edu:1998/iwt98/projects/worb, see also G. C. Fox, W. Furmanski and H. T. Ozdemir, "JWORB (Java Web Object Request Broker) based Scalable Middleware for Commodity Based HPDC", submitted to HPDC98.

[19] Client/Server Programming with JAVA and CORBA 2nd. Ed. by Robert Orfali and Dan Harkey, February'98, ISBN:047124578X

[20] Apache Http Server Project http://www.apache.org/

[21] Alliance'98 meeting,
http://www.npac.syr.edu/users/haupt/WebFlow/papers/alliance98.html