The Gateway System: Uniform Web Based Access to Remote Resources

Geoffrey Fox, Tomasz Haupt, Erol Akarsu, Alexey Kalinichenko, Kang-Seok Kim, Praveen Sheethalnath, Choon-Han Youn

Northeast Parallel Architecture Center at Syracuse University

Abstract
Exploiting our experience developing the WebFlow system, we designed the Gateway system to provide seamless and secure access to computational resources at ASC MSRC. The Gateway follows our commodity components strategy, and it is implemented as a modern three-tier system. Tier 1 is a high-level front end for visual programming, steering, run-time data analysis and visualization that is 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, access to high-performance computational resources is provided by implementing the emerging standard for metacomputing API.

1. Introduction

The last few years have seen the growing power and capability of commodity computing and communication technologies largely driven by commercial distributed information systems. All of them can be abstracted to a three-tier model with largely independent clients connected to a distributed network of servers. High performance can be obtained by combining concurrency at the middle tier with optimized, parallel back-end servers. The resultant system combines the performance needed for large-scale HPCC applications with the rich functionality of commodity systems.

In each commodity technology area we have impressive and rapidly improving software artifacts. Perhaps even more important than raw technology, we have a set of standards and open interfaces enabling the development of distributed modular software. 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 for building HPCC environments in a way that naturally inherits all commodity capabilities so that HPCC applications can benefit from the impressive productivity of commodity systems. We have termed such an approach High Performance Commodity Computing (HPcc).

In several related papers [1] we have described this HPcc activity, which were designed to demonstrate that it is both possible and useful to achieve simultaneously both high performance and functionality of commodity systems. One aspect of this activity is a specific high-level programming environment developed at NPAC – WebFlow [2] – that offers a user-friendly, visual graph-authoring metaphor for seamless composition of world-wide distributed high performance dataflow applications from reusable computational modules.

A WebFlow front-end editor applet offers an intuitive click-and-drag metaphor for instantiating middleware or back-end modules, representing them as visual icons in the active editor area, and interconnecting them visually in the form of computational graphs, familiar to AVS and Khoros users. WebFlow middleware is given by a mesh of Java web servers, custom extended with servlet-based support for the WebFlow session known as Module and Connection Managers. WebFlow modules are specified as Java interfaces to computational Java classes or wrappers (module proxies) to back-end services. The high-performance part of the back-end tier is implemented using the Globus metacomputing toolkit [3].

One of the most spectacular applications of the WebFlow is Quantum Monte Carlo Simulations [4], developed in collaboration with the NCSA Condensed Matter Physics Laboratory. Here, a chain of high-performance applications (either 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. The output file of one application in the chain is the input of the next one, after a suitable format conversion.

In spite of the success of the WebFlow project we see that the original implementation, based on Java web servers, suffers from severe limitations. Two of the most obvious areas in which we want to achieve improvements are fault tolerance and security. However, rather than add further complexity to already complexity and to extend customary protocol of exchanging data between servers, we have re-implemented the WebFlow middle-tier using the industry standards distributed-object technologies, JavaBeans and CORBA and industry standard secure communication protocols based on SSL.

The development of the new middle-tier of our system coincides with the JavaGrande [5] initiative to develop international standards for seamless Desktop Access to Remote Resources (DATORR). These standards replace the remaining two custom WebFlow interfaces: a computational graph generated by the WebFlow front end by Abstract Task Specification and specific Globus interface by the universal metacomputing API.

The new implementation of WebFlow is a part of the project of the Aeronautical Systems Center (ASC) Major Shared Resource Center (MSRC) Gateway, sponsored by DoD HPC Modernization Program, Programming Environment and Training. The objectives of this project are to provide seamless and secure access to computational ASC MSRC resources through Web-based interfaces. The functionality of the Gateway system is specified in Section 2. Section 3 presents the system architecture and provides a high-level description of its major components. In Section 4 we discuss the Gateway security model and in Section 5 we reveal details of the middle-tier implementation. Section 6 provides links to related research, and Section 7 offers a summary.

2. Overview of the Gateway Functionality

The Gateway system offers a particular programming paradigm implemented over a virtual Web-accessible metacomputer. A (meta-) application is composed of independently developed modules. The modules are implemented in Java, and they follow the distributed JavaBeans model. This gives the user the complete power of Java, and object-oriented programming in general, which with to implement module functionality. However, the functionality of a module does not have to be implemented entirely in Java. Existing applications written in languages other than Java can be easily encapsulated as JavaBeans.

Module developers have only limited knowledge of the system on which the modules will run. They need not 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 Gateway system hides such management and coordination functions from the developers, allowing them to concentrate on the modules being developed.

Often, the modules serve as proxies for particular back-end services made available through the Gateway system. For example, an access to a database is provided through JDBC API, thus delegating implementation of module functionality to a back-end DBMS. We follow a similar approach to provide access to high-performance resources: a Gateway module "merely" implements an API of back-end metacomputing services such as those provided by the Globus metacomputing toolkit. In particular, a module that serves as the GRAM (Globus Resource Allocation Manager) proxy generates a resource allocation request. The request essentially defines an executable, its standard input, error and output streams, and the target machine where the executable is to be run. The application represented by the executable is developed independently of the Gateway system (for example, it may be a legacy parallel code written in Fortran + MPI). The role of the Gateway module written in Java is reduced to generating the request following the low-level Globus Resource Specification Language (RSL) syntax. In this sense the Gateway system can be regarded as a high-level, visual user interface and job broker for the Globus system.

The Gateway system supports many different programming models for distributed computations from coarse-grain dataflow to object-oriented to fine-grain data-parallel model. In the dataflow regime a Gateway application is given by a computational graph visually edited by the end users. The modules comprising the application exchange data through input and output ports in a way similar to that used in AVS. This model is generalized in our new implementation of the Gateway system. Thanks to the fact that modules behave as distributed JavaBeans, each module may invoke an arbitrary method of the other modules involved in the computation.

3. Gateway Architecture

The Gateway system is implemented as a modern three-tier system, as shown in Fig. 1. Tier 1 is a high-level *front end* for visual programming, steering, run-time data analysis, and visualization, that is built on top of the Web and OO commodity standards. A 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.

3.1 Front End

Different classes of applications require different functionality of the front end. We have therefore designed the Gateway system to support many different front-ends: from very flexible authoring tools and problem solving environments (PSE) that allows for dynamical creation of meta-applications from pre-existing modules, to highly specialized and customized front-ends to meet the needs of specific applications. Also, we support many different computational paradigms, from general object-oriented to data-flow to a simple "command line" approach. This flexibility is achieved by treating the front end as a plug-in implementing the Gateway API.

3.2 Gateway API

The Gateway API allows the user's task to be specified in the form of an Abstract Task Descriptor (ATD), following the current DATORR recommendations. The ATD is constructed

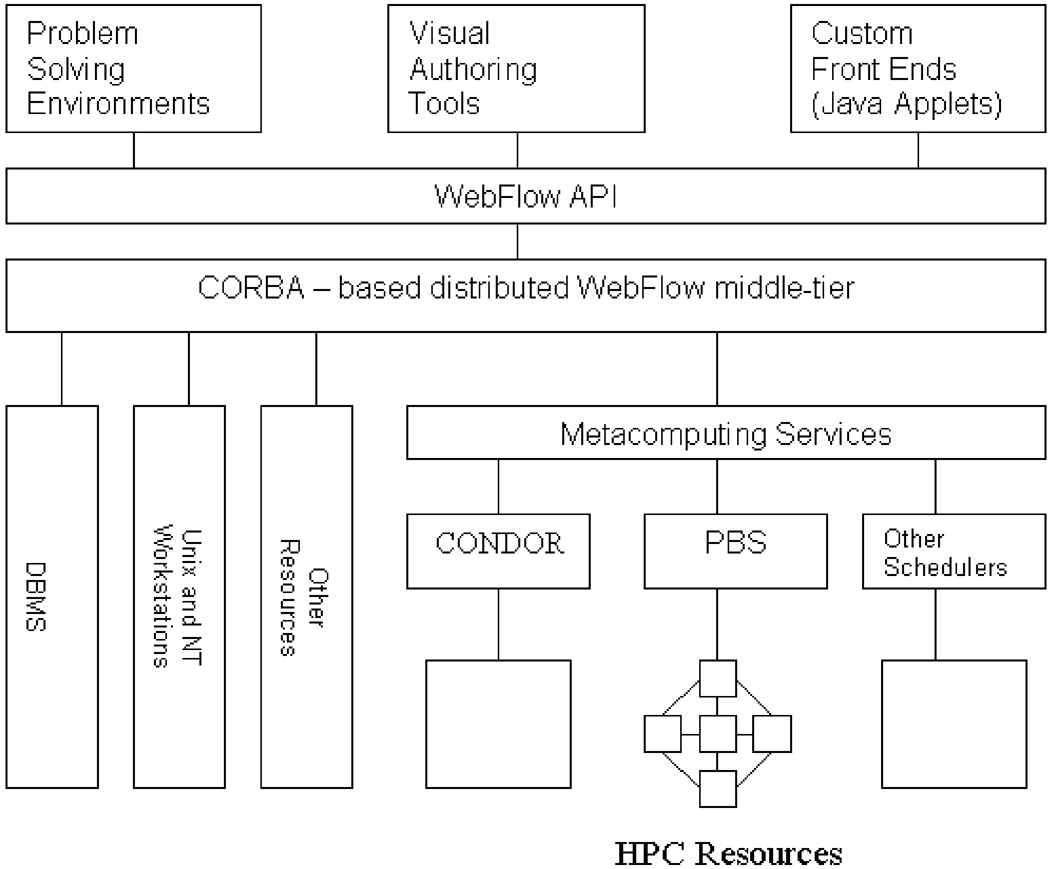


Figure 1: Gateway system architecture

recursively and may comprise an arbitrary number of subtasks. The lowest level, or atomic, task corresponds to the atomic operation in the middle tier, such as instantiation of an object, or establishing interactions between two objects through event binding. In many cases such details should be hidden from the end-user and even the front-end developer, thus the Gateway API provides interfaces to higher-level functionality, such as submitting a single job or making a file transfer.

When specifying a task, the user does not have to specify the resources to be used to complete the task, but instead may specify requirements that the target resource must satisfy in order to be capable of executing the job. The identification and allocation of the resources is left to the discretion of the system. Typically, the middle tier delegates it to the metacomputing services (such as Globus) or and external scheduler (such as PBS). Once the resources are identified, the abstract task descriptor becomes a Job Specification.

3.3 Middle Tier

The middle tier is given by a mesh of CORBA-based Gateway servers. A Gateway server maintains the sessions within which the users create and control their applications. The middle-tier services provide the means to control the life cycles of modules and to establish communication channels between them. The modules can be created locally or on remote hosts. In the latter case the task of module instantiation and initialization is transparently delegated to a peer Gateway server on the selected host, and the communication channels are adjusted accordingly. The services provided by the middle tier include methods for submitting and controlling jobs, methods for file manipulating, methods for providing access to databases and mass storage, as well as methods to query the status of the system, status of the users' applications, and their components.

3.4 Gateway Modules

The Gateway modules are CORBA objects conforming to the JavaBeans model. The functionality of a module is implemented either directly in the body of the module or the module serves as a proxy of specific back-end services, such as DBMS or HPCC services.

3.5 Metacomputing Services

Metacomputing services are yet another standard being developed within the DATORR initiative. It specifies all mandatory functionality of a metacomputing system and its interfaces. The Globus toolkit is an example of such metacomputing services. The functionality they provide include secure resource allocation (GRAM), secure file transfer (GASS), metacomputing directory services (MDS), a heartbeat monitor (HBM), and more.

4. Gateway Security Model

The Gateway system supports a three-component security model. The first component is responsible for secure Web access to the system and for establishing the user identity and credentials. The second component enforces secure interactions between distributed objects, including communications between peer Gateway servers and delegation of credentials. The third component controls access to back-end resources.

4.1 Secure Web Transactions: Authentication and Authorization

To implement secure Web transactions we use industry-standard https protocol and commodity secure web servers. The server is configured to mandate a mutual authentication. To make a connection, the user must accept the server's X.509 certificate and must present his or her certificate to the server. A commercial software package (Netscape's certificate server) is used to generate the user certificates, and they are signed by the Gateway certificate authority (CA).

The authorization process is controlled by the AKENTI server [6], which provides a way to express and to enforce an access policy without requiring a central enforcer and administrative authority. Its architecture is optimized to support security services in distributed network environments.

This component of security services provides access for authorized users only to the Gateway server associated with the gatekeeper, following policies defined in AKENTI (and thus representing the stakeholders' interests). Access to peer Gateway servers and access to the back-end services is controlled independently by the other two components of the Gateway security services, and is based on credentials generated during the initial contact with the gatekeeper.

4.2 Secure CORBA: middle tier security

Security features of CORBA are built directly into ORB and are therefore easy to use. Once the user's credentials are established, secure operations on distributed objects are enforced transparently. This includes authorized use of objects, and optional per-message security (in terms of integrity, confidentiality, and mutual authentication).

Access control is based on the access control lists (ACL). These provide the means to define policies at different granularity; from an individual user to groups defined by a role, and from a particular method of a particular object to computational domains. In particular, the role of a user can be assigned according to policies defined in AKENTI. In this way, access to the distributed objects can be controlled by the stakeholders.

In addition, for security-aware applications, the CORBA security service provides access to the user credentials. In this way access to the back-end resources can be controlled by the owners of the resources rather than Gateway system which system merely forwards the credentials.

The CORBA security service is defined, as an interface and the OMG specification is neutral with respect to the actual security technology to be used. It can be implemented on top of PKI technologies (such as SSL), private key technologies (such as Keberos), or it may implement GSS-API, to mention the most popular ones.

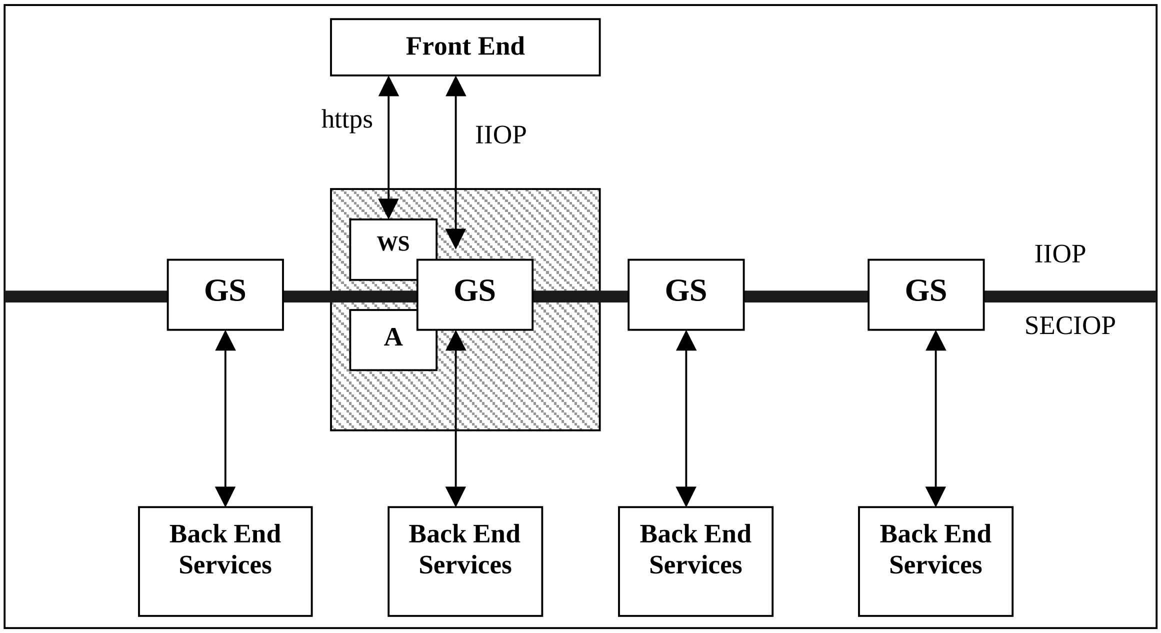
Distributed objects are inherently less secure than traditional client-server systems. Enhanced risk level comes, among other factors, from the fact that objects often delegate parts of their implementation to other objects (which may be dynamically composed at runtime). In this way objects serve simultaneously as both clients and servers. Because of subclassing, the implementation of an object may change over time. The original programmer neither knows nor cares about the changes, therefore the policy of privilege delegation is a very important element of system security. CORBA is very flexible here and supports no delegation model (the intermediary object uses its own credentials), a simple delegation model (the intermediary object impersonates the client), and a composite delegation (the intermediary object may combine its own privileges with those of the client). We follow the composite model. For security-unaware applications, we use the intersection of the client and the intermediary privileges. However, if the application applies its own security measures, we make the initiator's credentials available to it.

4.3 Control of Access to Back End Resources

There are no widely accepted standards for a secure access to resources. Different computing centers apply different technologies: SSH, SSL, Keberos5, or other. The design goal of the Gateway system is to preserve the autonomy of the resources owner to define and implement its security policies. In this respect, we are in a very similar situation similar to that of other research groups trying to provide a secure access to remote resources. Our strategy is to participate in the process of defining standards within DATORR and the common Alliance PKI infrastructure. It seems that the current preference is to build future standards on top of the GSS-API specification (and thus to support simultaneously private and public key-based technologies). The Globus project pioneered this approach, and therefore we use Globus GRAM to provide a secure access to the remote resources. To obtain access to resources available via GRAM a user must present a certificate signed by the Globus CA (currently an additional item of the Gateway user set of credentials).

5. Middle Tier

The Gateway middle tier is given by a network of Gateway servers (GS). A secure access to the system is facilitated by a dedicated gatekeeper server, as shown in Fig.2.



GS: Gateway Server **WS**: secure Web Server **A**: AKENTI server

Fig. 2: the Gateway middle tier architecture

5.1 Gatekeeper Server

The gatekeeper comprises three logical components: a (secure) Web Server, the AKENTI server, and a CORBA-based Gateway server. The user accesses the Gateway system through a portal web page from the gatekeeper web server. The portal implements the first component of gateway security: user authentication and generation of the user credentials that eventually will be used to grant access to resources. The authorization process is controlled by the AKENTI server. For each authorized user, the web server creates a session (that is, it instantiates the user context in the Gateway server, as described below) and gives permission to download the front-end applet. The applet is used to create or restore, run, and control user applications. The applet, using IIOP protocol, communicates directly with the CORBA-based Gateway server.

Currently, we are using a secure Apache web server [7] with servlets support [8]. Unfortunately for WindowsNT, the servlet support is based on a third-party plug-in, and therefore it is difficult to install and configure. We use Apache because it is the server of choice of the AKENTI developers. We also experiment with the secure Jigsaw server [9], which is the W3C reference web server implementation. It is written in Java, and therefore "natively" supports the servlet mechanism. The Jigsaw https support is build on top of the SSL library in Java by the Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology (Austria).

To implement the Gateway server we use the ORBacus [10] (formerly known as OmniBroker) secure ORB, for which we have obtained a free research license. The security services are implemented on top of the IAIK SSL library, the same as that used by Jigsaw.

5.2 Gateway Server

The Gateway server initializes the ORB and several generic CORBA and specific Gateway services. The main functionality of the Gateway server is managing Gateway sessions. A session is established automatically after the authorized user is connected to the gatekeeper by creating a user context. The user context is a container object that stores the user applications. The application is another container object that stores components of the user application. The application component is either a single Gateway module or another, finer-grain application context. This way, the Gateway server can simultaneously manage many sessions, and within each session, the user can define many applications hierarchically composed of many modules.

Each Gateway server maintains a persistent directory of available modules (PMD),
Currently it is a list of modules read from a configuration file at the initialization of the Gateway server, to be replaced by a database. In addition, at run time the Gateway server maintains the directory of active Gateway servers (ASD). The services

provided by these directories are made available to the front end, which allows to us visually select modules (i.e., module name and host) connecting included in an application. Information on the module implementation is available from PMD.

5.3 Life cycle of user modules

Creation of the user module is delegated to the Lifecycle service. The actual instantiation of the module depends on whether it is a local or remote module. A local module factory that runs as a separate process creates local modules. For the remote modules, a local *proxy module* is created followed by copying the application context to the remote Gateway server and instantiating the module on the remote host by its local lifecycle service.

The creation of proxy modules breaks the CORBA object location transparency. As in Java RMI, we make a difference between local and remote objects. There are two reasons to introduce proxy modules. One is that we are providing support to control the behavior of the back-end modules using the front end applet. A direct communication between the (unsigned) applet and a host other than the gateway (from which the applet is downloaded) violates the Java sandbox security model. Employment of signed applets potentially solves this problem; however the use of proxy modules to forward messages results in a more consistent security model. The other reason to use proxy modules is that it gives us a better control over the application container and to simplify the implementation of fault tolerance.

After the local Gateway server gets the reference of the newly instantiated module from the module Factory (local module) or the Proxy object (remote module), the reference is returned to the user. Therefore, the user will have access to his own remote modules through their proxies.

The process of a module creation is finalized by registering the module reference (IOR) under its name within the application and user contexts into the module directory. This operation is performed by the name service.

The reverse, the RemoveModule operation, removes the object reference from the directory and destroys the module. In case of the remote module, its proxy is destroyed as well.

5.4 Interactions between Gateway Modules

The Gateway modules follow the JavaBeans model, and they interact with each other by using JavaBeans methods through event binding, property binding, and vetoable property binding.

With JavaBeans, events are used to communicate information about the changing state of a bean. Events form a core component of the JavaBeans architecture in that they are largely responsible for enabling beans to be plugged together as building blocks in an application builder. Event notification in Java works using method invocation. The object that is a source of an event calls a method on the destination object for one event when the event is triggered. The destination of the message must implement the method (or methods) to be notified when the event occurs. The event object encapsulates all the information about an event.

Event targets are connected to event sources through a registration mechanism. Gateway applications are created dynamically from independently developed Gateway modules. Therefore, we provide support for a dynamical event binding based on the standard CORBA dynamic interface invocation (DII) and dynamic stub invocation (DSI) mechanisms. This is implemented by introducing an event adapter associated with the application context. The adapter maintains a binding table to associate the event sources with the actual event destinations.

6. Related Work

There are several other projects addressed to solving the problem of seamless access to remote resources. A comprehensive list of these is available from the JavaGrande web site [11]. Here we mention the three that are most closely related to this project.

The UNICORE project [12] introduces an excellent model for the Abstract Task Descriptor that most likely will strongly influence the DATORR standard and, consequently, we are taking a similar approach. The UNICORE middle tier is given by a network of Java web servers (Jigsaw). The WebSubmit project [13] implements web access to remote high-performance resources through CGI scripts. Both projects use https protocol for user authentication (as we do), and implement custom solutions for access control. The ARCADE project [14] is aimed at multidisciplinary applications, and its designers intend to use CORBA to implement the middleware. As of now, there is no available description of the ARCADE security model.

7. Summary

Exploiting our experience developing the WebFlow system, we designed a new system, Gateway, to provide seamless and secure access to computational resources at ASC MSRC. While preserving the original three-tier architecture, we re-engineered the implementation of each tier in order to strictly conform to the standards. In particular, we used CORBA and the JavaBeans model to build the new middle tier, which facilitates seamless integration of commodity software components. Database connectivity is a typical example of a commodity software component. However, the most distinct feature of the Gateway system is that we apply the same commodity components strategy to incorporate HPCC systems into Gateway architecture. By implementing emerging standard interface for metacomputing services, as defined by DATORR, we provide a uniform and secure access to high-performance resources. Similarly, by conforming to the Abstract Task Descriptor specification we enable seamless integration of many different front-end visual authoring tools.

The prototype Gateway system is now available [15], and the fully functional version is expected to be deployed by November 1999. This work is collaboration with Ken Flurchick of the Ohio Supercomputer Center and Bill Asbury of ASC, Dayton. The

original Gateway concept was developed with Dana Hoffmann of OSC. However the technology described here was largely developed by the CEWES and ARL DoD High Performance Computing Modernization Program Major Shared Resource Centers through the Programming Environment and Training (PET) activity and by the NCSA Alliance. DoD Modernization contract numbers are: DAHC 94-96-C-0002 with Nichols Research Corp, DAHC 94-96-C-0005 with Nichols Research Corp and DAHC 94-96-C-0010 with Raytheon Systems Company.

References

1. G. C. Fox, W. Furmanski, “High Performance Commodity Computing” in “The Grid. Blueprint for a New Computing Infrastructure”, a book edited by C. Kesselman and I. Foster, Morgan-Kaufmann Publishers, Inc., San Francisco, 1998; Geoffrey C. Fox, Wojtek Furmanski, Hasan T. Ozdemir and Shrideep Pallickara, “High Performance Commodity Computing on the Pragmatic Object Web” , report to be published by RCI (http://tapetus.npac.syr.edu/iwt98/pm/documents/)
2. D. Bhatia, V. Burzewski, M. Camuseva, G. C. Fox, W. Furmanski, G. Premchandran, “WebFlow – A Visual Programming Paradigm for Web/Java based coarse grain distributed computing”, Concurrency: Practice and Experience, 9, 555-578 (1997) (http://tapetus.npac.syr.edu/iwt98/pm/documents/)
3. Globus Metacomputing Toolkit, home page: http://www.globus.org
4. E. Akarsu, G. C. Fox, W. Furmanski, T. Haupt, “WebFlow - High-Level Programming Environment and Visual Authoring Toolkit for High Performance Distributed Computing”, in proceedings of Supercomputing ’98 (http://www.npac.syr.edu/users/haupt/WebFlow/papers/SC98/INDEX.HTM)
5. Java Grande Forum, home page: http://www.javagrande.org
6. S. S. Mudumbai, W. Johnston, M. R. Thompson, A. Essiari, G. Hoo, K. Jackson, Akenti – A Distributed Access Control System, home page: http://www-itg.lbl.gov/Akenti
7. Apache http server, home page: http://www.apache-ssl.org
8. Java Apache Project, home page: http://java.apache.org
9. Jigsaw http server, home page: http://www.w3.org/Jigsaw
10. Object Oriented Concepts, Inc., ORBacus SSL, home page: http://www.ooc.com/ssl/
11. DATORR WebSite maintained by Gregor von Laszewski: http://www.fp.mcs.anl.gov/~gregor/datorr/datorr.html
12. UNICORE: Uniform Access to Computing Resources, home page: http://www.fz-juelich.de/unicore
13. WebSubmit: A Web-based Interface to High-Performance Computing Resources, home page: http://www.itl.nist.gov/div895/sasg/websubmit/websubmit.html
14. ARCADE, home page: http://www.icase.edu:8080
15. Gateway project home page http://www.npac.syr.edu/users/haupt/WebFlow/demo.html