WebFlow: a Framework for Web Based Metacomputing

Tomasz Haupt,Erol Akarsu,Geoffrey Fox
{haupt,akarsu,gcf}@npac.syr.edu

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

Abstract. We developed a platform independent, three-tier system, called WebFlow. The visual authoring tools implemented in the front end integrated with the middle tier network of servers based on CORBA and following distributed object paradigm, facilitate seamless integration of commodity software components. We add high performance to commodity systems using GLOBUS metacomputing toolkit as the backend.

1 Introduction

Programming tools that are simultaneously sustainable, highly functional, robust and easy to use have been hard to come by in the HPCC arena. This is partially due to the difficulty in developing sophisticated customized systems for what is a relatively small part of the worldwide computing enterprise. Thus we have developed a new strategy - termed HPcc High Performance Commodity Computing [1] - which builds HPCC programming tools on top of the remarkable new software infrastructure being built for the commercial web and distributed object areas.

This leverage of a huge industry investment naturally delivers tools with the desired properties with the one (albeit critical) exception that high performance is not guaranteed. Our approach automatically gives the user access to the full range of commercial capabilities (e.g. databases and compute servers), pervasive access from all platforms and natural incremental enhancement as the industry software juggernaut continues to deliver software systems of rapidly increasing power.

We add high performance to commodity systems using a multi tier architecture with traditional HPCC technologies such as MPI and HPF supported as the backend of a middle tier of commodity web and object servers.

Our research addresses needs for high level programming environments and tools to support distance computing on heterogeneous distributed commodity platforms and high-speed networks, spanning across labs and facilities. More specifically, we are developing WebFlow - a scalable, high level, commodity standards based HPDC system that integrates:

- High-level front-ends for visual programming, steering, and data visualization built on top of the Web and OO commodity standards (Tier 1).

– Distributed object-based, scalable, and reusable Web server and Object broker Middleware (Tier 2)
– High Performance back-end implemented using the metacomputing toolkit of GLOBUS [2] (Tier 3)

We have demonstrated the fully functional prototype of WebFlow during Alliance'98 meeting as applied to a metacomputing application Quantum Simulations [3], and at SuperComputing '98 were we demonstrated the Landscape Management System [4] using WebFlow over a geographically distributed system.

In this paper we critically summarize our experience using WebFlow, and for the first time we present a redesigned, CORBA-based middle-tier. The paper is organized as follows. In section 2.0 we provide the WebFlow overview from the user point of view. In section 3.0 we discuss the original implementation of the middle-tier based on our experience applying the WebFlow to real applications. Section 4.0 is devote to our new design which is meant to remedy shortcomings indicated in Section 3.0. The paper is summarized in Section 5.0

2 Overview of WebFlow

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

WebFlow is a particular programming paradigm implemented over a virtual Web accessible metacomputer. We support many different programming models for the distributed computations: from coarse-grain dataflow to object-oriented to fine-grain data-parallel model. In the dataflow regime, a WebFlow application is given by a computational graph visually edited by end-users, using Java applets. The dataflow WebFlow modules, atomic computational units of an metacomputing application, exchange data via input and output ports, in a way similar to that used in AVS [5]. This exchange of data can realized in the following way. Whenever a module is ready to send the data (encapsulated as an object), it fires an event triggering all modules interested in receiving these data to call the corresponding method to retrieve the data. This model can be generalized to allow the module to fire arbitrary events, and add arbitrary event listeners. As the result, the module can invoke an arbitrary method of the other modules involved in the computation. The reason why we distinguish between the dataflow and object-oriented model has an historical origin. Our first WebFlow implementation [6, 7] supported the dataflow model exclusively. Also, we use a different strategies to implement the front-end of the system for the dataflow and the general object-oriented models.

Nothing prohibits the user to encapsulate a data parallel application as a single WebFlow module. In this case the user is solely responsible for the interprocessor communications (we used HPF and MPI-based codes to run WebFlow modules on a multiprocessor systems[8]). More, using the DARP system[9], implemented as a WebFlow module, we were able to interactively control an HPF application at runtime, and dynamically extract the distributed data and send them to a visualization engine. This approach can be used for computational steering, runtime data analysis, debugging, and interprocessor communications on demand. Finally, we integrated two independently written applications that write checkpoint data [4]. We used WebFlow to detect the existence of the new data, and transfer them to the other application.

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

An important part of our design is devoted to providing a secure and seamless access to remote resources, in particular to HPCC systems. We build the WebFlow security infrastructure on top of the SSL protocol [11]. In particular, we use GSS-API [12] based Globus GRAM to allocate HPCC resources. In this sense, the WebFlow can be regarded as a visual, high-level user interface to Globus, or conversely, we may state that we use Globus as the high performance backend of the WebFlow.

3 WebFlow Applications

3.1 Quantum Simulations (QS)

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

This application can be characterized as follows. A chain of high performance applications (both commercial packages such as GAUSSIAN or GAMESS or custom developed) is run repeatedly for different data sets. Each application can be run on several different (multiprocessor) platforms, and consequently, input and output files must be moved between machines. Output files are visually inspected by the researcher; if necessary applications are rerun with modified input parameters. The output file of one application in the chain is the input of the next one, after a suitable format conversion.

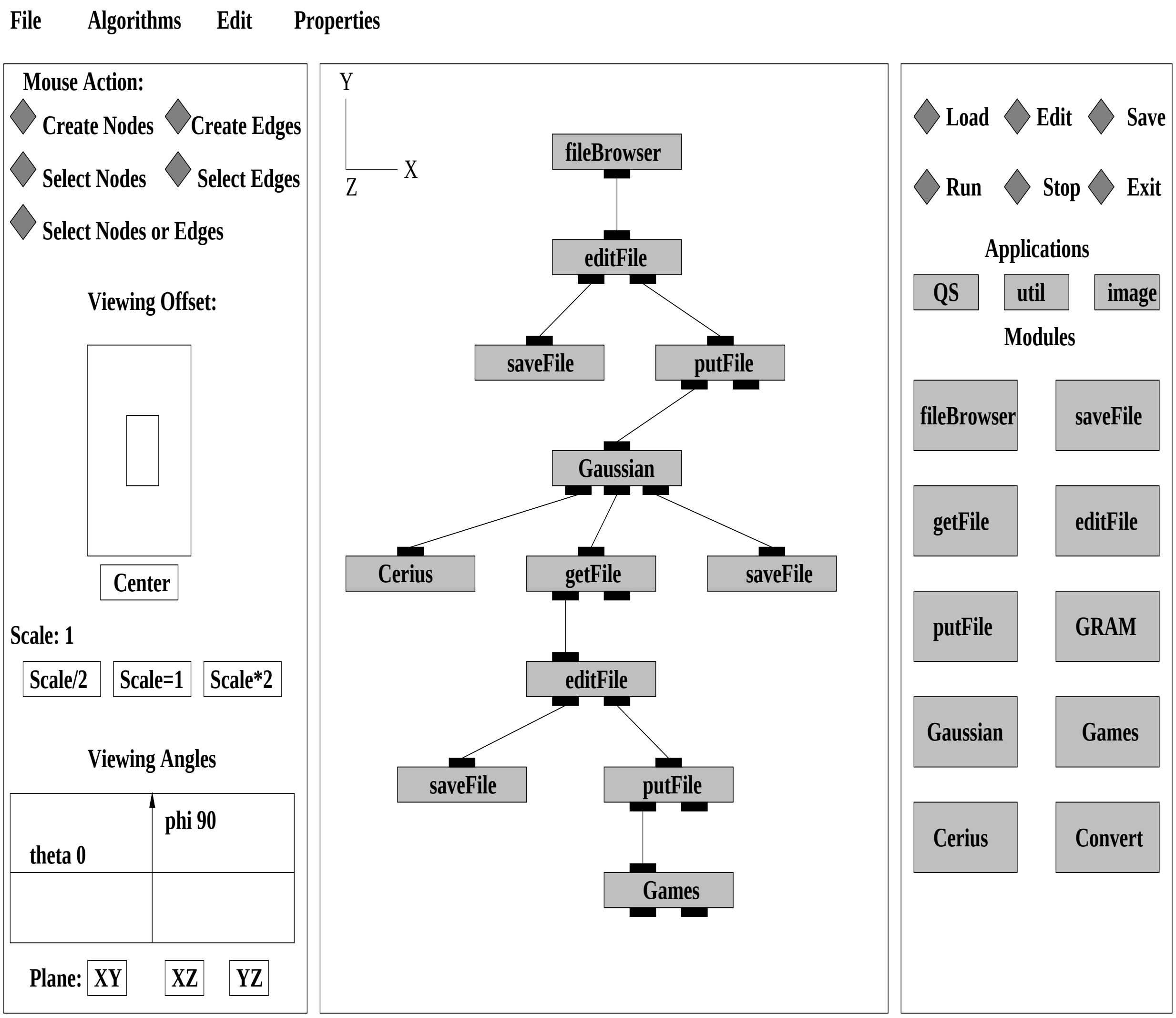

Fig. 1. Composing a dataflow application using the WebFlow visual editor (a Java applet)

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

For QS we are using the WebFlow editor as the front-end (c.f. fig 1). The WebFlow editor provides an intuitive environment to visually compose (click-drag-and-drop) a chain of data-flow computations from preexisting modules. In the edit mode, modules can be added to or removed from the existing network, as well as connections between the modules can be updated. Once created network can be saved (on the server side) to be restored at a later time. The workload can be distributed among several WebFlow nodes (WebFlow servers) with the interprocessor communications taken care of by the middle-tier services. More, thanks to the interface to the Globus system in the backend, execution of particular modules can be delegated to powerful HPCC systems. In the run mode, the visual representation of the metaaplication is passed to the middle-tier by sending a series of requests (module instantiation, intermodule communications) to the Session Manager.

3.2 Land Management System (LMS)

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

- Data retrieval from many different remote sources (web sites, databases)
- Data preprocessing to prune and convert the raw data to a format expected by the simulation software.
- Execution of two simulation programs: EDYS for vegetation simulation including the disturbances and CASC2D for watershed simulations during rainfalls. The latter results in generating maps of the soil condition after the rainfall. The initial conditions for CASC2D are set by EDYS just before the rainfall event, and the output of CASC2D after the event is used to update parameters of EDYS.
- Visualizations of the results.

The purpose of this project was to demonstrate feasibility of implementing a system that would allow launching and controlling the complete simulation from a networked laptop. We successfully implemented it using WebFlow with WMS and EDYS encapsulated as WebFlow modules running locally on the laptop and CASC2D executed by WebFlow on remote hosts

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

3.3 Shortcomings of the original WebFlow implementation

In the original prototype of WebFlow [6], the middle-tier is given by a mesh of Java enhanced Web Servers, running servlets that manage and coordinate distributed computation. This management is implemented in terms of the three servlets: Session Manager, Module Manager, and Connection Manager. These servlets are URL addressable and can offer dynamic information about their services and current state. Each of them can also communicate with each other through sockets. Servlets are persistent and application independent.

In spite of the success of the WebFlow project we see that our current implementation suffers form some limitations. Two the most obvious areas of improvement we want to achieve are fault tolerance and security. However, instead of adding complexity to already complex and to large extend custom protocol of exchanging data between the servlets based on low-level sockets, we re-implemented the WebFlow middle-tier using industry standards distributed object technologies: JavaBeans [13] and CORBA [14].

4 CORBA Based Implementation of WebFlow

The architecture of the new implementation of the WebFlow is shown in fig.2:

Architectureofthe WebFlow system Visual authoring tools, PSEs, or custom developed application front-ends allow the user to specify or compose her application, and they implement WebFlow interface to the middle-tier. This way a particular front-end solution comes as a "plug-in" to the system, and can be easily replaced by another one. It is our experience that different applications require different front-ends. The WebFlow API specifies how the front-end interacts with the middle-tier. It includes establishing and termination of a WebFlow session, which generally requires authentication of the user and authorization to use the resources. We base the authentication process on X.509 certificates and authorization services we model after AKENTI system [15]. We are in the early stage of implementing this functionality, and a more detailed description will be provided in a forthcoming paper [16]. Further, the WebFlow API defines how the user creates applications from predefined modules in a form of an abstract job description that specify all resources (hardware and software) needed to build and execute the meta-application. The abstract job description may request a particular resource or delegate the selection of the resources that match the application requirements to the system. The WebFlow middle tier, in turn, delegates this task to the services provided by a metacomputing toolkit such as Globus' MDS or CONDOR's matchmaker. Finally, the WebFlow API provides means to control of the modules life cycle.

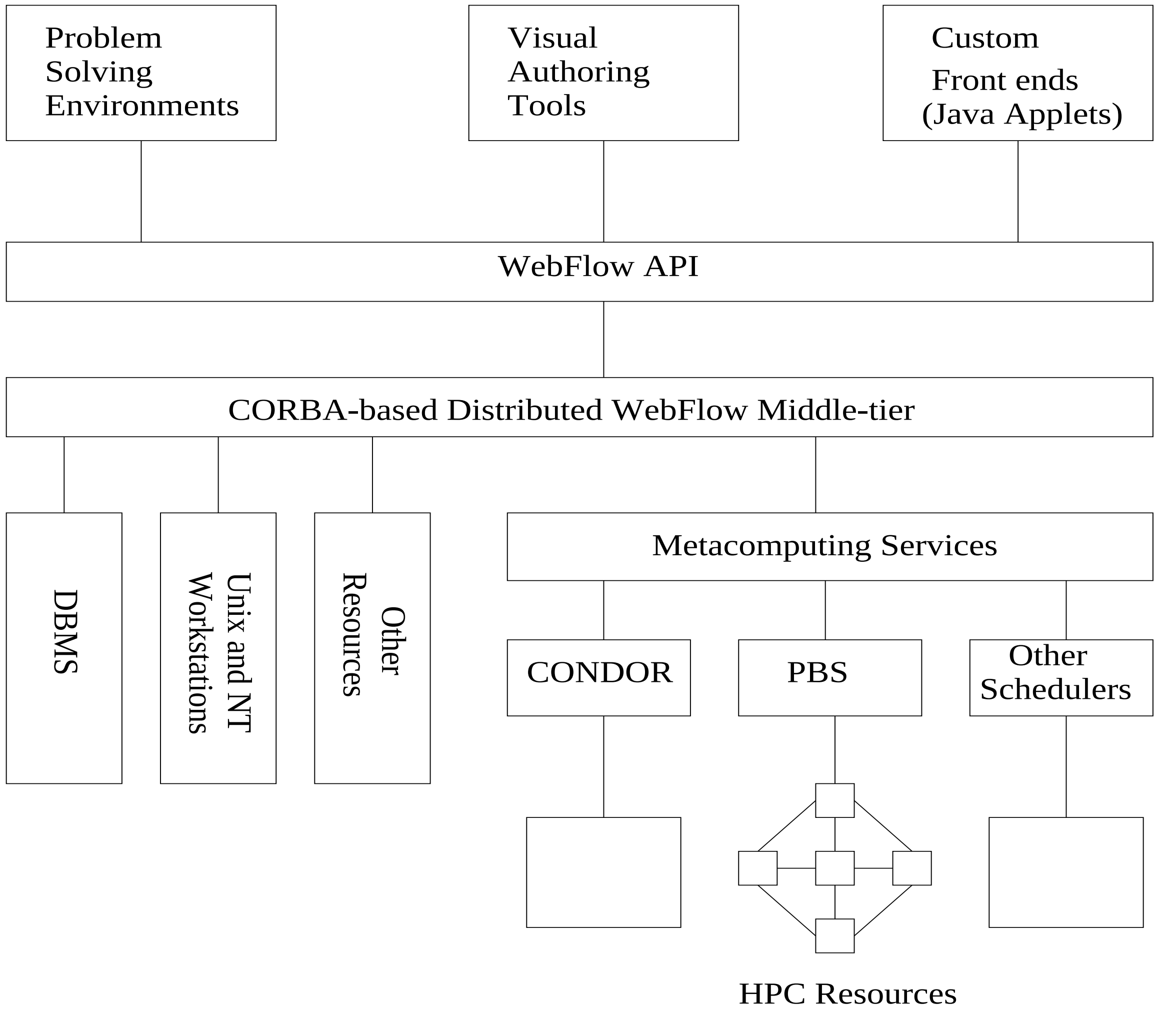

Figure 2. Architecture of the WebFlow System

The middle-tier is given by a mesh of WebFlow servers implemented as pairs of Web server and ORB. Web Servers are needed for communications with the front-end. In particular, the front-end applets are downloaded from these servers. We tested two secure (i.e., supporting SSL) Web servers: Apache [17] and Jigsaw[18]. All communications between peer WebFlow servers and the back-end modules are facilitated using CORBA. We use ORBacus [19], a free ORB implementation for this purpose, and we are testing JWORB [10], a multiprotocol server in Java that supports both http and IIOP.

Once the user is authorized to use the WebFlow resources, a session context (a container object) is created for her. Within her session context, the user creates application contexts, so that the user can run several independent applications within one WebFlow session. Within the application context, the user builds her application from preexisting modules, and registers the events and corresponding event listeners with an event adapter (which is run as a CORBA service). The event adapter maintains a translation table, and in conjunction with the CORBA name service and interface repository it allows for binding of events with methods of the target modules independently of the modules location. For modules than run on remote WebFlow servers proxy modules are automatically produced. The proxy modules make it easier to control the life cycle of the modules in the application context. In addition, they are necessary to maintain communications with the front-end controls, implemented as Java applets, of the remote module (to avoid the Java sandbox restrictions).

The WebFlow modules are CORBA objects. Typically, we implement them as the Java beans. The very fact that we are using CORBA and not Java RMI makes it possible to include objects written in languages different than Java, in particular C++.

For high performance computations the WebFlow modules serves as proxies to services provided by the metacomputing toolkits. We have demonstrated this in our implementation of the Quantum Simulations. The codes that actually run on Origin 2000 were either made available to us only in a binary form (GAUSSIAN and GAMES are commercial products), or were written in Fortran. Using them within WebFlow did not require any modifications, letting alone rewriting them in Java. We just created WebFlow proxy modules that on behalf of the user generated requests to the Globus GASS to stage the data and executables on the target machine, and retrieve the results, and requests to the Globus GRAM to allocate the resource (including authentication and authorization). Note that all GRAM requests to be specified in a low-level RSL (resource specification language) were generated by the WebFlow modules in the fly by interpreting the abstract job description which in turn was created using a visual WebFlow front-end running as an applet within a Netscape browser.

5 Summary

To summarize, we developed a platform independent, three-tier system: the visual authoring tools implemented in the front end integrated with the middle tier network of servers based on CORBA and following distributed object paradigm, facilitates seamless integration of commodity software components. In particular, we use the WebFlow as a high level, visual user interface for GLOBUS. This not only makes construction of a meta-application much easier task for an end user, but also allows combining this state of art HPCC environment with commercial software, including packages available only on Intel-based personal computers.

References

1. G. Fox and W. Furmanski, "HPcc as High Performance Comodity Computing", chapter for the "Building National Grid" book by I. Foster and C. Kesselman, http://www.npac.syr.edu/users/gcf/HPcc/HPcc.html
2. I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit", International Journal of Super computing Applications, 1997. See also Globus Home Page: http://www.globus.org
3. Quantum Simulations , http://www.ncsa.uiuc.edu/Apps/CMP/cmp-homepage.html
4. T. Haupt, "WebFlow High-Level Programming Environment and Visual Authoring Toolkit for HPDC (desktop access to remote resources)", Supercomputing '98 technical presentation, see http://www.npac.syr.edu/users/haupt/WebFlow/papers/SC98/foils/index.htm
5. Advanced Visualization System, http://www.avs.com/
6. D. Bhatia, V. Burzevski, M. Camuseva, G. C. Fox, W. Furmanski and G. Premchandran, "WebFlow - a visual programming paradigm for Web/Java based coarse grain distributed computing", Concurrency: Practice and Experience, Vol. 9 (6), pp. 555-577, June 1997.
7. T. Haupt, E. Akarsu, G. Fox, W. Furmanski, "Web based metacomputing ", Special Issue on MetaComputing for the FGCS Int. Journal on Future Generation Computing Systems, see also http://www.npac.syr.edu/users/haupt/WebFlow/papers/FGCS/index.html
8. G. Fox, W. Furmanski and T. Haupt, SC97 handout: High Performance Commodity Computing (HPcc), http://www.npac.syr.edu/users/haupt/SC97/HPccdemos.html
9. E. Akarsu, G. Fox, T. Haupt, DARP: Data Analysis and Rapid Prototyping Environment for Distribute High Performance Computations, Home Page http://www.npac.syr.edu/projects/hpfi/
10. JWORB Project Home Page http://osprey7.npac.syr.edu:1998/iwt98/projects/worb, see also G. C. Fox, W. Furmanski and H. T. Ozdemir, "JWORB (Java Web Object Request Broker) based Scalable Middleware for Commodity Based HPDC", submitted to HPDC98.
11. SSL, Netscape Communications, Inc, http://home.netscape.com/eng/ssl3/index.html
12. RFC 1508, RFC 2078
13. Sun Microsystems, Inc., http://java.sun.com
14. CORBA - OMG Home Page http://www.omg.org
15. AKENTI home page: http://www-itg.lbl.gov/security/Akenti/homepage.html
16. Check the WebFlow home page at http://www.npac.syr.edu/users/haupt/WebFlow/demo.html

17. Home page http://www.apache.org
18. Jigsaw home page: http://www.w3.org/Jigsaw/
19. Object Oriented Concepts, Inc., http://www.ooc.com/ob.html