IBuilding Web/Commodity based Visual Authoring Environments for Distributed Object/Component Applications - A Case Study using NPAC WebFlow System

E. Akarsu, G. C. Fox, W. Furmanski[1]*, T. Haupt, H. Ozdemir, Z. Odcikin Ozdemir and T. Pulikal*
Northeast Parallel Architectures Center, Syracuse University, Syracuse, NY

Submitted to Middleware '98. IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing, The Lake District, UK, 15-18 Sept. 1998

Summary

We present here an approach towards visual authoring environments for Web/Commodity based distributed object/omponentware computing using the WebFlow system under development at NPAC as a case study. WebFlow is a 3-tier Java based visual dataflow system with applets based authoring, visualization and control front-ends, and with servlets based middleware management of backend modules that wrap legacy codes such as databases or high performance simulations. We summarize here the WebFlow architecture, we describe a set of demos and early applications in various areas of distributed computing (including imaging, collaboration, condensed matter physics and military wargaming simulations), and we outline the next phase design, based on lessons learned in the current prototype. New WebFlow uses JWORB (Java Web Object Request Broker) middleware and employs WOMA (Web Object Management Architecture) methodology to establish a testbed for testing, evaluating and integrating the emergent componentware standards of CORBA, DCOM, Java and W3C/WOM.

Contents

1. Contact information: Wojtek Furmanski, 201 Physics Bldg., Syracuse University, Syracuse, NY 13244-1130, voice (315) 443-3891, fax (315) 443-9103, e-mail furm@npac.syr.edu, url http://www.npac.syr.edu/projects/webspace/doc/middleware98/

1.0 Introduction

World-Wide Web took the computing world by storm in '94, with the initial impact on technologies for the broad dissemination of (hyper)textual information. Local computational extensions of the early Web model were provided by the CGI protocol and exploited mainly for building on-site Web/Database interfaces. In principle, one could adress CGI+HTTP based distributed computing already at this time and in fact our early Web computing ideas such as WebWork [1] were created in this context of early Web technologies - but only after the onset of Java in '95 the Web was fully enabled to start addressing non-trivial distributed computing frameworks and applications.

At NPAC, our background and expertise comes from the high performance computing domain and hence our approach to Web computing is currently focused on domains such as metacomputing, linking coarse grain dataflow modules, scientific visualization etc. We view Web as a promising new platform and a pervasive base for what we call High Performance Commodity Computing [2]. Strategies for building HPCC on top of commodity software are being now explored by the HPCC community [3]. We describe here a specific high-level programming environment developed by NPAC - WebFlow [4][5] - that addresses these issues and offers a user friendly visual graph authoring metaphor for seamless composition of world-wide distributed high performance dataflow applications from reusable computational modules.

Design decisions of the current WebFlow were made and the prototype development was started in '96. Right now, the system is reaching some initial stability and is associated with a suite of demos or trial applications which illustrate the base concepts and allow us to evaluate the whole approach and plan the next steps for the system evolution. New technologies and concepts for Web based distributed computing appeared or got consolidated during the last two years such as CORBA, RMI, DCOM or WOM. In Chapter 8, we summarize our ongoing work on the new WebFlow framework [6][7][8][9] under development that addresses support for and integration of these competing new distributed object and componentware technologies towards what we call Pragmatic Object Web [10]. In Chapter 3-7, our main focus is on presenting the current WebFlow system, its applications and the lessons learned in this experiment. While the implementation layers of the current (Chapters 3-7) and the new (Chapter 8) WebFlow models are different, several generic features of the system are already established and will stay intact while the implementation technologies are evolving. Chapter 2 provides an overview of the system vision and goals which exposes these stable generic characteristics of WebFlow.

2.0 WebFlow Vision/Goals

Our main goal in WebFlow design is to build a seamless framework for publishing and reusing computational modules on the Web so that the end-users, capable of surfing the Web, could also engage in composing distributed applications using WebFlow modules as visual components and WebFlow editors as visual authoring tools. The success and the growing installation base of the current Web seems to suggest that a suitable computational extension of the Web model might result in such a new promising pervasive framework for the wide-area distributed computing and metacomputing.

In WebFlow, we try to construct such an analogy between the informational and computational aspects of the Web by comparing Web pages to WebFlow modules and hyperlinks that connect Web pages to inter-modular dataflow channels. WebFlow content developers build and publish modules by attaching them to Web servers. Application integrators use visual tools to link outputs of the source modules with inputs of the destination modules, thereby forming distributed computational graphs (or compute-webs) and publishing them as composite WebFlow modules. Finally, the end-users simply activate such compute-webs by clicking suitable hyperlinks, or customize the computation either in terms of available parameters or by employing some high-level commodity tools for visual graph authoring.

New element of WebFlow as compared with the current "vertical" instances of the computational Web such as CGI scripts, Java applets or ActiveX controls is the "horizontal" multi-server inter-modular connectivity, specified by the compute-web graph topology and enabling concurrent world-wide data transfers, either transparent to or customizable by the end-users depending on their preferences. Some examples of WebFlow computational topologies include:

- *ring* - post-processing an image by passing it through a sequence of filtering (e.g. beautifying) services located at various Web locations;

- *star* - collecting information by querying a set of distributed databases and passing each output through a custom filter before they are merged and sorted according to the end-user preferences;
- *(regular) grid* - a large scale environmental simulation which couples atmosphere, soil and water simulation modules, each of them represented by sub-meshes of simulation modules running on high performance workstation clusters.
- *(irregular) mesh* - a wargame simulation with dynamic connectivity patterns between individual combats, vehicles, fighters, forces, environment elements such as terrain, weather etc.

When compared with the current Web and the coming Mobile Agent technologies, WebFlow can be viewed as an intermediate/transitional technology - it supports a single-click automation/aggregation for a collection of tasks/modules forming a compute-web (where the corresponding current Web solution would require a sequence of clicks), but the automation/aggregation patterns are still deterministic, human designed and manually edited (whereas agents are expected to form goal driven and hence dynamic, adaptable and often stochastic compute-webs).

3.0 Current WebFlow Software Architecture

Current WebFlow is based on a coarse grain dataflow paradigm (similar to AVS or Khoros models) and it offers visual interactive Web browser based interface for composing distributed computing (multi-server) or collaboratory (multi-client) applications as networks (or compute-webs) of Internet modules.

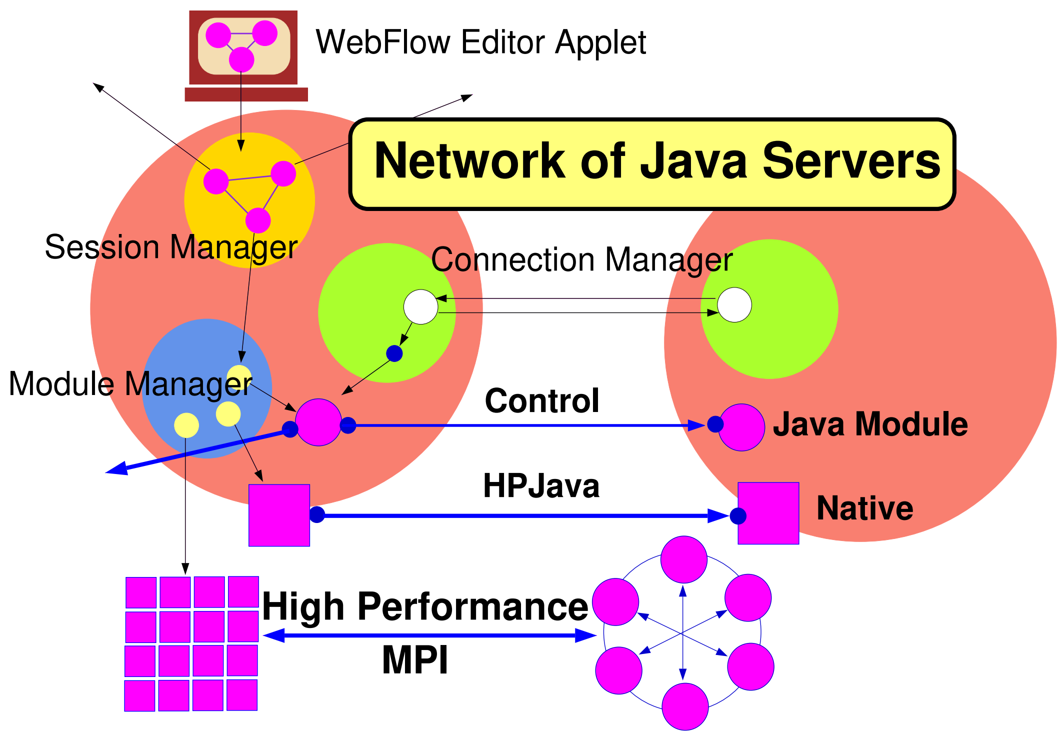

Fig. 1. Overall Architecture of the 3-tier WebFlow model with the visual editor applet in tier-1, a mesh of Java Web Servers in tier 2 (including WebFlow Session Manager, Module Manager and Connection Manager servlets), and (high performance) computational modules in tier-3.

WebFlow front-end editor applet, constructed by extending the GEF (Graph Editing Framework) package from UCI [11], offers intuitive click-and-drag metaphor for instantiating middleware or backend modules, representing them as visual icons in the active editor area, and interconnecting them visually in the form of computational graphs, familiar for AVS or Khoros users.

WebFlow middleware is given by a mesh of Java Web Servers from JavaSoft, custom extended with servlet based support for the WebFlow Session, Module and Connection Management. WebFlow modules are specified as Java interfaces to computational Java classes in the middleware or wrappers (module proxies) to backend services.

To start a WebFlow session over a mesh of the WebFlow enabled Java Web Server (JWS) nodes, user specifies URL of the Session Manager servlet, residing in one of the server nodes. The server returns the WebFlow editor applet to the browser and registers the new session. After a connection is established between the Editor and the Session Manager, the user can initiate the compute-web editing work. WebFlow GUI includes the following visual actions:

- *Selecting a module* from the palette and placing its icon in the active editor area. This results in passing this module tag to the Session Manager that forwards it to the Module Manager. Module Manager instantiates the module and it passes its communication ports to the Connection Manager.
- *Linking two visual module icons* by drawing a connection line. This results in passing the connected modules tags to the Session Manager, and from there to the Connection Managers in charge of these module ports. WebFlow channels are formed dynamically by Connection Managers who create the suitable socket connections and exchange the port numbers. After all ports of a module receive their required sockets, the module notifies the Module Manager and is ready to participate in the dataflow operations.
- Pressing the Run button to *activate the WebFlow computation*
- Pressing the Destroy button to *stop the WebFlow computation* and dismantle the current compute-web.

WebFlow Module is a Java Object which implements webflow.Module interface with three methods: init(), run() destroy(). The init() method returns the list of input and output ports used to establish inter-modular connectivity, and the run() and destroy() methods are called in response to the corresponding GUi actions described above.

4.0 Associated NPAC systems: DARP, SciVis

Independently of the Webflow system, other web-based system has been developed at NPAC. Two of them, the Scientific Visualizations System (SciVis) [12], and the Data Analysis and Rapid Prototyping (DARP) [13] systems are of particular interest in the context of the work presented in this paper. As it is described in Section 5 below, we take them as commodity software components, and encapsulate them as WebFlow modules. In this way these independent packages become elements of larger applications built visually using the WebFlow editor.

The SciVis system is a portable scientific visualization package for one-, two-, and three-dimensional data sets, developed entirely in Java. With a very rich, and user extensible, set of data filters, and full support for a collaborative use it is a very powerful tool for a rapid data analysis. The architecture of the system follows the client-server model. The server is implemented as a Java application that typically runs on the user's workstation. The client feeds the data though a socket connection. An intuitive API provided with the system allows for implementation of the client applications in any language. In particular, the system provides support for Fortran, C/C++ and Java programs to connect to the server via function calls. The SciVis is being used by several research groups, including the Binary Black Hole Grand Challenge Alliance [14].

The DARP system integrates compiled and interpreted HPF (High Performance Fortran) creating a powerful application development environment targeted for high performance parallel and distributed systems. With a debugger-like interface implemented as a Java Applet the system gives the user a full control over the application at the runtime. The detailed description of the system can be found elsewhere [13]. In this paper context, the most important feature of the DARP system is an interactive access to distributed data. This is possible thanks to instrumentation of the code that is performed automatically by DARP so that at runtime the user can then set an action point at an arbitrary place within the code. Once the action point is reached, a predefined dynamically linked proxy function is called. The execution of the compiled code is resumed as soon as the function call is completed. For the WebFlow demonstrations we used the function that extracts the data and sends them to the input port of a Webflow module. In addition, through the DARP front-end, the data can be modified using either native HPF commands or dynamically linked computational modules.

Consistently with our HPcc strategy, the DARP system implements a three-tier architecture: the Java front-end holds proxy objects produced by an HPF front-end operating on the back-end code. These proxy objects can be manipulated with an interpreted Web client interacting dynamically with compiled code through a typical tier-2 server (middleware). Although targeted for HPF back-end, the system's architecture is independent of the back-end language, and can be extended to support other high performance languages.

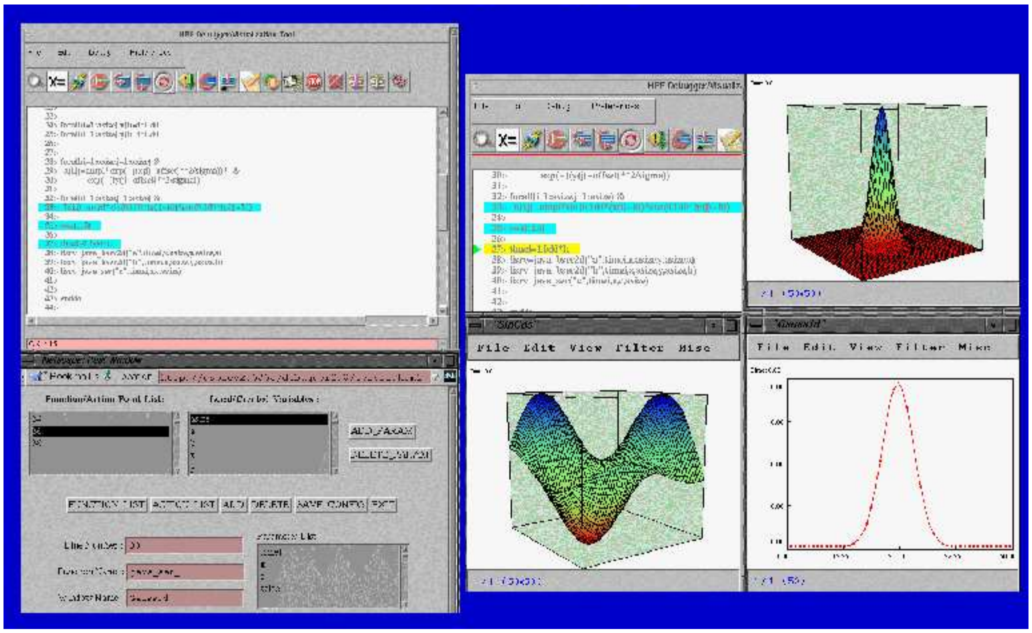

Fig. 2. A screendump from an interactive visual debugging session of an HPF application, using the DARP editor appplet and the SciVis visualization system.

5.0 Trial Application Domains

The WebFlow model is most adequate for computational domains with a set of well defined coarse grain component tasks and with a rich variety of application specific interconnection topologies. Based on our previous experience with pre-Web dataflow systems such as AVS, we expect WebFlow to be useful for a broad range of distributed computing problems in computational science and engineering which require interactive experimentation, fine-tuning of complex parameter spaces, selecting optimal algorithms in the trial-and-error mode etc. In fact, we are now testing this ansatz in a set of domain specific module and compute-web development activities. We started from a set of internal demos in a few domains such as image processing or synchronous collaboration. More recently, we also started to expose the system to selected external users in advanced computing labs such as NCSA, CEWES or ARL. We work with NCSA researchers via NPAC participation in the NCSA Alliance, and with the DoD labs such as CEWES or ARL as part of NPAC involvement in the DoD HPC Modernization Program.

We present here a brief overview of the computational domains being addressed by WebFlow and the associated module libraries developed so far, including both the internal demos and the external applications. As illustrated below, the analysis of these experiments allows us to test various computational topologies, user interface requirements, and the lessons learned in this first round of trial applications are helping us in developing the new, refined specification of the system, outlined in Chapter 8.

5.1 Image Processing

One of the earliest technology demonstrations of the WebFlow run-time system involved the design and development of customizable ImageProcessing Filters albeit expressed through the modules and the base sub-system. These ImageProcessing modules, facilitate creation of custom filtering operations akin to AVS or Khoros employing the universal interface engine of a browser, the flexibility of client-server communications and the rapid-response needed in similar stand-alone applications. Using these WebFlow Image Processing modules, Image objects can be created from raw data, the raw data of an existing Image Object can then be examined, and filters created to have modified versions. Image Objects created by the WebFlow runtime system can used similar to Image objects in the Java runtime system, they can be drawn to a display surface, or the result of a filtering operation can be used as input for to other ImageProcessing modules. Among the various ImageProcessing modules present in the system are:

1. HueModule: Which can modify the wavelength of light reflected from or transmitted through an object.
2. SaturationModule: Capable of editing the amount of gray in proportion to the Hue in colors.
3. EdgeDetectors: Employing the classic 9x9 Laplacian Transformed matrix and also the Canny Edge Detection routine.

Other modules perform filtering operations such as Luminance, Chromacity, Warping etc.

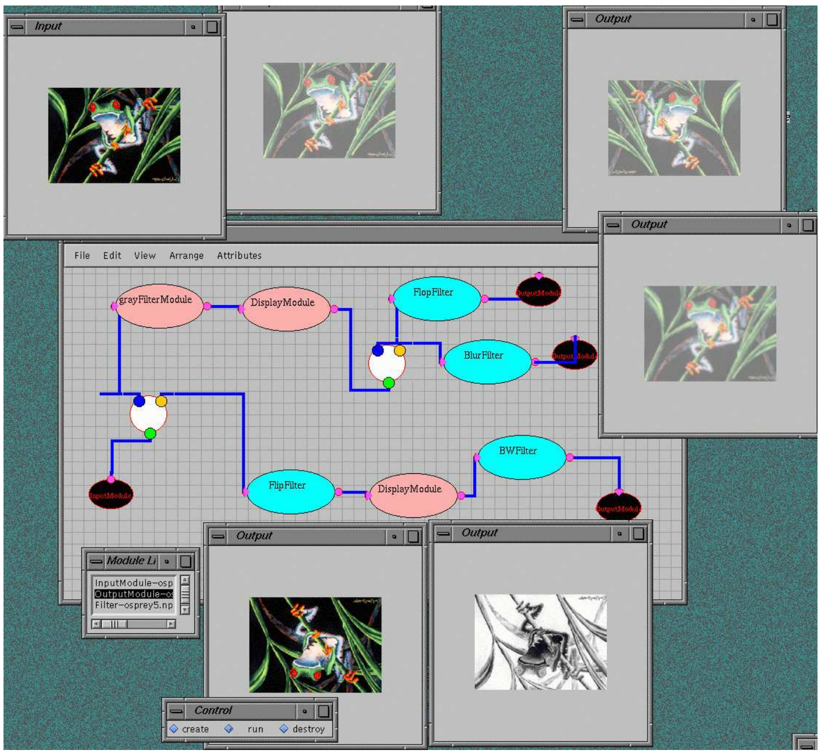

Fig. 3. A sample image processing demo of WebFlow. The input image is forked into several concurrent streams, passed to imaging filters (such as flip, flop, bleach etc.) running on various server nodes. Image input and output modules are associated with display applets which allow to visualize the results of all imaging operations.

5.2 Collaboration

Another application domain which was ported to the WebFlow Subsystem was Collaboration. The collaboration environment used was JSDA (Java Shared Data Architecture) from JavaSoft. JSDA provides a Shared Framework for Java at the data level. Data objects are shared over specific instances of *Channels* between two or more *Clients*. Channels are abstractions for data communication paths in JSDA, whereas Clients are objects which are the source or destination of data in a collaboration environment. Any Client object which needs to register its interest in receiving messages sent over a channel must implement the Channel Consumer Interface. On similar lines if a client is interested in being notified about changes in state of some other object it should implement the Channel Observer Interface. To register interest in a certain Channel, a Client first needs to join the Session which the Channel is a part of and then the Channel. A Client could be part of multiple Sessions and thus register interest in Channels across those various Sessions. JSDA also has objects which encapsulate management policies for other given objects. A classic example to this point is the Session Manager authenticating clients to determine if they could join a session.

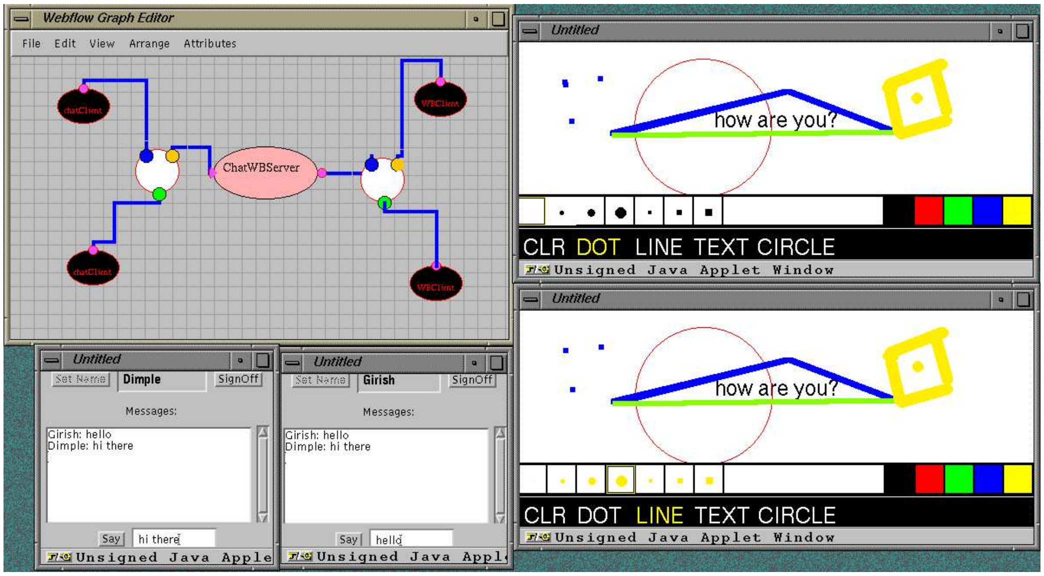

Fig. 4. A sample collaboratory demo of WebFlow, including JSDA based chat + whiteboard server module and four collaboratory client modules, associated with two users. WebFlow is used to setup the connection topology of a particular collaboration session and to initiate the JSDA socket or MI connections between collaboratory clients and servers.

The Collaboration module involved porting of the JSDA Session Server & Application-Client by writing WebFlow Wrappers around it. The applications ported to the WebFlow-Collaboration Framework included Chat, Whiteboard and Imaging. This framework provided for visual control of the base abstractions provided in the original JSDA environment via Session Servers, Channels and Clients. This meant that the Sessions and Clients could be coordinated in an intuitively simple click-and-drag fashion. All in all this seemed to be a step in the right direction for Collaborative environments, with JSDA providing the mechanism for fine-grained collaboration and WebFlow providing the scaffoldings for a coarse-grained distributed infrastructure.

5.3 Scientific Visualization

As a next step after the initial demos described above, we applied WebFlow as a wrapper technology for a HPCC computation (Monte Carlo simulation of the Potts spin system), developed within the PCRC (Parallel Compiler Runtime Consortium) project at NPAC. A sample WebFlow application in this area is demonstrated in Fig. 5. HPCC simulation program (Potts spin system) running in tier-3/backend is wrapped as a WebFlow module in tier-2/middleware 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 in tier-1/frontend.

More recently, we coupled WebFlow with a more powerful scientific visualization package SciVis [12], developed at NPAC and packaged as a 100% pure Java client-server system. Visualization server listens to the incoming data, applies filters selected by user and displays filtered data in an appropriate window. SciVis API can be used from C/ C++, Fortran and Java.

We developed 2D and 3D SciVis modules which can be used from WebFlow. After these modules receive the SciVis server address from the user, 2D and 3D SciVis visualization services can be used from the WebFlow environment by clicking, dragging and connecting the corresponding WebFlow modules/wrappers. These modules provide also the necessary conversion to the SciVis data stream formats.

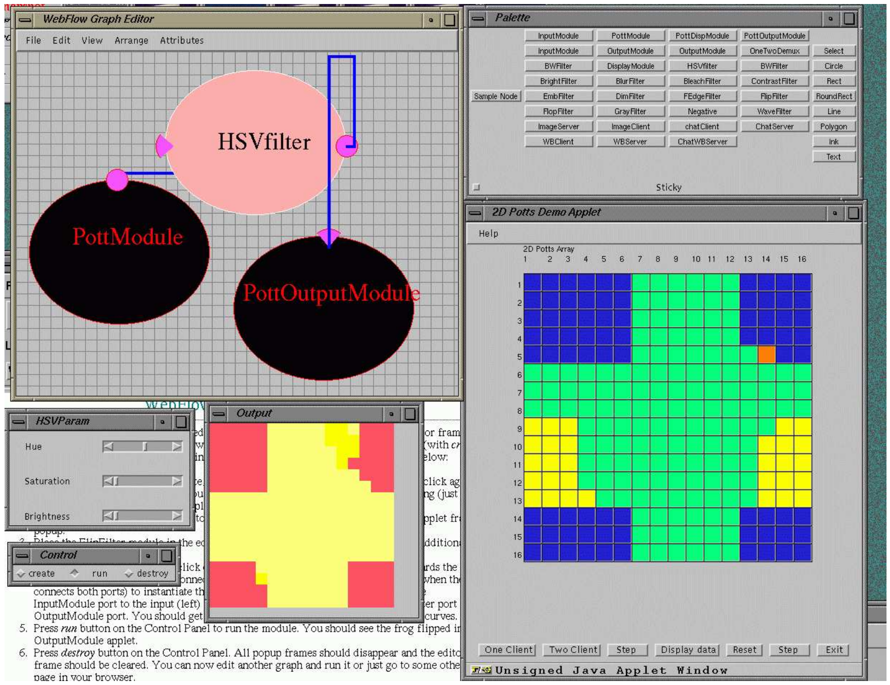

Fig. 5. WebFlow wrapper for HPCC simulation (Potts splin model). Visualization stream is sent both directly to the Potts applet and to the WebFlow HSV filter which allows for real-time tuning of the display paramaters.

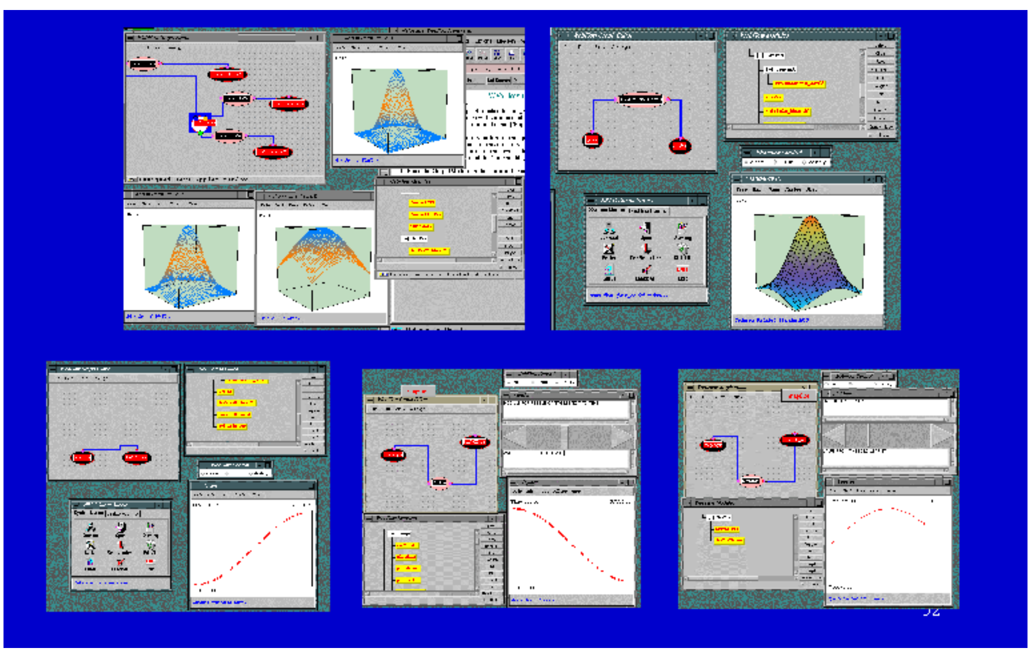

Fig. 6. Selected screendumps from WebFlow-SciVis demos. Data streams generated by source modules are passed via filters that support temporal or spatial modulation of the real-time visualization data.

Although the data from an application can be sent directly to SciVis, in many cases it might be useful to preprocess them before sending them through the network. As an example, we developed WebFlow modules that subsample the stream of data to be visualized. As a data source we used a simple code written in Java, or as it is described in section 5.4, an HPF code run inside DARP environment. We decreased the size of the data by decreasing the resolution (a spatial filter), by reducing the frequency of sampling (a temporal filter), or by selecting a two-dimensional slice from a three-dimensional data set. In each case the user has a full interactive control on the reduction rate (Fig.6).

The data producer, implemented as a Webflow module sends the raw data through its output port to the filter module. The filter does the specified data reduction according to parameters defined by the user via a Java applet which is associated with the module. After reduction the data are sent to a module that serves as a SciVis server proxy. The proxy implements the SciVis API, and makes the socket connection with the SciVis server on behalf of the user. Note, that once the SciVis proxy module is implemented, the user can visualize her data without any knowledge of the SciVis API. With a set of WebFlow modules that interface several different visualization packages, the user can alternate between the packages without any changes in his application by exchanging modules using the visual editor of WebFlow.

5.4 Interactive Prototyping and Debugging

In the next demonstration, we used an HPF code as a data producer. We used the DARP feature that allows to set action points to the HPF code without modifying the source code. Each action point set in the HPF code is an output port of the corresponding WebFlow module. Since the action points can be dynamically set in the DARP environment, we introduced a script that generates on the fly a WebFlow module representing the HPF code with the number of output ports corresponding to the number of the action points set.

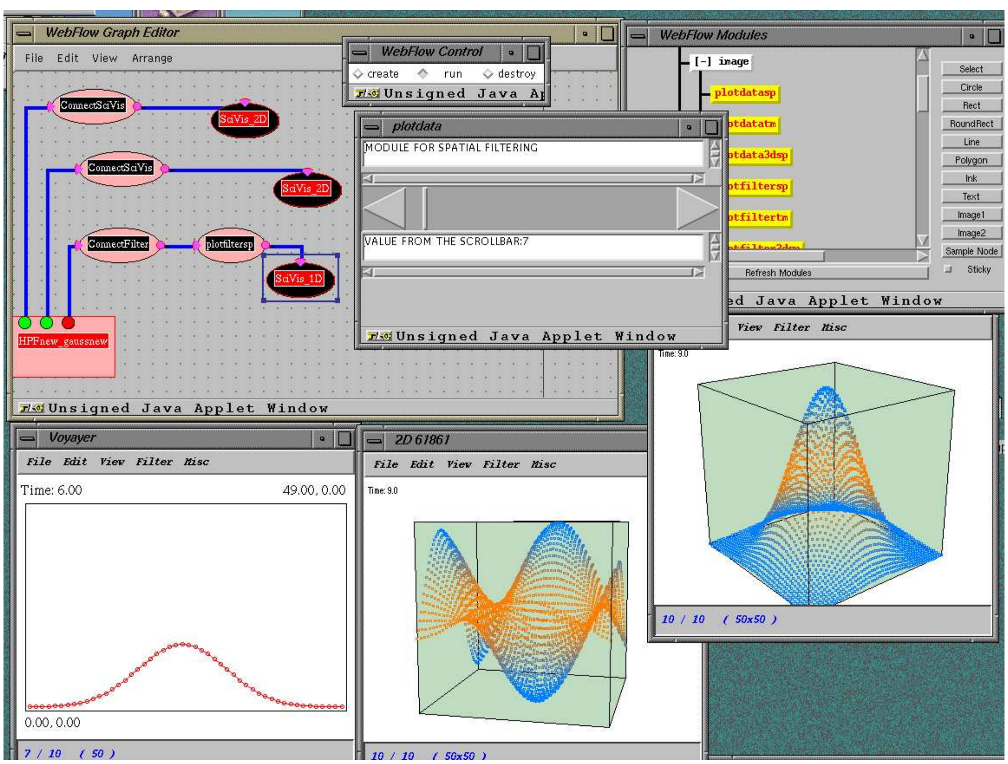

Fig. 7. Interactive prototyping/debugging session of a HPF application using WebFlow over DARP and SciVis module wrappers, described in the text and demonstrated at Supercomputing'97.

As a result we obtained a system with both DARP and SciVis systems encapsulated as Webflow modules. Having instrumentation of the Fortran code, and conversion to the WebFlow module done automatically, as well as ready to use SciVis proxy modules, we could perform an analysis of the data generated by the application code without any modification of the original Fortran code. Adding other features of the DARP system, such as setting breakpoints and modifying values of the distributed arrays using the HPF interpreter, we created a very powerful HPF development tool.

This integration of the DARP and Webflow is done for the demonstration purpose only. The "production-quality" implementation would remove some software redundancy introduced here. Both DARP and Webflow are three-tier systems, with an overlapping functionality of the middleware. The 2nd-tier DARP server can be implemented as a part of the Webflow middleware, with the DARP back end serving as a Webflow module, and DARP front-end as a control Applet of the that module.

5.5 Quantum Monte Carlo Simulations

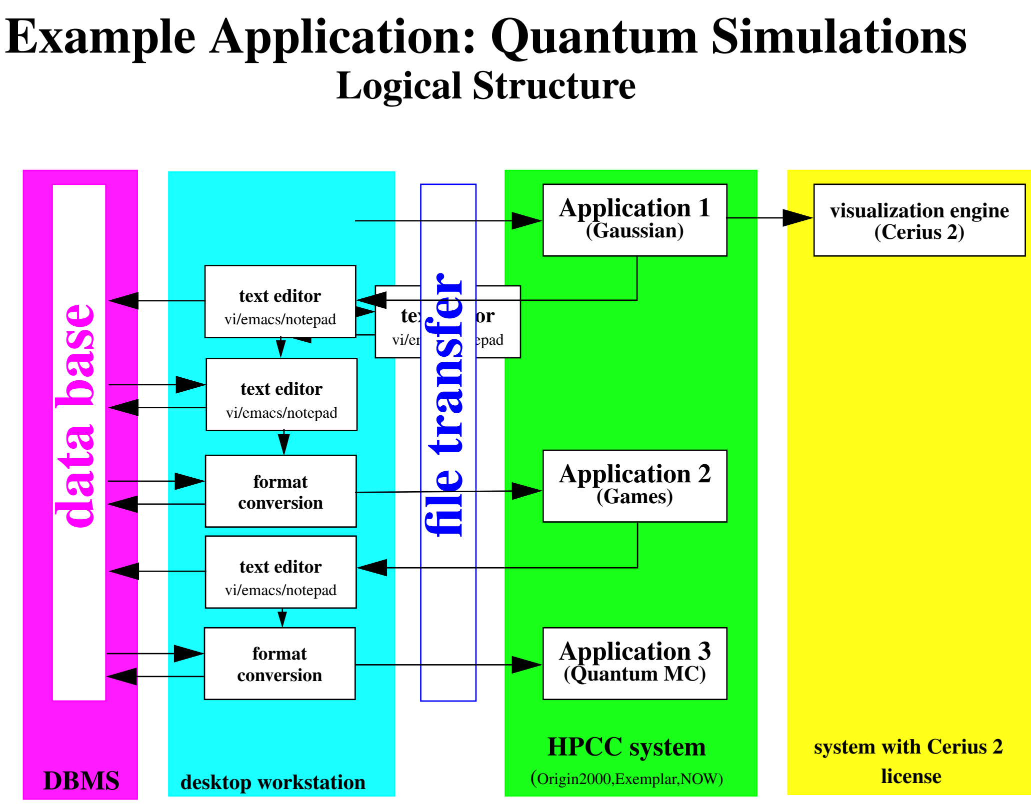

Fig. 8. Typical dataflow patterns for Quantum Monte Carlo simulations discussed in the text.

Encouraged by the results of the early experiments with the WebFlow concept, we applied this system to a real life application, Quantum Monte Carlo Simulation for Condensed Matter Physics at NCSA [15]. For the purpose of our research, we can describe this application as a chain of computational modules executed on different, typically high performance machines (c.f. Fig.8). Each module takes an ASCII input data, which is derived from the output of the previous module in the chain. This output-to-input transformation requires the professional expertise, and often the researcher's intuition as well, and therefore it is difficult to automate. As the result, after each step in the simulation chain the data flow is interrupted, and the researcher is given an opportunity to visually inspect the output file. It is left to the researcher discretion to continue with the next module, or to repeat the current step with a modified input file.

The WebFlow support for such computation requires therefore a suite of modules that submit the executables on the target machines, open an editor window on the user's workstation, and transfer data files between systems. A typical data-flow application would use the port mechanism (a pipe) to move the data between modules. In this particular case we decided to use FTP instead. There are three reasons to do so. First, the saved copies of the intermediate results serve as checkpoints. Second, the data flow is interrupted anyway, since at each step the data are opened in the editor. Third, the data are often sent to the machines where the WebFlow servers are not (yet) installed - a typical case when using a high performance platform. Consequently, we developed a WebFlow module that serves as a FTP proxy.

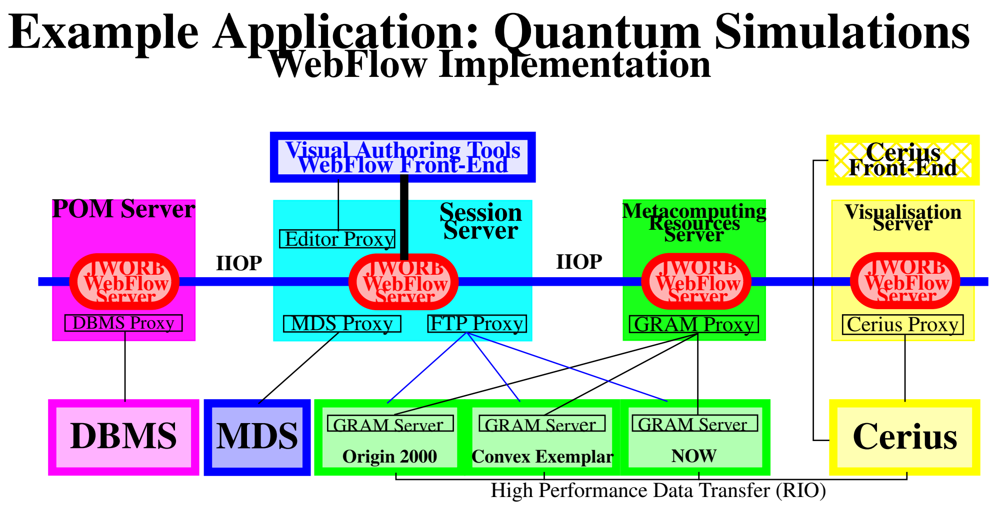

Fig. 9. WebFlow interface to Globus services under development as part of the NCSA National Grid activities, to be tested for the Quantum Monte Carlo application discussed in the text.

This application, pursed within the NCSA National Grid program, give us also an opportunity to experiment with the WebFlow system as an authoring tool for developing metacomputing applications. Following our general strategy of reusing commodity software components, from commercial off-shell products to achievements of the academic research, we use the Webflow system as a visual front end to the Globus[16], a low level metacomputing toolkit developed at ANL and USC. In particular, we use Metacomputing Directory Services (MDS) of Globus to assist the user with identifying available computational resources, and we use Globus Resource Allocation Manager (GRAM) for authentication and remote job submission.

Another aspect of this application is the need for an organized method of saving data in order to be able to reproduce a given simulation in the future, or analyze data coming form a set of simulations. At present the data are distributed between different systems, in directories of students participating in the research. We used the Webflow middleware for a seamless integration of the simulation software with an off-shell DBMS.

5.6 Military Modeling and Simulation

NPAC participates in the PET (Programming Environments and Training) part of the DoD HPC Modernization Program in the technology area of FMS (Forces Modeling and Simulation). This field includes a broad range of military wargaming systems, based on real-time models (typical for combat level simulations), logical-time models (typical for forces level simulations) or faster-than-real-time models (typical for testing, evaluation and analysis). All these diverse simulation techniques and frameworks, so far supported by dedicated kernels and specialized development tools are now being integrated towards the common HLA [17] (High Level Architecture) framework. HLA is a distributed object/component based model and it specifies:

- its object system, including SOM (Simulation Object Model) specification for components or federates, and FOM (Federation Object Model) specification for aggregates or federations;
- RTI (Run-Time Infrastructure) which acts as a common software bus that interconnects and provides a set of communication services for all federates participating in a given federation.

By imposing a set of DoD-wide interoperability standards, HLA enforces reusability of military simulations. However, new generation authoring tools are required to support composition of simulations from its components, developed locally or retrieved from remote FOM/SOM repositories and customized for a federation at hand.

At NPAC, we are developing WebFlow based tools to facilitate HLA component transfer between heterogeneous distributed object databases and to specify connectivity between federates using visual graph editing techniques. HLA enforces reusability and composability in the tool space within the FEDEP (Federation Development Process) specification that offers a set of DIFs (Data Interchage Formats) to be used as mandatory input and/or output formats for the individual tools. Hence, FEDEP approach maps naturally on the WebFlow architecture with HLA tools to be wrapped as WebFlow modules, connected bia DIF-conforming communication channels.

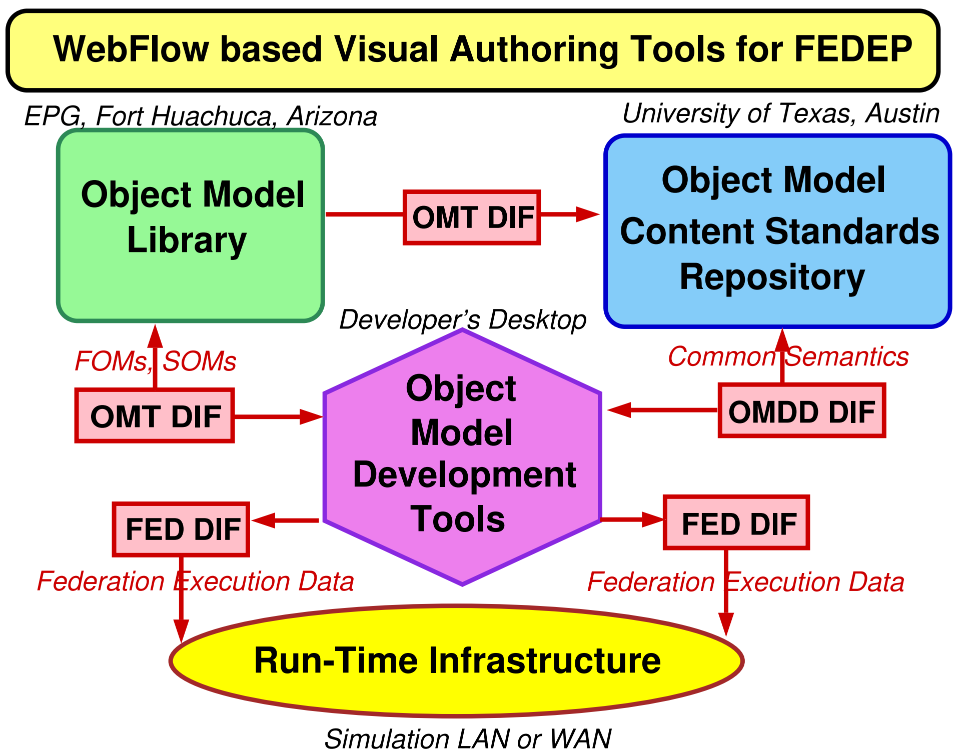

Fig. 10. Typical dataflow patterns between HLA FEDEP compliant tools. Geographically distributed services such as HLA object databases, FOM/SOM authoring tools and simulation runtime are interconnected via DMSO DIF conforming communication chanels and hence the HLA FEDEP naturally maps on the WebFlow model.

In another FMS project, we are also developing Java/CORBA based RTI that will use our new middleware layer JWORB (Java Web Object Request Broker) discussed in Section 8.1. By combining WebFlow tools for HLA object authoring with JWORB based RTI, we will be able to construct the full WebHLA system, including both the simulation development and runtime components. Early demonstrations in this area are planned for summer/fall '98.

5.7 Other Planned Applications

We are currently exploring the use of the Webflow middleware in some other areas of advanced distributed computing, including the Landscape Modeling System (LMS) by DoD and the ASCI (Advanced Strategic Computing Initiative) systems in the DoE Defense Labs.

In the LMS case, we intend to treat a multidisciplinary meta-application, consisting of a number of distributed, tightly coupled applications, each representing a different aspect of environmental modeling, as a single object. The challenge here is to identify and allocate computational resources, identify and retrieve input data sets from many different sources (GIS, weather satellites, etc.), and launch the simulations, typically on a geographically distributed system.

In the ASCI [18] case, we plan to develop a general purpose 3-tier system with JWORB based middleware, UML based visual authoring and Globus based metacomputing backend. Some components of this system are outlined in Chapter 8 below. Such ASCI-WebFlow would offer a user-friendly and standards based high performance computing environment to support seamless visual development of advanced metacomputing applications.

6.0 WebFlow Source Organization, Distribution and Availability

We give have an overview of the WebFlow release organization, inclduing a brief description of some important packages. The top level directory consists of directories like *doc*, *gjt*, *jsda*, *uci*, *webflow*, etc. The *jsda* directory contains the Java Shared Data Architecture package from Javasoft, recently renamed as the Java Shared Data Toolkit, used for the collaboratory demos. The *gjt*, *uci* and *symantec* packages are part of the vector graphics package GEF, developed at University of California, Irvine. GEF is the backbone for building WebFlow's user interface. The *doc* directory contains the HTML documentation for the webflow system and the various modules used in the demo. The *webflow* directory contains all the core packages used by the Webflow system.

The main packages used by the WebFlow system are divided into *frontend*, *backend*, *modules* and *utils* sub-directories. *Frontend* consists of classes for building WebFlow's user interface. It uses the GEF package for building the Webflow editor applet and the various module representations. The *WFtree* package inside the *frontend* directory consist of classes used for building the scrollable module tree palette used as the navigation bar for selecting and placing modules into the WebFlow editor. The *backend* directory consists of the three manager servlets and it's associated classes. This directory also contains the two classes that the module developers will use, namely the *Port* class and the *Module* interface, and some custom messaging classes used by the WebFlow system. *WFSession* sub-directory hold classes for session management. The *util* directory contains classes for the worker pool threads and event management used by the WebFlow system.

The WebFlow prototype comes with a set of demo modules packaged into the *modules* directory. Default input and output modules are present in the *io* sub-directory and the utility modules for displaying images and forking the data into two modules etc. are present in the *util* sub-directory. The *image* sub-directory contains various imaging modules used for the imaging demos in the WebFlow prototype. Various filter modules are present in this sub-directory like the Flip, Flop, Negative, Rotate, Wave filters etc. The *collab* directory contains the collaboratory modules developed on top of the JSDA package. The demo modules present in this directory include a chat module, white board module and collaboratory imaging toolkit module. The *HPF* sub-directory contains modules for integrating WebFlow with the HPF server and the *SciVis* sub-directory contains the modules for integration with the SciVis server. The *doc* directory present at the WebFlow top level directory contains tutorials for connecting and running all these modules. The current release of webflow prototype can be downloaded from WebFlow homepage at *http://osprey7.npac.syr.edu:1998/iwt98/products/webflow.*WebFlow distribution comes in gziped and tar format for Unix and 32 bit Windows platforms, respectively. WebFlow demos should run on all web servers with servlet support.

7.0 Lessons Learned in Current WebFlow Prototype

The overall concept and design of WebFlow was found successful in the first round of trial applications and the prototype software turned out to be reasonably stable. Of course, we share this credit with Jason Robbins at UCI who developed the GEF package and the Java Web Server team at JavaSoft, but we effectively demonstrated that Web/Java based visual dataflow systems can be prototyped with a modest amount of effort and that the production version of such a system is worth considering.

Recent, more demanding applications such as SC'97 demos or Quantum Monte Carlo support for NCSA exposed various deficiencies of the current prototype, both in the GUI and module API sectors. Support for module applets was found too simplistic and the need arised for better management of multi-applet multi-frame displays. Other required and currently missing features include: creating composite modules via visual grouping of component icons, saving/fetching compute-webs, support for more flexible and robust module repositories, automatic resource allocation, automatic module parameter control, module input data caching and defaulting etc.

Another flaw of the current WebFlow is lack of any convenient programmatic or other third party interface to WebFlow module libraries. WebFlow modules can be composed to form distributed applications only within the WebFlow environment and only using the human-operated visual tools. One can therefore expect developers to be reluctant to embrace the WebFlow model unless the system accumulates some critical mass of useful module libraries.

8.0 Building new WebFlow Framework for Pragmatic Object Web

Based on lessons learned in the current WebFlow prototype summarized in Chapter 7, we are now building new WebFlow framework that conforms to the emergent standards for distributed objects and components such as CORBA, DCOM, RMI or WOM [19][20]. A standards based WebFlow middleware will assure protection of investment in WebFlow module development since such modules will be reusable in other commodity containers. Furthermore, a standards-based environment will facilitate development of more advanced features listed above which are required by the WebFlow users and missing in the current prototype.

However, standards in this area are still in the early formation stage. For example, recent OMG/DARPA workshop on compositional software architectures [20] illustrated very well both the growing momentum and the multitude of options and the uncertainty of the overall direction in the field. A closer inspection of the distributed object/component standard candidates indicates that, while each of the approaches claims to offer the complete solution, each of them in fact excels only in specific selected aspects of the required master framework. Indeed, it seems that WOM is the easiest, DCOM the fastest, pure Java the most elegant and CORBA the most realistic complete solution.

In our Pragmatic Object Web [10] approach we adopt the integrative methodology i.e. we setup a multiple-standards based framework in which the best assets of various approaches accumulate and cooperate rather than competing. We start the design from the middleware which offers a core or a 'bus' of modern 3-tier systems and we adopt Java as the most efficient implementation language for the complex control required by the multi-server middleware. We adopt CORBA as the base distributed object model at the Intranet level, and the (evolving) Web as the world-wide distributed (object) model. System scalability requires fuzzy, transparent boundaries between Intranet and Internet domains which therefore translates into the request of integrating the CORBA and Web technologies. We implement it by building a Java server (JWORB) which handles multiple network protocols and includes support both for HTTP and IIOP. On top of such Pragmatic Object Web software bus, we implement specific computational and collaboratory services.

In this Chapter, we summarize the main layers and features of the new WebFlow framework under development, including JWORB middleware and several commodity alternatives for the front-end and back-end technologies.

8.1 JWORB (Java Web Object Request Broker) based middleware

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 Server and it handles the IIOP connections as an Object Request Broker. As an HTTP server, JWORB supports base Web page services, Servlet (Java Servlet API) and CGI 1.1 mechanisms. In its CORBA capacity, JWORB is currently offering the base remote method invocation services via CDR based IIOP and we are now implementing the Interface Repository, Portable Object Adapter and selected Common Object Services.

After the core JWORB server starts up, it looks at configuration file to find out which protocols are supported and it loads the necessary protocol classes for each protocol (*Definition*, *Tester*, *Mediator*, *Configuration*). *Definition* Interface provides the necessary *Tester*, *Configuration* and *Mediator* objects. *Tester* object looks at the current connection's stream and decides whether it can interpret this connection or not. *Configuration* object is responsible for the configuration parameters of a particular protocol. *Mediator* object serves the connection. New protocols can be added simply by implementing the four classes described above and by registering a new protocol with the JWORB server.

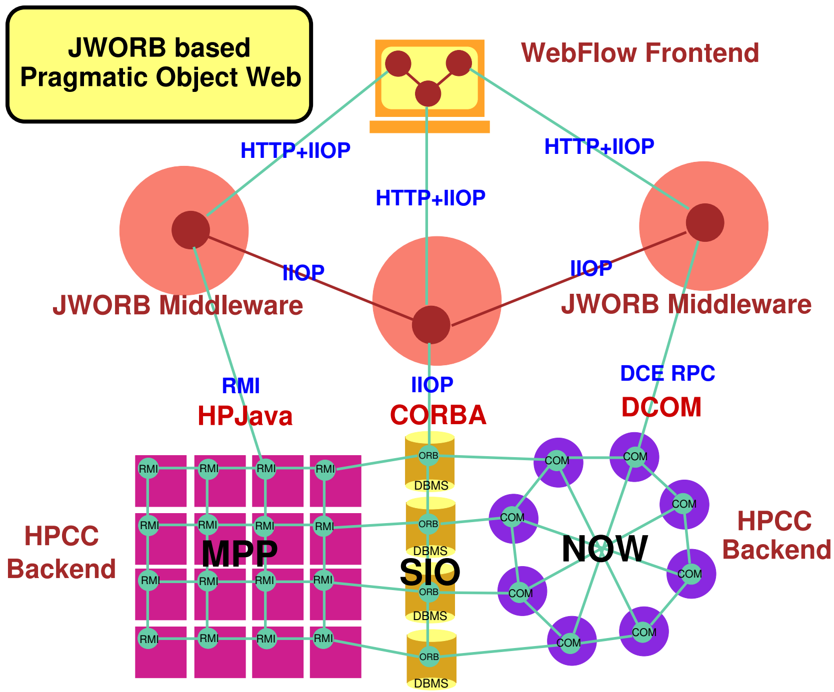

Fig. 11. JWORB based protocol integration. Frontend browsers are connected with JWORB middleware via HTTP+IIOP combination. JWORB nodes communicate using IIOP and connect to the backend modules via IIOP or RMI or via suitable bridges to the DCOM world.

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. Current algorithm looks at each protocol according to their order in the configuration file. This process can be optimized with randomized or prediction based algorithm. At present, only HTTP and IIOP messaging is supported and the current protocol is simply detected based on the magic anchor string value (*GIOP* for IIOP and *POST*, *GET*, *HEAD* etc. for HTTP).

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 the 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.

8.2 WOMA (Web Object Management Architecture) based Integration

As discussed above, our multi-standard integration framework starts from the Java based CORBA middleware and then gradually incorporates other emergent standard candidates, using the OMA services and facilities as the implementation base. The advantage of this approach is that new dynamic features of HTTP-NG can be quickly and reliably prototyped using Java in the solid software engineering framework of CORBA. The specific API for new WebFlow modules will likely emerge at the cross-road of the CORBA component, DCOM component and the Enterprise JavaBeans models. Armed with a Java based CORBA augmented by the CORBA/COM bridge, we will be able to freely experiment with all these component standard candidates.

We are also analysing several other emergent standard candidates coming from the Web, Enterprise, Desktop of Military domains and we are exploring their integration patterns within the new WebFlow framework. We discuss some of these activities below and in the next few Sections.

In the visual graph editing sector, UML appeared recently as a new promising standard candidate, adopted by OMG and also embraced by Microsoft. We therefore intend to base new WebFlow authoring model on the UML model, extended towards runtime support for UML activity and collaboration diagrams.

The WebFlow operating environment, represented in the current prototype by custom Session-, Module- and Connection Managers, shares in fact several common features with the standard RTI layer we are currently developing within the FMS projects. It seems therefore reasonable to build the new WebFlow runtime around the JWORB based RTI communication dynamics, thereby assuring close interoperability with the new generation on-line gaming systems.

W3C introduces its own distributed object/componentware framework, sometimes referred as WOM, and emergent as a combination of XML, RDF and DOM standards. We are currently evaluating XML as a possible candidate for the universal data format for inter-modular transmission channels. We also intend to incorporate DOM as a CORBA service and to integrate RDF with our efforts towards transparent persistence across UNIX and PC platforms discussed below.

We coined the term WOMA (Web Object Management Architecture) which can be viewed as a merger of W3C WOM [20] and OMG OMA and we use to refer to our multi-standards based integration approach. Basically, rather than debating if we should follow OMA, Java, COM, WOM etc. we are setting instead an evaluation testbed and an integration framework that will let these technologies to work in concert. Such process will clearly expose both the synergies and the essential differences between various approaches and hopefully it will contribute itself to enforcing and enabling common standards for the Web, Desktop, Enterprise and Defense computing. In the following sections, we summarize some of our current activites, aimed at integrating the alternative standard candidates for the front-end, middleware and the backend layers.

8.3 Integrating DirectX/Java3D/VRML to support commodity desktop frontends

We are looking into Java3D, DirectX and VRML for the runtime display support in the WebFlow front-end. Java3D is a graphics runtime that uses a hierarchical set of relationships between object in three dimensional scene to render 3D graphics. Java3D API is also provided for writing 3D graphics applications or web-based 3D applets. DirectX is a group of technologies designed by Microsoft to make Windows-based computers a winning platform for running and displaying applications rich in multimedia elements such as full-color graphics, video, 3D animation, and surround sound. Built directly into the Windows family of operating systems. DirectX API, is designed specifically for high-performance applications like games. DirectX is a thin layer providing direct access to hardware services. The technology takes advantage of available hardware accelerators and emulates accelerator services when accelerators are not present. VRML on the other hand, is the International Standard (ISO/IEC 14772) file format for describing interactive 3D multimedia on the Internet. VRML is also one of the file formats supported by Java3D. Hence, there are natural tradeoffs between DirectX performance and Java3D/VRML platform independence which we intend to address in terms of suitable bridges between the underying component layers.

8.4 Integrating DCOM via COM/CORBA bridges

CORBA-COM Bridge is a communication mechanism between two distinct object management systems: Microsoft's COM and OMG's CORBA. The purpose of the COM - CORBA bridge is to specify support for two-way communication between CORBA and COM objects. The goal is that object from one object model should be able viewed as if they existed in the other model. There are many similarities between the two systems. In particular both are centered around the idea that an object is a discrete unit of functionality that presents its behavior through a set of fully-described interfaces. Each system hides the details of implementation from its clients. Much of the COM/CORBA Interworking specification from OMG simply involves mapping the syntax, structure and facilities of each to the other.

There are however, differences in the CORBA and COM object models. COM and CORBA, each have a different way of describing what an object is, how it is typically used, and how the components of the object model are organized. These differences raise a number of issues as how to provide the most transparent mapping. The optimal solution would allow objects from either system to make their key functionality visible to clients using the other system as transparent as possible. We are currently exploring the COM/CORBA bridge issues both in the Java/J++ (JWORB) and C++ (omniORB2) domains.

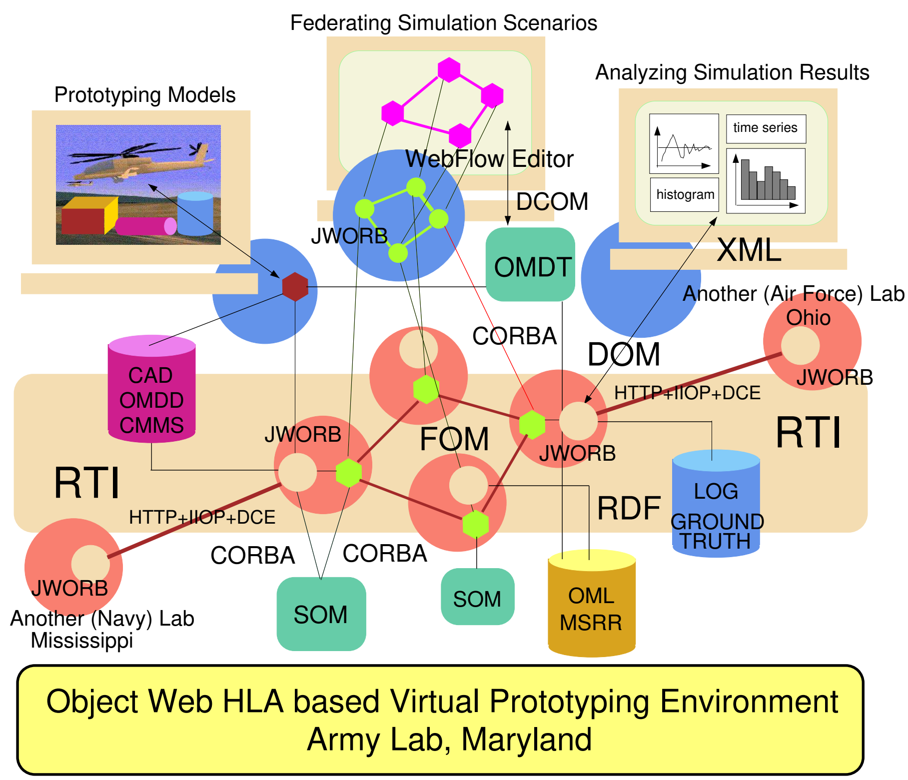

Fig. 12. An example of advanced WebFlow/JWORB application under development: WebHLA based Virtual Prototyping Environment for Simulation Based Design/Acquisiton. A spectrum of authoring tools for federate object engineering, federation scenario development, runtime control and analysis is packaged as WebFlow modules and integrated within the JWORB/WOMA multi-standard model.

8.5 Integrating OLEDB/JDBC/PSS to support transparently persistent backends

We are currently exploring different database integration technologies like JDBC API from Javasoft, OMG's Persistence State Service and OLEDB/ADO by Microsoft for database support for the WebFlow system. JDBC is a standard SQL database access interface for accessing heterogeneous databases from Java programs. It encapsulates the various DBMS vendor proprietary protocols and database operations and enables applications to use a single high level API for homogenous data access.Persistent State Service (PSS) addresses the issues of making persistent CORBA objects across machines, platforms and datastores. PSS provides the platform for storing and managing distributed business objects over heterogeneous datastores in a reliable and scalable manner for general and common shared use. OLEDB, which is the core of Microsoft's Universal Data Access strategy, defines a set of COM interfaces by which data providers, consumers and service components can interact with ease, for developing multitier enterprise applications. Applications or service components like query processor, cursor engine etc. can access the underlying diverse data in a unique way. The ActiveX Data Objects built on top of OLEDB gives a language and data provider -neutral, extensible and easy to use way for manipulating the data.Thus applications can use the same interface to access various heterogeneous datastores like mail stores, project management tools, ODBC databases etc. We are currently building a general purpose support for persistent WebFlow modules and we are addressing the associated JDBC/PSS/OLEDB integration issues.

9.0 Summary and Outlook

We presented here the overall design, the current status and the next steps of the WebFlow system under development at NPAC. The first applications of our new WebFlow, based on the JWORB middleware and WOMA integration model will appear later in '98 in the area of HLA/RTI based advanced military modeling and simulation systems. Fig. 12 illustrates a JWORB/WOMA based Virtual Prototyping Environment to support growing DoD needs for Simulation Based Acquisition systems which we use as a guide and we view as the eventual goal for our WebFlow based WebHLA system under development.

Our exploration of WebFlow application domains only just started and we expect to find the system useful in several other areas of modern computing. We believe that our JWORB+WOMA integration approach will offer an efficient testbed for evaluating and possibly integrating new emergent Web/Commodity technologies. Given the ongoing competition between the major standards bodies, such a testbed operated within an academic lab linked with several federal programs such as NPAC might play a role itslef in the formation process of Web/Commodity standards for distributed object/componentware computing.

10.0 References

1. G. C. Fox, W. Furmanski, M. Chen, C. Rebbi and J. Cowie, "WebWork: Integrated Programming Environment Tools for National and Grand Challenges", March 1995, http://www.npac.syr.edu/techreports/html/0700/abs-0715.html.
2. G. Fox and W. Furmanski, "HPcc as High Performance Commodity Computing", book chapter in "Building National Grid", edited by I. Foster and C. Kesselman, Morgan and Kaufman 1998, http://www.npac.syr.edu/users/gcf/HPcc/HPcc.html.
3. Dennis Gannon, "Component Architectures for High Performance Distributed Meta-Computing", OMG/DARPA Workshop, January 1998, http://www.objs.com/workshops/ws9801/papers/paper086.html.
4. D. Bhatia, V. Burzevski, M. Camuseva, G. Fox, W. Furmanski and G. Premchandran, "WebFlow - a visual programming paradigm for Web/Java based coarse grain distributed computing", June '97, in the special issue of "Concurrency: Practice and Experience" on Java for Scientific Computing, http://www.npac.syr.edu/projects/webspace/doc/cpande/feb97/feb97.ps.
5. WebFlow Project Home Page, http://osprey7.npac.syr.edu:1998/iwt98/products/webflow.
6. D. Dias, G. C. Fox, W. Furmanski, V. Mehra, B. Natarajan, H. T. Ozdemir, S. Pallickara, Z. Ozdemir, "Exploring JSDA, CORBA and HLA based MuTech's for Scalable Televirtual (TVR) Environments", presented at the Workshop on OO and VRML, VRML98 Conference, Monterey, CA, Feb 16-19,1998, http://tapetus.npac.syr.edu/iwt98/pm/documents/vrml98/paper.html.
7. G. C. Fox, W. Furmanski and H. T. Ozdemir, "JWORB - Java Web Object Request Broker for Commodity Software based Visual Dataflow Metacomputing Programming Environment", submitted for the HPDC-7, Chicago, IL, July 28-31, 1998, http://tapetus.npac.syr.edu/iwt98/pm/documents/hpdc98/paper.html.

8. G.C. Fox, W. Furmanski, B. Natarajan, H. T. Ozdemir, Z. Odcikin Ozdemir, S. Pallickara and T. Pulikal, " Integrating Web, Desktop, Enterprise and Military Simulation Technologies To Enable World-Wide Scalable Televirtual (TVR) Environments", submitted to the Workshop on Web-based Infrastructures for Collaborative Enterprises, the WET ICE'98 Conference, Stanford University, June 17-19,1998, http://osprey7.npac.syr.edu:1998/iwt98/projects/webhla/users/hasan/papers/WETICE/paperWETICE.html.

9. D. Bernholdt, G. C. Fox, W. Furmanski, B. Natarajan, H. T. Ozdemir, Z. Odcikin Ozdemir and T. Pulikal, "WebHLA - An Interactive Programming and Training Environment for High Performance Modeling and Simulation", submitted to the DoD HPC 98 Users Group Conference, Rice University, Houston, TX, June 1-5 1998, http://tapetus.npac.syr.edu/iwt98/pm/conferences/DoDHPC98UGC/Abstract.htm.

10. G. C. Fox, W. Furmanski and S. Pallickara, "Building Distributed Systems for the Pragmatic Object Web", book in progress, http://www.npac.syr.edu/users/shrideep/book.

11. Jason Robbins, "GEF: Graph Editing Framework", University of California, Irvine, 1997, http://www.ics.uci.edu/pub/arch/gef/.

12. B. Ki and S. Klasky, "Collaborative Scientific Data Visualization", in proceedings of ACM 1998 Workshop on Java for High-Performance Network Computing, http://kopernik.npac.syr.edu:8888/scivis/.

13. E. Akarsu, G. Fox, T. Haupt, "DARP: Java based Data Analysis and Rapid Prototyping Environment for Distributed High Performance Computations", in proceedings of ACM 1998 Workshop on Java for High-Performance Network Computing, http://www.npac.syr.edu/users/haupt/HPFI/.

14. Binary Black Hole Grand Challenge Project Home Page, http://www.npac.syr.edu/projects/bh.

15. NCSA Condensed Matter Physics Home Page, http://www.ncsa.uiuc.edu/Apps/CMP/cmp-homepage.html.

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

17. High Level Architecture (HLA) by the Defence Modelling and Simulation Office (DMSO), http://www.dmso.mil/hla/.

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. Robert Orfali and Dan Harkey, "Client/Server Programming with Java and CORBA" , 2nd Edition, Wiley 1998, http://www.wiley.com/compbooks/catalog/24578-X.htm

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