Abstract

The goal of this project was to prototype a Web based computational and collaboratory environment that would act as a commodity technologies based front-end or Gateway to larger HPC systems such as ANL LabSpace. The original name – WebSpace – proposed for such a system was taken during the course of this project by Silicon Graphics and used for their Web authoring environment. To avoid confusion, we changed the name of this project’s system to WebFlow.

WebFlow system prototype was successfully developed by this project and then tested, demonstrated and disseminated. The system followed the 3-tier architecture with the central control and integration WebVM layer in tier-2 (middleware), interacting with a Web based visual graph editor applets in tier-1 (front-end) and the legacy systems or dedicated HPC modules in tier-3 (back-end). WebVM (Web Virtual Machine) was given by a mesh of Java Web servers that managed a collection of visually composable distributed computational modules. All system control structures were implemented as URL-addressable servlets that collectively enabled Web browser-based authoring, monitoring, publication, documentation, collaboration and software distribution for distributed computing. WebFlow/WebVM was found to be a promising programming paradigm and coordination model by other government funded HPC projects, both within DOE and other agencies. These projects adapted WebFlow and deployed it as part of larger HPC systems.

In this final report, we briefly outline the R&D process that led us to WebFlow concepts, we summarize the WebFlow architecture, we illustrate the prototype system in action on a suite of demos and test case applications, and we describe selected WebFlow success stories in other HPC projects (DoE, NSF, DoD). The latter include WebFlow interface to Globus (DoE, NSF) and WebFlow integration with the DoD High Performance Modeling and Simulation. A set of papers published based on the work performed or started in this project is attached at the end of the report.
This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.
DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.
1. Introduction

At NPAC, we pioneered in early-to-mid '90s the concept of Web based Computing. We called it WebWindows and envisioned as a Web based computational environment with Web browsers acting as open standards based 'windows' into a computational mesh of Web servers. These servers, working in concert within a Web Virtual Machine coordination framework, would offer in turn a universal access to legacy systems and dedicated HPC modules in the backend, wrapped as visually composable modules (or Web Services).

The goal of this project was to quantify the WebWindows concepts in the form of a working prototype that could act as Web Gateway for larger HPC systems such as ANL LabSpace. The original name – WebSpace – proposed for such a system was taken during the course of this project by Sillicon Graphics and used for their Web authoring environment. To avoid confusion, we changed the name of this project's system to WebFlow.

This project involved three natural stages:

1. research and experimentation with various emergent computational Web technologies ('96-'97);
2. design and development of prototype Web Flow system ('97-'98);
3. deployment, adaptation and integration of WebFlow with other HPC systems ('98-).

These stages are addressed in the following Chapters of this document: Stage 1 is discussed in Chapter 2: WebFlow/WebVM Concepts; Stage 2 is covered in Chapter 3: WebFlow Architecture and Chapter 4: WebFlow Prototype Demonstrations; and Stage 3 is summarized in Chapter 5: WebFlow Applications.

2. WebFlow/WebVM Concepts

In the period '95 – '97, the expressive power of Java attracted developers and we observed an explosion of first generation Java systems on the Internet. Examples included: NCSA Habanero for synchronous collaboratory; dynamic HTTP servers such as Jigsaw from MIT/W3C or Jeeves from JavaSoft; Marimba's Castanet and Bongo trying to establish a new pure Java based Web-like framework; Caltech Infospheres, IBM aglets for intelligent agents based computing; and many others.

At NPAC, we were closely monitoring this 'bleeding edge' of interactive Web and we observed that although these new systems offer attractive capabilities, the first generation Java software was still difficult to customize, repackage or reuse. The reason was that a Java class was too small, too fine grain encapsulation unit and hence reusing a Java package required usually a detailed understanding of a large number of its tightly interwoven classes.
Our goal within the WebWindows vision was to provide a coarser grain packaging model and the associated user-friendly authoring framework for distributed applications on the Web. We believed that we should build on top of the established standards such as HTTP, HTML and Java, and hence we adopted Java Web server as a base runtime and coordination node of a distributed Web system. Dataflow model, already proven effective by previous systems such as AVS, Khoros, CODE, HeNCE and others, seemed to be a natural coordination framework to extend the client-server Web information model towards multi-server distributed computation systems.

Along this line of arguments, we developed the concept of a runtime environment given by a mesh of Web Java servers that coordinate distributed computation represented as a set of channel-connected coarse grain Java modules. Modules were thin veneer Java interfaces so that any chunk of Java could be easily modularized and connected to other modules via suitable communication ports, acting as terminals of point-to-point dataflow channels. Modules were to run asynchronously, they were mobile i.e. they could be instantiated on any WebVM server, and they would communicate by exchanging Java objects along their dataflow channels.

Aspects of such emergent architecture could be already found in the first generation Java systems, analyzed in the initial phase of the WebFlow project. For example: Jigsaw/Jeeves developed the concept or resources/servlets as control encapsulation units; Infospheres developed portlets/mailboxes as terminals for communication channels; Habanero already was a multi-server system (but focused only on real-time collaboration), and so on.

We initiated the WebFlow/WebVM design process by experimenting with existing systems. In the period '95-'96, we evaluated a suite of first Java systems including Aglets, Habanero, Infospheres, Jeeves, Jigsaw, JSDA, and others. One of the early decisions we made was that rather than developing custom Java servers from scratch as in Habanero or Infospheres, we should build on the emergent Java Web Server base by adding new services and maintaining them within the Web Java server addressing space. Such organization facilitated management and offered natural, Web-browser based monitoring, publication and distribution support for the Web software.

3. WebFlow/WebVM Architecture

Overview Our prototype WebVM was given by a mesh of Jeeves servers, running servlets that manage and coordinate distributed computation. Atomic encapsulation units of WebVM computation were called WebFlow modules (a more contemporary name now is Web Services) and they would communicate by sending objects along channels attached to module ports. Unlike management servlets which were usually persistent and application independent, our modules were more transient and could be dynamically created, connected, scheduled, run, relocated and destroyed by servlets. WebFlow was a particular programming paradigm implemented over WebVM and given by a dataflow programming model (other models under experimentation included data parallel, collaboratory, and televirtual paradigms).
WebFlow application was given by a computational graph, visually constructed and edited by end-users via Java applet based authoring tools. Modules were written by module developers who, by the very nature of Web distributed computing, had only limited knowledge of the system on which the modules will run. The central design requirement therefore was that the module developers should 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.

**WebVM Middleware** The WebFlow management system was designed to hide these management and coordination functions from the developers, allowing them to concentrate on the modules being developed. WebFlow management was implemented in terms of the following three servlets: Session Manager (in charge of executing all the actions the user performs on the front end), Module Manager (in charge of running modules on demand), and Connection Manager (in charge of creating connections between the modules). These servlets were URL addressable and could offer dynamic information about their services and current state. Each of them could also communicate with each other via WebFlow ports and channels.

Fig 1 illustrates the overall WebFlow/WebVM concepts based on 3-tier Web architecture with a mesh of WebFlow middleware servers. Fig. 2 exposes the internal structure of one WebFlow server with the three base management servlets employed in setting up and managing WebFlow operation. Responsibilities of individual managers illustrated in action in Fig 2 are as follows:
Session Manager receives graph specification from the editor applet, creates an image of the whole compute-web using module proxy objects called ModuleRepresentation, decides on the compute-web decomposition strategy, and notifies Module Manager about local modules to be instantiated.

Module Manager starts and maintains ModuleWrapper threads than run Modules. Each module, when created, notifies Connection Manager about the connectivity required by this module Ports, and waits for the connections to be established.

WebFlow channels connecting two module Ports are formed dynamically by the corresponding Connection Managers: Sockets returned by their 'accept' and 'connect' calls are passed to the appropriate ports. After all ports of a module receive their requested sockets, the module notifies the Module Manager and is ready to participate in the dataflow operations.

**Figure 3:** Front-end authoring and middleware management layers of the WebFlow system: visual icons become middleware modules; inter-icon connections become inter-modular communication channels. Modules can be associated with visualization or steering applets that render module feedback or control module parameters during runtime.

**Figure 4:** Middleware management and back-end computational layers of the WebFlow system: middleware modules act either as autonomous middleware computational components or as wrappers / proxies for back-end modules (such as legacy systems or HPC boxes.)

Figs 3 and 4 illustrate the whole WebFlow authoring and runtime process. Fig. 3 exposes the visual authoring applet and Java Server middleware components. Fig. 4 illustrates the inner workings of the WebFlow management system, responsible for module instantiation and connectivity. Connection Managers in two Java Web Servers exchange port / connectivity information to establish a stable socket connection between the modules they manage.

WebFlow offers a well-defined API for module developers that hides the communication details in terms of port and module abstractions. The main abstractions include Ports (to handle connectivities), Modules (to handle computation) and MetaModules (to handle information about system capabilities). The module API is very simple. Each module implements a specific WebFlow Java interface, metamodule. In practice, the module developer needs to implement just three methods: initialize, run, and destroy. The initialize method registers the module itself and its ports with the Session Manager and it establishes the
communication between itself and its frontend applet. The run method implements the desired functionality of the module, whereas the destroy method performs clean-up after the processing is completed such as closing all socket connections, releasing the module resources etc.

![Diagram](image)

**Figure 5:** A WebFlow server, managing a heterogeneous collection of modules for various computational domains, including collaboration, database management and high performance computing.

**Figure 6:** A WebFlow demo application, presented as Supercomputing '97: a set of scientific visualization modules, acting as runtime probes for a HPC simulation in the domain of gravitational waves.

**Browser Front-End** WebFlow front end offers a tool for visual authoring of computational dataflow graphs that integrate the existing public domain software modules. It is based on highly intuitive visual icons and click-and-drag design metaphors which are hiding the inherent complexity of the WebFlow system. In the prototype implementation of the front end we used the UCI's Graph Editing Framework (GEF) as a basis to develop the front end of the WebFlow system. GEF supported the basic graph editing mechanisms, was well structured with cleanly decoupled layers, and it was naturally extensible.

A GEF based WebFlow front end was implemented as an applet that resided in the top level layer of the system, and it created and maintained a connection with the Session Manager in the middleware. The user created a computational graph from modules as building blocks, simply by selecting the corresponding icons from a palette of available modules in the system and by inserting them visually into the graph. Multiple instances of a specific module could be created and their internal states, locations and connections were completely independent.

After the modules were inserted as nodes in the graph, the applet requested their initialization from the WebFlow management. After the initialization was completed, the back end replied to the applet, bringing information about the interface of the selected module. The applet was building and storing the representation of the graph, maintaining the information just about its visual representation. The information about the actual modules and their mapping on real machines was stored by the WebFlow management system in the middleware.
In the same fashion as the modules, the connections between modules are created in the visual authoring mode, simply by drawing suitable connection lines between module icons. Individual modules and/or their connections can be also visually removed from the graph, which results in deleting them from the computational mesh in the middleware/backend.

After a computational graph is created it can be executed. The computation can be monitored through the input/output modules that are inserted in the graph. The execution of a computational graph can generate variety of feedback patterns, ranging from just producing final results of a complex computation, to periodic performance visualization and system monitoring or steering modes, to real-time interactive display modes. Various such feedback or control modes can be implemented by suitable applets that attach to selected modules and allow for two-way interaction with the user during the runtime WebFlow sessions.

**WebFlow Back-End** WebFlow modules can be pure Java or native (C/C++) and reside fully in the Java Server middleware or they can act as proxies to back-end (typically UNIX domain) modules, responsible for large data storage (e.g. RDBMS) or HPC computation. Both types of modules are illustrated in Figs. 2 and 4. As part of WebFlow prototype development effort, we constructed a set of test modules, both in Java middleware and UNIX backend domains. Some examples of such test or demo applications are described and illustrated in the Chapter 4: WebFlow Prototype Demonstrations. More significant WebFlow modules, responsible for large scale backend computation, are discussed in Chapter 6: WebFlow Applications.

**Summary** WebFlow 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 places on different platforms and networks. New distributed applications can be composed dynamically from reusable components just by clicking on visual module icons, dragging them into the active WebFlow editor area, and linking by drawing the required connection lines. Individual modules are typically represented by visualization, control, steering or collaboration applets, and the system also offers visual monitoring, debugging and administration of the whole distributed application and the underlying metacomputing resources. New applications, created within the WebFlow framework will likely follow a natural modular design in which one accumulates in the first phase of a project a comprehensive problem domain specific module library. In the next phase, one would explore the computational challenges of the problem domain in a visual interactive mode by composing the optimal solutions in a sequence of on-the-fly trial applications. Successful multi-module solutions constructed this way can be scripted, saved and reused or embedded as macro-modules in yet larger scale Web based distributed computations.

### 4. WebFlow Prototype Demonstrations

A prototype implementation of WebFlow, based on the architecture described in the previous Chapter, was constructed at NPAC in '96-'97 and tested in a set of Web based distributed computing demonstrations. A generic nature of WebFlow computation, based on a loosely coupled collection of computational or communication modules, allowed us to rapidly prototype test applications in variety of computational domains.
Fig 5 illustrates how WebFlow can be used to support a spectrum of computational tasks ranging from collaboration to database management to high performance computation. Specific WebFlow test or demo projects developed in period '97-'98 included: an image processing system; a collaboratory environment (chat, whiteboard); a statistical physics (Pott's spin model) simulation; a scientific visualization toolkit; a visual front-end for High Performance Fortran Interpreter.

The attached set of screendumps illustrate WebFlow prototype in action for various selected computational domains. This first row in Fig. 7 illustrates screendumps from a chat application – here WebFlow visual authoring is used to setup the content modules (members) and topology (connectivity between members) of a chat room or more some more elaborate (e.g. whiteboard) collaboratory environment.

The left screens in the first and the second row in Fig 7. illustrates WebFlow based image processing demo. This application offers a library of image processing modules such as edge detection, contrast enhancement, RGB filtering etc. By suitable superposition of such elementary signal processing functions one can build a fully-fledged image processing macro-modules that are naturally distributed and Web accessible.

The bottom right image in Fig. 7 illustrates WebFlow applied to mange and visualize a statistical physics simulation of a Pott's spin system. This Monte Carlo simulation is performed using High Performance Fortran module, and the real-time visualization is provided by a 2D graphics/imaging module.
Screendumps in Fig. 8 illustrate various stages of scientific visualization and computational backend management for a High Performance Fortran interpreter module, performing a gravitational wave simulation.

Figure 8: Supercomputing '97 demonstration of WebFlow application for gravitational wave simulation: WebFlow computational graph shown in the first (upper left) image includes HPF interpreter based gravity solver and a suite of visualization modules that are illustrated and calibrated the following screendumps.

Screens in the second row in Fig. 8 illustrate the process of assembly and calibration of the individual simulation controls, including time stepping, synchronization and various visualization filters. Fig 6 and the top left image in Fig. 8 illustrate the actual gravity simulation, including a suite of visualization and control modules and the central solver of Einstein gravity equations, implemented in High Performance Fortran interpreted environment, wrapped as a WebFlow back-end module.

5. WebFlow Applications

We describe here briefly two representative WebFlow applications: WebFlow/Globus Integration and HPC Forces Modeling and Simulation. In both cases, the WebFlow prototype was adapted, fine-tuned and deployed as part of the large scale HPC system, funded and developed by an independent project.

5.1 WebFlow/Globus Integration.

The goal of this project was to explore WebFlow as a Web based visualization front-end and middleware management framework for the ANL Globus metacomputing toolkit. The overall 3-tier architecture of WebFlow/Globus integration is illustrated in
This project developed a dedicated WebFlow visual front end, illustrated in Fig. 11. We used the following Globus components in this project: Nexus as the high performance communication library, MDS (Metacomputing Directory Services) for the resource identification, GRAM (Globus Resource Allocation Manager) for secure allocation and scheduling of the resources, GASS package (Global Access to the Secondary Storage) for high performance, secure data transfer, and RIO (Remote Input/Output) library for parallel data file systems. WebFlow interacted with the Globus via GRAM gatekeeper. A dedicated WebFlow module served as a proxy of the gatekeeper client, which in turn was sending requests to GRAM.

As a driving application for the WebFlow/Globus integration we selected Quantum Simulations conducted by the NCSA Team. 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 Fig. 9.

This example meta-application demonstrates strength of our WebFlow approach. The WebFlow editor provides an intuitive environment to visually compose the chain of data-flow computations from preexisting modules. The modules encapsulate many different classes of applications: from massively parallel to custom developed auxiliary programs to commodity commercial ones (such as DBMS or visualization packages). The seamless integration of such heterogeneous software components is achieved by employing distributed objects technologies in the middle tier.
The high performance part of the backend tier is implemented using the GLOBUS toolkit. In particular, we use MOS (metacomputing directory services) to identify resources, GRAM (globus resource allocation manager) to allocate resources, including mutual, SSL-based authentication, and GASS (global access to secondary storage) for a high performance data transfer. The high performance part of the backend is augmented with a commodity DBMS (servicing Permanent Object Manager) and LDAP-based custom directory service to maintain geographically distributed data files generated by the Quantum Simulation project. The diagram illustrating the Webflow implementation of the Quantum Simulation is shown in Fig. 12.
5.2 WebFlow for DoD HPC Modeling and Simulation

Another large scale application domain where we adapted, fine-tuned and deployed WebFlow as part of the full HPC solution was Forces Modeling and Simulation – one of the CTA areas of the DoD HPC Modernization Program. The essential backend technology for modular distributed computing in this domain was provided by HLA (High Level Architecture). HLA is a language-independent object-based distributed software architecture for simulation reusability and interoperability that is now being enforced DoD-wide across all individual M&S programs, systems and simulation paradigms, including both real-time (DIS) and logical time (event-driven) management models. HLA views distributed simulation as a federation of coarse grain opaque semi-autonomous entities called federates that govern locally and independently their simulation objects and that conform strictly to some global federation rules, specifying the information exchange policy across the federation. The associated Run-Time Infrastructure (RTI) offers the software bus services available to the HLA-compliant federates and including Federation, Object, Declaration, Ownership, Time and Data Distribution Management. We illustrate the overall organization of RTI in Fig 13.

Within the FMS project, we extended the WebFlow management architecture by standardizing it around the CORBA distributed object model. The resulting new server technology – JWORB (Java Web Object Request Broker) was used as the base for WebFlow adaptations in the FMS area. JWORB architecture, based HTTP and IIOP (CORBA) compliant multi-protocol Java Web server is outlined in Fig. 15. On top of JWORB, we developed WebHLA framework that offered open implementation of HLA in terms of a suite of emergent object standards for the Web based distributed computing. WebHLA was an interactive 3-tier environment that included: a) DMSO HLA architecture and our WebFlow/JWORB based Object Web RTI implementation in the middleware; b) Web/Commodity front-ends (such as WebFlow authoring); and c) Customer and application specific back-end technologies (ranging from legacy...
systems such as relational databases to HPC modeling and simulation modules). Object Web RTI architecture is outlined in Fig. 16. RTI federates are WebFlow modules, managed by JWORB as distributed CORBA objects. WebFlow was used in WebHLA both in the authoring and runtime domains. Fig 14 illustrates a suite of WebFlow modules involved in the HLA Federation Development Process (FEDEP), following DMSO standards. Runtime modules included JWORB based OWRTI, legacy DIS simulations such as ModSAF, DIS/HLA bridge (JDIS), SimVis front-ends, logger/playback databases (PDUDB) and HPC simulation backends (Parallel CMS).

The WebHLA framework outlined above was applied in a large scale FMS project conducted by NPAC that developed Parallel and Metacomputing CMS (Comprehensive Minefield Simulation) based on the CMS simulator from Ft. Belvoir. This effort included converting the CMS system from the DIS to HLA framework, constructing scalable Parallel CMS federate for Origin2000 and linking it with ModSAF vehicle simulators and other utility federates towards a Metacomputing HPC CMS federation.
The overall configuration of such Metacomputing CMS environment that span three geographically distributed DoD HPC laboratories and utilized most of the WebHLA tools and federates discussed above is shown in Fig 17. ModSAF, JDIS and SimVis modules (such as a Windows desktop battlefield visualizer in Fig. 18) were typically running on a workstation cluster at NPAC in Syracuse University. JWORB/OWRTI based Federation Manager (marked as RTI in Fig 17) was typically running on Origin2000 at ERDC in Vicksburg, MS. Parallel CMS federate was typically running on Origin2000 at NRL in Washington, DC. The central management node running continuously (at ERDC) was WebFlow/JWORB/OWRTI based Federation Manager. A typical dataflow of a WebFlow managed Metacomputing CMS simulation included the following elements. After the Parallel CMS was started at NRL, it joined distributed federation (managed at ERDC) and automatically activated the PDUDB playback server at NPAC that started to stream vehicle PDUs to JDIS which in turn converted them to HLA interactions and sent (via RTI located at ERDC) to Parallel CMS federate at NRL. Each such event, received by node 0 of Parallel CMS was multicast via shared memory to all nodes of the simulation run and used there by the node CMS programs to update the internal states of simulation vehicles. Inner loop of each node CMS program was continuously tracking all mines scattered into this node against all vehicles in search of possible explosions.

Having constructed fully scalable WebFlow/JWORB based Parallel CMS federate and having established a robust Metacomputing CMS experimentation environment, we conducted a set of experiments with wide area distributed large scale FMS simulations, using CMS as the application focus and testbed. We were able to successfully distribute large minefields of millions of mines over several Origin2000 machines in various DoD labs using domain decomposition, followed by the scattered decomposition of each minefield domain over the nodes of a local parallel system.

WebFlow infrastructure, underlying our WebHLA framework, played essential role in conducting such complex distributed experiments. This included both the middleware management provided by JWORB middleware as well as the visual authoring and visualization provided by the WebFlow front-end. The WebFlow/WebHLA based Metacomputing CMS simulation described here was in fact the first ever Web based HLA compliant simulation, and yet developed and successfully operated by a small academic R&D team. The crucial technology that enabled this successful metacomputing application was the WebFlow/JWORB based management, visualization, monitoring and steering environment.
WebFlow Related Publications


