WebVM (Virtual Machine) based WebFlow Paradigm for Visual Programming in Adaptive HPCC Applications

Geoffrey C. Fox, Wojtek Furmanski, Tom Haupt, Scott Klasky

Northeast Parallel Architectures Center, Syracuse University, Syracuse, NY
{gcf,furm,haupt, scott}@npac.syr.edu, http://www.npac.syr.edu

Marina Chen

Department of Computer Science, Boston University, Boston, MA
mcchen@cs.bu.edu http://cs-www.bu.edu

James H. Cowie

Cooperating Systems Corporation, Chestnut Hill, MA
cowie@cooperate.com http://www.cooperate.com

Jim Browne, Manish Parashar

Department of Computer Science, University of Texas, Austin, TX.
{browne, parashar}@cs.utexas.edu

Matthew Choptuik

Department of Physics, University of Texas, Austin, TX.
{choptuik}@infeld.ph.utexas.edu

Proposal to the National Science Foundation
New Technologies Program

Draft9.4 • May 21, 1996

PROJECT SUMMARY

HPCC presence on the Web is currently limited to the archival of text (and to a limited extent, sourcecode and data) by institutions, for retrieval by individual researchers. This picture will change radically within the next few years, as Web servers, clients, and standards achieve universal penetration of the desktop environment. The Web's mesh of hyperlinked text, video, and executable software services will become the most widely accepted distributed computing software platform for the next decade, and the use of Web technologies to achieve HPCC goals will grow accordingly.

With the addition of a cooperative multi-server executable content, the Web will form an attractive basis for the geographically distributed data intensive applications and collaborations envisaged in the resolicitation of the NSF supercomputer centers. Critical research challenges to be addressed include integration with traditional parallel and distributed computing approaches, scalability concerns for messaging and I/O within a network of cooperating Web servers, interactive large-dataset visualization, little languages for distributed problem solving environments, and the role of object-based based message passing (MIME evolving to Java and CORBA).

Our workplan is positioned to take advantage of the best technologies that emerge as the de facto standards. We initiated this process in 1995 with early WebTop pilots such as WebTools for personal CGI-extended web servers and WebWork for linking such computational servers to form distributed processing surfaces. Our first WebWork application, RSA130 Factoring-by-Web was presented at the Supercomputing 95 and was given the award as the most geographically dispersed and heterogeneous solution in the Teraflop Challenge contest. Lessons learned in the Factoring-by-Web experiment, new technologies coming from the Web industry, and the growing interest in Web based HPCC within the Grand Challenge community, are now converging in the form of our proposed systems: WebVM (Web based Virtual Machine), MetaWeb (system/cluster management) and WebFlow (adaptive distributed dataflow). These models exhibit aspects of coarse grain software integration which are insensitive to the modest bandwidth and high latency of geographically distributed computing and current HTTP Web Servers.

WebVM is given by a mesh of interacting computationally extended Web servers, and forms the base infrastructure for a variety of high level programming environments. WebVM can gradually build reliable world-wide scalability by using portable transparent module API design starting from the Intranet domain and current Web technologies. MetaWeb facilitates this process by adding system/cluster management and performance visualization support. WebFlow imposes a dataflow programming paradigm on top of WebVM and offers tools for visual authoring of computational graphs as well as "little language" based scripted support for adaptive programmatic variations of the dataflow topology.

WebFlow inherits concepts from previous coarse grain dataflow based computations, popularized by systems such as AVS, Khoros, HENCE or CODE. The Web client-server model provides natural support for dataflow-based access to distributed information. Furthermore, the Web's established framework for electronic publication can be extended to support Web publication of software modules within a standardized plug-and-play interface. WebFlow integrates computing, parallel I/O and information (database, VRML visualization) services in this paradigm.

This proposal focuses on base WebVM and WebFlow support; MetaWeb extensions are addressed in a separate proposal. In this proposal we combine this research and technology development with an application testbed focused on supporting adaptive mesh refinement (AMR) for the numerical solution of PDEs. In three tightly coupled components of the proposed project, we propose to develop a WebVM/WebFlow infrastructure, an AMR benchmarking suite, and a PDE testbed/toolkit environment.

We will continually make the software components of the developing Intranet testbed available for experimentation by the public to get feedbak and facilitate formation of other WebHPCC Intranet sites world-wide – a WebFlow/HPCC developers' network. By augmenting our developing WebVM/WebFlow framework with the solid MetaWeb system management, and by linking the sites of the WebFlow HPCC developers' network, we can provide foundation for a true Web based Metacomputer that can span the globe. This will allow HPCC researchers to fully leverage the Web's primary strength: universal access to common tools and standards for computing, authoring and information.

TABLE OF CONTENTS

PROJECT DESCRIPTION

1 Introduction, Motivation and Overview

We propose a new paradigm for high performance computing and communications which builds on top of computationally extended Web technologies. Initial concepts in the WebWork project in early 1995 were demonstrated in the RSA130 Factoring-by-Web prototype world-wide application, which was awarded with the Teraflop Challenge at Supercomputing 95. Factoring-by-Web addressed a teraflop size, coarse grain, embarrassingly parallel computational problem (Random Field Sieve factoring algorithm) by mapping it onto a dynamic tree of computationally extended Web servers, linked by low bandwidth, high latency communication channels that formed a World-Wide Computer.

We have also prototyped other aspects of the integration of HPCC with the Web. These prototypes include: the use of relational and object Web-linked databases to store Java and VRML to display information produced by large scale applications; a prototype of a PVM linked Web servers to provide a Web interface to HPF; Java collaboratory technology; and a prototype of a Java visual editor for a distributed computing environment.

New Web technologies developments such as the Java programming language offer a promising technology framework for systematic refinement and improvement of our WebWork design, and the gradual adaptation of Web based computing to solve Grand Challenge and Metacomputing problems of increasing complexity. In fact, our Web based HPCC concepts recently attracted the interest of the Binary Black Hole Grand Challenge community, weather prediction modeling systems[CAPS], and Mary Wheeler to implement robust AMR based PDE solvers.

We propose to build two base components of the Web based HPCC: A WebVM (Web based Virtual Machine) infrastructure layer and a WebFlow programming paradigm for visual and adaptive dataflow computation, to be prototyped in an AMR testbed environment, applied to Grand Challenge computations.

WebVM is a mesh of interacting, computationally extended Web servers. WebFlow imposes dataflow based topology (*compute-web*) on top of WebVM and offers Java applet based visual authoring tools for such computational graphs. Some of these concepts are inherited from previous dataflow systems such as AVS, Khoros, HenCE or CODE, and extended by the Web's own natural publication, navigation and information processing capabilities. AMR applications require dynamically varying compute-web topologies which goes beyond static visual programming and is implemented by a mesh of simple "little language" interpreters that extend our original WebWork design.

The proposed three stage implementation will be based on the best available Web technologies which will be compatible with new developments via stable transparent API for the end users, module developers and system administrators. Stage 1 (months 1–7) is based on conventional HTTP+CGI WebVM along with a WebTools based navigational front-end. During stage 2 (months 8–12) we will introduce a Java applet based visual compute-web authoring system which will be coupled to the Stage 1 WebVM backend. Finally, Stage 3 addresses the high performance implementation of the WebVM layer in terms of distributed Java based performance optimized compute-servers (months 13–16) and advanced interactive visualization based on VRML 2.0, Java and DSI–like simulation framework (months 17–20).

The project will deliver (months 21–24) an operational Web based AMR toolkit for PDE's and a more general purpose WebVM and WebFlow programming paradigm, ready to address other Grand Challenge and MetaComputing applications. Although a full problem solving environment is outside the scope of this proposal, we believe our systems probe critical basic research issues underlying sophisticated Web based domain specific environments that we expect to be developed in the future.

Our modular portable framework offers an interesting integration platform for computing, parallel I/O, information processing, databases, visualization and scientific collaboration. The current proposal focuses on a geographically localized Intranet implementation but the overall WebVM/WebFlow design will be ready for wide area aggregation of such regional Web HPCC activities. In a parallel project, MetaWeb, we address the cluster and system management support that will facilitate production version of our Intranet prototypes and scalability of the model towards wide-area and eventually world-wide metacomputing.

2 WebVM – Web Virtual Machine

2.1 Computational Web Concepts

The base support for the computational extensions of the Web is already present as part of the standard model in the form of the Common Gateway Interface (CGI) protocol. These CGI extensions are usually exploited in conventional client-server configurations, e.g. for dynamic document generation, handling forms or accessing database servers. Typically, a CGI process receives a set of arguments from the client, performs some computation at the server side, and returns a MIME object to the client. More elaborate services, which started to appear recently, request an URL of the input MIME object from the client domain, fetch this object to the server domain, perform a suitable transform or filter and return it to the client as on output MIME object. Popular examples include AutoCAD DXF to VRML file convertors, remote image editing and improvement services, or remote VRML authoring kits.

A natural next step could include combining such services, e.g. by dataflow chaining. This would bypass all the intermediate formats and manual manipulations on temporary files. At the moment, such linkage of Web services is impractical or impossible by the lack of a standard methodology for publishing and connecting individual computational modules. *WebVM* will provide a framework that would standardize this level of interoperability.

Once the standards for data exchange between computational modules are established, the WebVM will become a platform for software integration on an unprecedented scale. Particular services, implemented on the hardware of choice, become **opaque** nodes of an inherently distributed, heterogeneous WebVM. We have experience in using AVS for this purpose, to compose computational kernels encapsulated into AVS modules within a single, complex, multidisciplinary application run concurrently on several machines of different architectures [Gang94]. The dataflow paradigm employed here has two very attractive features. First, it enables collaborative creation of a community owned library of modules. The AVS users in fact created such a library available from the International AVS center, making it simple to develope AVS applications. The second feature of the dataflow computations is that the user defines only data dependencies between the modules and not a fixed schedule. When implemented on a distributed system it is possible to exploit a coarse grain parallelism. Concurrent execution of data independent modules has a nontrivial impact on the overall performance of the application by hiding the costs of intermodule communications on a typically high latency network.

To demonstrate the feasibility of a distributed dataflow computation using existing Web standards, we developed the concept of WebWork [FFCRC95] - a distributed computing environment based on a mesh of computationally extended Web servers. Since the HTTP protocol can be used to implement the point-to-point message passing between servers we see that dataflow offers a natural high-level programming paradigm in this framework. WebFlow is built on these concepts which can use the Java programming language for developing the visual authoring tools for editing WebFlow networks, or compute-webs. For a simple proof of this concept, a WebWork application - parallel search - was prototyped in terms of a mesh of Web servers [Bala95]. A more advanced WebWork application in the area of large–integer factorization within the RSA factoring challenge was presented at Supercomputing 95 and awarded within the Teraflop Grand Challenge Contest [Cowie95].

Factoring is a very computationally demanding application, and in a traditional approach it would be implemented on a high-end supercomputer delivering maximum available performance. Instead, the task has been completed on a heterogeneous collection of workstations utilizing otherwise idle CPUs. The key to the success was modularity of the factoring algorithm, structure of data dependencies that allowed for concurrent execution of modules, and conformance to http protocol for intermodule communication.

WebWork can be enhanced by adding capabilities to monitor system resources, for example to detect idle workstations. Here it would be possible to create a dynamical, self-adjusting network of workstations solving a problem, following an *adaptive* dataflow (*adaptive WebFlow,*) paradigm, a scripted extension of the base WebVM model. Hence, we are following these ideas to develop a Web based cluster manager systems, MetaWeb [Baker96].

The need for adaptivity comes not only from the varying load of the system, which is external to the application, but it may be driven internally by an application itself. An excellent example of an aggressively adaptive application is a PDE solver using Adaptive Mesh Refinement (AMR) techniques. AMR's basic principle, ‘‘distribute resolution as needed", can be directly translated to the adaptive WebFlow concept ‘‘assign resources as needed,” and the same

MetaWeb scripting mechanism can be used. As a result, both external and internal demands for adaptivity can be served simultaneously by the same MetaWeb scripting mechanism.

Modularity of the AMR algorithm promises an efficient implementation on an adaptive WebVM. The same operations (unigrid hyperbolic and/or multigrid elliptic PDE solvers) are performed on all grids in the AMR hierarchy. These operations are local to a grid, and they are computationally expensive enough to overshadow the overhead cost of intergrid communications (initialization of data on fine grids and restrictions). On the other hand, an adaptive WebVM seems to be a very well suited plaftorm for AMR computations since it can be more responsive to requests for large and dynamically varying memory segments than an MPP with fixed memory per node. Below, we discuss our design which builds on the previous WebWork concepts and takes into account the most recent developments in the Web software industry.

In WebWork, we described the HTTP+CGI mechanism for server-to-server communication and we indicated how to generalize and improve this approach in terms of the next generation of dynamic Web server technologies. These technologies are becoming available today from several vendors, most notably from Netscape and JavaSoft. In the following, we refine our minimal base design and we address the issue of interoperability with the coming commercial products.

2.2 HTTP+CGI – Minimal Model

Minimal WebVM is given by a collection of CGI extended Web servers, typically with an independent server running on each node of a workstation cluster. The CGI extension includes a library of system CGI modules which are provided as part of the WebVM distribution, and a set of custom, user/application provided CGI modules. The overall computational paradigm is given by dataflow with user CGI processes acting as computational nodes/modules, and with the system CGI processes providing the required system control and management functions.

Each module receives a set of input MIME objects, performs an arbitrary local computation, and returns a set of output MIME objects. Both input and output objects are URL-addressable, i.e. they reside as MIME documents in a server document tree, and can be transferred between servers during the dataflow actions.

Each module reads input files and writes output files to the local disk. Server-to-server object transfer is implemented via HTTP protocol and carried out by a system CGI module called *Flow Executive*. Fig, 1 illustrates the steps involved in transferring the output object $O_1^{(1)}$ created by module M_1 from the server S_1 document tree to the server S_2 document tree, where it is stored as the input object $I_1^{(2)}$, ready to be used by the module M_2. The transfer starts after the module M_1 has finished its computation and writes $O_1^{(1)}$ to the local disk. After completing the write action, M_1 opens up a connection to S_2 and sends a POST message to the Flow Executive F_2, passing the current URL of stored $O_1^{(1)}$. F_2 sends the GET message to S_1, retrieves $O_1^{(1)}$, and stores it as $I_1^{(2)}$. Next, F_2 starts module M_2 in the background and exits. At this point, M_1 can also exit and all connections between servers S_1 and S_2 are closed. Module M_2 reads its input files $I_1^{(2)}$, $I_2^{(2)}$, performs local computation, writes its output files $O_1^{(2)}$, $O_2^{(2)}$, and notifies the Flow Executive processes in the destination servers.

Fig.1 illustrates only two neighboring nodes M_1, M_2 and one arc ($O_1^{(1)} \rightarrow I_1^{(2)}$) of a larger computational graph called a *compute-web*. A given web server can host and handle modules from several compute-webs, belonging to different users. Local connectivity information is stored in configuration files in each server. The initial or idle state of a WebVM involves a set of (always) running Web servers which act as network daemons, and no other WebVM-specific active CGI processes. Dataflow is typically initiated by a browser which activates the initial module, (e.g.) M_1 in Fig.1, via the usual URL addressing mode and the CGI convention. The subsequent M processes are activated whenever the state of any of their input objects changes as a result of some previous dataflow action, and the F processes are activated upon each object transfer request, issued by the M processes. A prototype implementation of the minimal model for the server-to-server communication described above has been recently accomplished at NPAC [Haupt96].

2.3 Management Modules

The minimal model in Fig.1 exposes only one system module (Flow Executive) per node. More realistic version of WebVM will contain additional management modules. New modules are added and existing modules are detached from their compute-webs by the *Link Manager* module that is notified by the user or the application

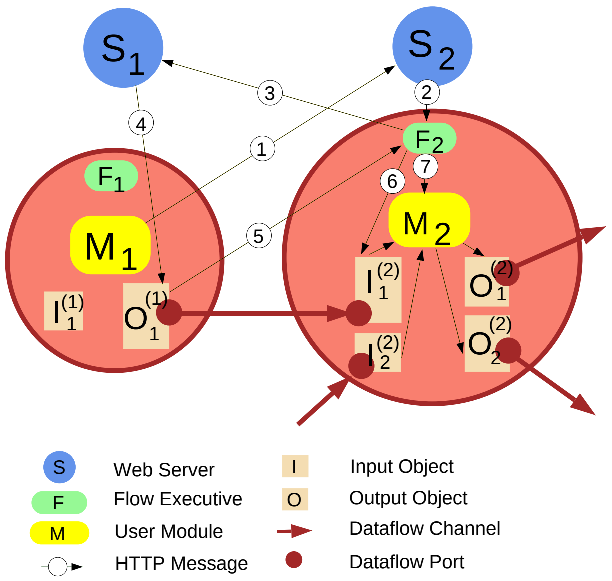

Fig. 1: Minimal model for HTTP+CGI based Server-to-Server Communication discussed in Section 2.2

and updates the compute-web configuration files. These configuration files maintain local connectivity information about all compute-webs attached to a given physical node, and they require some management support to assure atomicity and concurrency control in the multi-user interactive environment. This responsibility is assigned to the *Configuration Manager* module which provides interface to a database that maintains the compute-web connectivity information. In complex systems, we will use relational databases such as Oracle but in simpler compute webs a lighter weight DBMS will be used into the same Web linkage. Optimal placement of new modules on physical nodes in a workstation cluster requires some information about the available overall system resources. These resources vary dynamically and require a dedicated *Resource Manager* module to assure fault tolerance, static resource allocation, and dynamic load balancing via module mobility. We are currently exploring several existing Cluster Management Systems (CMS) [Baker95] and selecting the most adequate solutions to be adapted to and plugged into the resource management sector of the WebVM. Web based CMS support is the main focus of the associated project, MetaWeb, mentioned previously. Management modules outlined in this section provide the practical interoperability platform between WebVM and MetaWeb thrusts.

2.4 High Performance WebVM

The minimal model is built on top of Web servers which are stateless, whereas WebVM requires module state information (such as the current values of all input objects) to be available in the runtime. The minimal model implements the state of the module in terms of the MIME files which is stored on the local disk. Because of the combination of implmentation of the minimal model along with the need for new socket connections to establish each dataflow channel, results in high latency of our server-to-server communication model. A series of new products coming from the Web software vendors, such as Netscape and JavaSoft, open up a new and higher performance implementation avenue for the WebVM architecture.

A **high performance** implementation of the message passing in Fig.1 can be realized in terms of auxiliary servers (referred to as *CGI servers* as in Section 1.4), present in all WebVM nodes, cooperating with Web servers and supporting object and configuration caching. In the minimal implementation of Fig.1, the two large blobs represent document trees associated with servers S_1 and S_2, with modules as heavy–weight processes and objects as MIME files. In the CGI server based implementation, these blobs become object-oriented, multithreaded servers with modules represented as threads and objects as class instances, cached in the dynamic memory.

Several new advanced technologies are coming from the Web software development community and will be explored in this project as candidates for dynamic compute-server support. Examples include the industry products such as distributed Java (RMI) and vendor-neutral database interfaces (JDBC) from JavaSoft, FastTrack servers from

Netscape with LiveWire support for high performance server side scripting, W3C efforts to specify Web-CORBA interoperability protocol.

Our current plan is to start the CGI server development from a minimal Java server per node with its own threaded communication support and a little language interpreter. It will then grow dynamically by downloading the newly required object classes. Operations in WebVM which are critical in efficient perfornace will be gradually be videntified in the testbed and systematically migrated from the HTTP+CGI to the higher performance distributed Java layer.

2.5 Portable Module API

In view of the rapidly evolving Web technology products, a unique technology selection and detailed WebVM specification is a difficult task at the moment. Two possible strategies could be: (a) select the best software platform, e.g. the latest server model from Netscape or JavaSoft and build WebVM in this framework, or (b) seek for a maximally portable and scalable design, which must be then given by the common denominator of various available and coming soon approaches. We will follow option (b) , since it's the safer approach given the current Web technology "soup". This guarantees a larger near term installation base, and fits the prevailing spirit of abstract, platform and vendor independent Virtual Machine concepts.

Our main design objective is to provide a framework which would allow each W3C compliant Web server to participate in WebVM computations, while also providing hooks for more advanced servers to exploit their advantages. The Virtual Machine concept provides a natural methodology to satisfy both constraints. The essential point is the distinction between module API, i.e. "what the user sees" and the WebVM implementation layer. We already described in Section 2.2 the specific HTTP+CGI based implementation of the minimal model and pointed out that Java server implementation would look quite different. Users will develop application specific modules using their favorite languages (C/C++, Fortran, COBOL, ADA, Java etc.) and follow the dataflow metaphor. The minimal module API is built around the abstraction of a named dataflow port (heavy dot in Fig.1) and two function calls: *inp_handle = port_read(port_name)* and *port_write(port_name,out_handle)*. For example, module M_1 would execute *port_read* twice to receive handles for input objects $I_1^{(2)}$, $I_2^{(2)}$, perform the computation, then execute *port_write* twice to pass handles for the output objects $O_1^{(2)}$, $O_2^{(2)}$ to the system layer, and exit. This minimal API is identical for various WebVM implementation modes, be it pure CGI, Java or Netscape/LiveWire. Various sites publishing their WebVM compliant modules can select their favorite implementation strategies and in fact a given Intranet site can contain a mixture of implementation techniques deemed appropriate for individual application needs.

2.6 Scripted Extensions

The conventional dataflow model is most useful for possibly heterogeneous but topologically static distributed applications. However, several relevant scientific computing and information processing problems require dynamically varying topologies. For example, Adaptive Mesh Refinement (AMR) algorithms for hyperbolic PDEs dynamically generate new transient tasks for refined grids, and a WebAMR programmer would like to distribute such processes efficiently over the WebVM nodes.

Similar expansion needs arise in Intelligent Agent-based information processing problems. Agents, represented as chunks of interpretable software sent to the "cloud" in search of specific information, often find it useful to spawn other agents/slaves, send them with suitable subtasks to specialized subdomains, and then collect, rank and integrate the individual information scores.

We initially focus on simpler, more structured and deterministic dynamic compute-webs such as required by AMR. One possible solution involves extending the module API by domain specific calls such as *grid_refine* etc. However, API extension would mix domain specific and generic WebVM components and result in a complexity explosion, since each domain-specific call (and there will be many such calls for various domains) needs to be associated with multi-language binders for all languages supported by WebVM.

We chose instead a more elegant and problem-independent approach based on scripted extensions of the base WebVM model. In terms of the minimal model and its management modules, we introduce another system process called *Script Executive*, and one more routine in the module API – *script_execute(language, string)* — which passes scripts in the application-specific *little language* to the Script Executive for immediate execution/interpretation.

Script Executive will offer a default set of commands such as creating, destroying or modifying the compute-web topologies, as well as mechanisms for installing new application specific commands by users or their domain system administrators.

Script Executive builds its functionality on top of the base WebVM layer, including server-to-server message passing, and hence it can also exploit the distributed interpretation capabilities. For example, a user's command to create a new node in the current compute-web can be implemented with built-in dynamic load balancing: Script Executive can issue a status request to Configuration Managers, select the optimal node based on this information, and finally send the module placement request to the Link Manager in this node. An alternative and often more efficient strategy for load balancing could use periodically refreshed information maintained by local Resource Managers but the scripting channel will offer more direct insight into the realtime operation of WebVM and a platform for refining the existent and prototyping new commands in the WebVM instruction set. System and user strategies can also be more easily integrated in the scripted mode. For example, applications may provide a special handler module to refine or even override the default node placement process and pass the URL of such a handler to the Script Executive as part of the command script.

At the current design stage we do not specify the syntax and the instruction set of the coordination script. The syntax ambiguity might be resolved soon by new products – for example, Netscape can provide an open LiveWire support which can be useful for our purposes. The W3C/CORBA initiative may also come soon enough with standard Web bindings for various language interpreters. Initially, our needs in the scripting sector will be limited and we can tentatively develop a custom small interpreter while waiting for more stable solutions by the Web community. We also note that our scripted extension shares common aspects with other related published specifications, such as: Script Node in VRML 2.0 to specify dynamic behavior of other, more static VRML nodes; or the SQL scripting mechanism in the JDBC interface to relational databases.

3 WebFlow for Visual and Adaptive Programming

3.1 Visual Authoring for the Web

Several high level programming paradigms can be implemented on top of the WebVM layer discussed in Chapter 2. Based on our previous experience with AVS and Khoros, we view visual authoring of application-specific compute-webs as a particularly attractive and useful system integration technique. A Java capable browser such as HotJava offers a natural implementation vehicle for such high level tools. Here the WebVM modules can be mapped on the client side Java threads and then on visual icons in an interactive compute-web editor applet. Visual feedback, corresponding to currently instantiated dataflow channels is implemented via notification messages sent by individual Flow Executives to the JavaVM at the client side. Depending on a particular network security model, these client side threads which act as images of the WebVM modules reside in the browser itself (this is possible e.g. in the HotJava model) or in the personal Java based CGI server (this is required in a more restricted security model of Netscape2).

We expect such WebFlow i.e. visual Web programming paradigm, to be come increasingly popular with the growing complexity of the multi-language Web. Today, a Web developer needs to be a proficient HTML, Perl, Java, JavaScript, LiveWire and VRML programmer, and the WebCORBA initiative will likely extend soon this list of official Web languages. Visual tools for computation authoring such as proposed in the WebFlow model will hide this growing complexity in terms of intuitive visual icons and click-and-drag design metaphors. When sutiably customized for individual communities, such visual authoring tools will facilitate several layers of the advanced application development process, including base module engineering, compute-web design and prototyping, performace visualization, high level integration, runtime navigation, product presentation and marketing.

The first commercial tools in the area of visual Web programming are already available (e.g. JFactory for Java software engineering) and simple Java applet based visual editor prototypes are now also within the reach of the Web research community (see e.g. our NPAC prototype compute-web editor front end at X). We expect a family of such authoring tools to appear soon, specialized for various editing tasks and operating at various granularity levels, ranging from the fine grain Java class or VRML behavior editors, to coarse grain problem solving and system integration toolkits.

3.2 Integrating Navigation, Authoring and Visualization

Advanced computing in science and engineering usually refers to some complex space, with the individual modules operating in physical or functional subspaces. WebFlow visual authoring can be therefore naturally integrated with navigation and advanced visualization. By using suitable spatial metaphors for module icons, one can construct interpolating visual spaces which superimpose visualization functions on WebFlow topology, and use dataflow channels as navigation pathways.

A simple hyper-textual version of this concept was discussed in the previous Chapter in the context of personal server based WebVM management. The coming generation of advanced tools for the visual Web such as VRML 2.0 will integrate VRML with Java to provide a standard development platform for advanced interactive 2D and 3D visualization modules. At the moment, the Web is already populated by nifty Java applets and simple VRML worlds, but there is not yet any organized activity in building reusable computational resources for Java and VRML. Our modular model of WebFlow and the application focus on AMR which requires advanced visualization could effectively contribute to such efforts in the domain of Java/VRML module repositories for scientific visualization.

3.3 WebToolkits as Meta Modules

In our visual WebFlow design work, we plan to focus on some minimal universal editing primitives that appear in a natural way and are useful at various granularities of the underlying compute-webs. Hierarchical design, in which compute-webs of finer grain modules are encapsulated and become coarse grain modules in a larger compute-web, will play an essential role in providing the bridge and navigability between various WebFlow editor instances. We expect such coarse grain Meta Modules to be useful both in the information processing sector (for example advanced WebToolkits for the content authoring, or intelligent agent services) and in the computation sector (for example modular but packaged and integrated aggregates for high performance distributed computing).

AMR is well suited for testing these concepts. The components utilized in AMR can be treated as separate modules which can be combined in a data flow fashion into one large Meta Module - a single node in a distributed WebVM machine. The micro-library would be comprised of exchangable modules such as clustering, interpolators, or even complete multigrid solvers. I/O and visualization modules would be of key importance. Once the prototype is debugged and validated, for the sake of efficiency of the production code, it would be compiled as a single program.

3.4 Research Issues

Our proposed WebVM/WebFlow project will include prototyping, test application, testbed and computer science research activities. Our presentation so far was organized based on the system prototyping requirements. The following chapters discuss the application testbed aspects of the project and the integration of the system and application thrusts. Here, we summarize the research issues to be addressed in parallel with prototyping and/or testing various aspects and subsystems of the WebVM/WebFlow model:

1. *Integration of “computation” (number crunching), “authoring” (visual flow editors, CASE tools) and “information” (documentation, monitors).* Traditionally, these three domains of computing were developed independently by different communities. The Web has gradually blurred these boundaries, but the involvement of the HPCC community is still limited at present. WebVM and WebFlow will bring in HPCC as a driving force in the on-going evolution of the Web technologies.
2. *Scalability of WebVM architecture from Intranets to a World-Wide Computer.*This proposal focuses on Intranet support for adaptive PDEs but we already addressed previously a World-Wide Computer based computation in the RSA130 Factoring-by-Web pilot project. A natural evolution of WebVM towards WWVM can be achieved by initiating concurrently and then linking several Intranet projects, pursued by cooperating HPCC teams in different labs. WebVM scalability issues pertaining to this process will be researched as part of this project.
3. *Scalable Parallel I/O in heterogeneous distributed systems.* Modular design of WebVM will allow us to verify/integrate the ongoing SIO approaches pursued by the HPCC community as well as address their integration with distributed heterogeneous relational and object RDBMS systems, explored by the software industry.

4. *High Performance WebVM implementation strategies (PVM/MPI vs distributed Java).* The interoperability between current generation HPCC systems such as PVM, MPI, FortranM and the corresponding Web technologies such as Java require detailed study and will be addressed in the WebVM design and prototyping thrust of the project.
5. *Performance visualization and realtime monitoring for the distributed computation in the multithreaded (e.g. JavaVM based) client environment.* Java offers an attractive new platform for developing multithreaded applications, both for the multi-server backend and for the front-end interactive visualization and collaboration. Performance/usability tradeoffs between Web (Java, VRML) and HPCC (Nexus, Pablo, HeNCE, CODE) technologies in this area will be addressed in the WebFlow part of this project.
6. *Little languages and their interpreters for distributed computation.* New models for distributed interpreters such as JavaScript/LiveWire are now being offered by the Web industry. These frameworks need to be evaluated as possible candidates for PSE little languages and compared with the corresponding HPCC concepts and activities.
7. *Evolution from MIME based to true object-based (e.g. CORBA) message passing.* Web industry is bringing the full power of object-oriented computing to the Web environment. Public activities such as W3C/CORBA alliance are also in process, while the Web in large is still based on a simple object-based MIME model. Possible evolution scenarios towards an object-oriented Web will be naturally addressed in our MIME->Java framework for WebVM evolution.
8. *Evolution path towards object-oriented distributed multithreaded interpreted scalable multiserver systems.* The ultimate model will be likely based on meshes of object-oriented servers-interpreters. At the moment, the maintenance of such systems based on current multi-faceted Web technologies would be prohibitively complex. We believe that our WebVM design will introduce some practical interoperability platform between several current complementary or competing approaches and an integration framework towards a logically homogeneous system on top of the current Web technology soup.

4 Testbed Application – Adaptive Mesh Refinement (AMR)

4.1 Applications of the Adaptive WebFlow

4.1.1 Requirements. To demonstrate the capabilities of the adaptive WebFlow run on a WebVM we need to select an application area that requires both interdisciplinary software integration and difficult to satisfy computational resources: CPU power, memory, and disk storage. The application is to be of immediate practical importance so it can be used and evaluated by other research groups.

4.1.2 WebVM and WebFlow Testbed. One such application that satisfies these criteria is Adaptive Mesh Refinement for PDE solvers. In particular we will focus on the use of AMR for hyperbolic and elliptic partial differential equations. A prototype of a Web based Problem Solving Environment will be built, that will allow for custom transformation of an existing stable unigrid solver to a stable multigrid solver. Our approach to this area lies not only in its simplicity of using a GUI along with libraries of reusable modules which can be combined into a single dataflow computation, but it will also provide a platform that will satisfy this memory and CPU intensive application. This will make it possible to migrate AMR software, now an exclusive domain of expensive MPPs, to a dynamically configured, distributed memory WebVM environment.

We anticipate a growing interest in multiscale algorithms from fundamental physics (Binary Black Holes)[BBHGC] to practical engineering (multiresolution CFD simulations as pioneered by NPSS and NASA Lewis)[]. We believe WebFlow can evolve to an attractive general problem solving environment for multiscale applications and we view AMR as an initial test of this general capability.

The system will be demonstrated using existing PDE solvers developed by ongoing Grand Challenge research and other projects. The process of building an AMR version in a WebVM to some extent will resemble making visualizations using tools like AVS or Explorer. For example, a researcher making visualizations can test different

filters and mappers to expose the relevant features of his/her data by a quick point-and-click selection of the available modules. Without
visualize the data. Similarly, we want to provide a library of modules such as interpolators and black box Poisson solvers to enable the user to compose his/her AMR based solver to develop a convergent and stable multigrid code. The
and

Visualization modules for AMR will be an integral part of our test suite. Adequate visualization of a multilevel evolution is a challenging task in itself, and it is an active area of research. What is needed, is an adaptive AVS-like visualization scheme and we expect our adaptive WebFlow to offer an adequate framework. Here the visualization modules will be created on-the-fly when the grids are created. Presently packages like AVS are static, therefore to use these packages one must first define a map with all the modules that will be used for the visualization. This can be cumbersome with AMR where several dozen grids can appear at once.

In the following sections we describe the importance of AMR techniques followed by a brief description its principles. The full description of the algorithms used in AMR can be found elsewhere (citation). Then we discuss challenges involved in making visualizations of the dynamical data introduced by AMR. We conclude this section with description of our vision of a WebFlow implementation of AMR on a WebVM.

4.2 Adaptive Mesh Refinement

To address the problem of insufficient resources to solve systems of hyperbolic PDEs with required precision, Berger and Oliger introduced a new technique of integration, commonly referred to as Adaptive Mesh Refinement (AMR)[Berger84]. This technique allows one to compute the solution with a variable resolution, adaptively concentrating computational effort on selected computational subdomains, corresponding to a variable scale of the underlying physical processes. This is achieved by tracking regions in the domain that require additional resolution and dynamically overlaying finer grids over these regions. AMR techniques start with a base coarse grid with minimum acceptable resolution that covers the entire computational domain. As the solution progresses, regions with unacceptably large truncation errors are tagged and finer grids are overlayed on the tagged regions of the coarse grid. Refinement proceeds recursively so that regions on the finer grid requiring more resolution are similarly tagged and even finer grids are overlayed on these regions. The resulting grid structure is a dynamic adaptive grid hierarchy.

The same base unigrid PDE solver – a grid update operator – is applied to each grid on all levels of refinement. The Berger-Oliger algorithm defines the sequence of grid integration (see Fig. X1) and requirements for data exchange between grids. Conversion from a unigrid mode to AMR thus does not involve any modifications of the original unigrid PDE solver. Instead, it becomes a module in adaptive dataflow computations, with data dependencies defined by the Berger-Oliger algorithm.

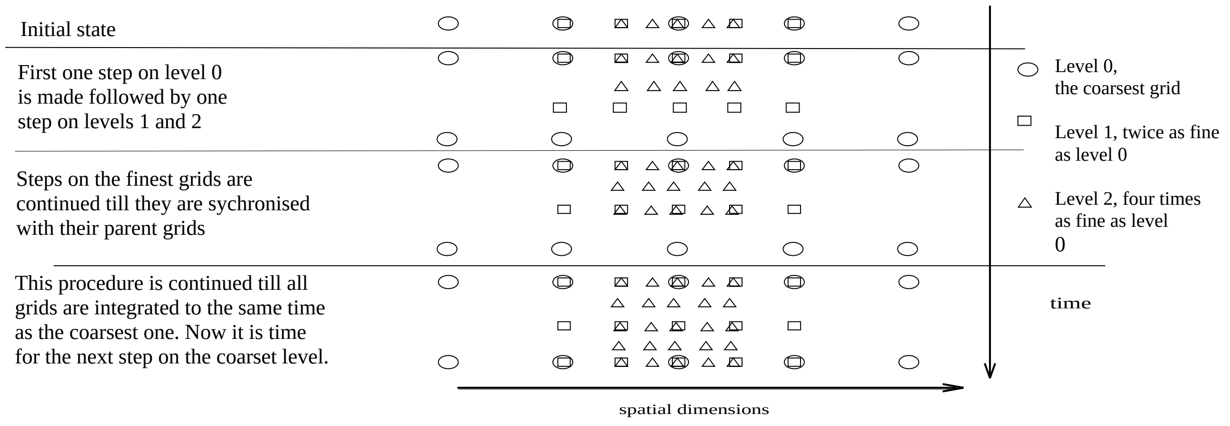
Initial state
First one step on level 0 is made followed by one step on levels 1 and 2
Steps on the finest grids are continued till they are sychronised with their parent grids
This procedure is continued till all grids are integrated to the same time as the coarsest one. Now it is time for the next step on the coarset level.
Level 0, the coarsest grid
Level 1, twice as fine as level 0
Level 2, four times as fine as level 0
time
spatial dimensions

Fig. 2: The sequence of grid integrations according to the Berger-Oliger algorithm. Every p steps regridding take place (not shown).

The Berger-Oliger algorithm can be divided into several relatively separate components – modules. These components fall into three categories: operators that act on a single grid, operators that act between two grids, and operators that modify the grid hierarchy.

The first component is the update operator that moves the solution on the grid from time t to $t + \Delta t$, where the size of Δt is proportional to the spatial resolution on given refinement level in order to satisfy the CFL condition. The update of the boundary conditions, if these change in time, can be included here, or they can be implemented as a separate module. The next module is the truncation error estimator, which is usually based on the Richardson extrapolation [Rich00]. The output of this module is then processed by a clustering routine which transforms a irregularly distributed set of flagged points into Cartesian grids [Berger91, Chrisochoides]. At this moment the actual regridding takes place, namely the old grid hierarchy is replaced by a new one, in other words, the data flow scheme is updated. Once the new structure is created, new grids are initialized. This is done by spatial and temporal interpolations [Klasky95] that can be encapsulated as stand alone modules. Finally, there is a need for a module that performs flux corrections [Berger94]. This insures conservation at grid interfaces by modifying the coarse grid solution for coarse cells that are adjacent to a grid. In addition, support for I/O and visualization must be incorporated into this scheme.

The modularity of the Berger-Oliger algorithm make it an excellent target for dataflow implementation. Moreover, this paradigm makes it possible to simplify development of even more complex PDE systems: systems of coupled hyperbolic and elliptic PDEs. Elliptic solvers, to speedup convergence also employ a multilevel integration scheme. Contrary to AMR, the multigrid methods [Brandt] uses coarsening rather than refinement of grids. There are two basic ways to incorporate multi-grid into our AMR scheme.

The first method assumes that on every time step when we must solve an elliptic problem then we multi-grid off of each grid in the distributed adaptive grid hierarchy separately. This then involves adding data structures to handle the coarse grids that are used for the coarse grid corrections in the multi-grid algorithm. Each fine grid problem is viewed as solving a Dirichlet boundary problem, where boundaries remain unchanged in the elliptic problem. This approach invites a hierarchical dataflow model: each AMR module exhibits a fine structure with an internal dataflow which can be visually programmed. The other method involves solving the elliptic system over the entire hierarchy. This is a Multi Level Adaptive Technique (MLAT). In this approach, additional grids might have to be created in order to acclerate the convergence of the MLAT solver. A more detailed discussion of MLAT can be found in [KlaskyPhD]. MLAT will be used as our testbed for static AMR implementations. The MLAT technique is more challenging to implement in a dataflow paradigm and certainly requires some research in order to define optimal granularity of modules and their interactions with the flow manager.

Our expertize in AMR comes from the research done in Numerical Relativity and**GET APPS FROM MANISH**. We started with a one dimensional sequential implementation to investigate critical phenomena in black hole formation [Choptuik]. This experience became a foundation for a general purpose 3–dimensional, data parallel implementations in HPF[HauptKlasky] and in C++/MPI (DAGH) [Parashar]. Our experience from MLAT is again based on Choptuik's early work [ChoptuikMLAT] and later from Klasky[KlaskyPhD].

4.3 AMR as a WebFlow Application

AMR techniques allow for a significant reduction of the required resources by preforming computations with variable resolution, to obtain accurate solutions in critical regions of the computational domain. The demands for processing power, even with AMR, far exceed the current high-end workstations when performing multidimensional calculations. Therefore the current implementations of the AMR are targeted mainly to massively parallel computers with hundreds on computational nodes, or vector supercomputers.

Due to the high bandwidth and low latency communication networks typical for tightly coupled multiprocessor machines, it is efficient to implement the unigrid solvers in a data parallel way either using directly high performance message passing libraries, such as MPI, or data parallel languages such as HPF, releasing this way an enormous computational power of MPPs. Advances in compiler technology, in particular C++ and Fortran 90/HPF made the problem of dynamical memory management easier to code efficiently, and in fact port it to the parallel distributed memory systems. However, parallel implementations of AMR add an additional level of complication to the already complex AMR algorithm. Challenges include load balancing, minimizing communication costs, handling irregular boundary conditions and overall parallel efficiency.

In order to use AMR solvers on adaptive WebVM we need an approach that will eliminate critical dependency on the latency of the system. A natural solution is to expose task parallelism rather than data parallelism. Each node is assigned a whole grid, all computations are performed locally, independently of computations executed on other processors. The nodes communicate with each other at the regridding and restriction phases, which force the nodes to synchronize.

The modular structure of the Berger-Oliger algorithm makes it possible to implement AMR in a classical data flow paradigm on WebVM. At any time instance we have a set of modules, most of them are identical unigrid update operators. The data dependencies between the modules are uniquely defined by the Berger-Oliger algorithm, as shown in Fig. X2A.

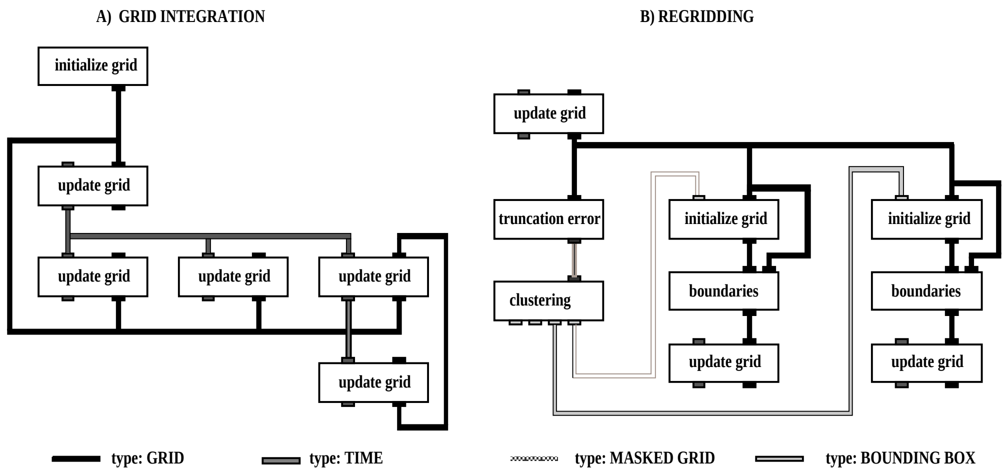

Fig. 4: A) Possible dataflow implementation of AMR PDE integration according to the Berger-Oliger algorithm. Every p steps regridding is performed, not shown here.
B) Possible dataflow implementation of the regridding operation. The regridding starts on the current finest level, and proceeds down to the level on which regridding was initiated. For clarity, enforcing proper nesting of grids not shown here. The final step of regridding is removing the old grid structure, not shown.

The extension we propose adds adaptivity to the data flow, in a sense that we allow the computational modules at the regridding phase to interact directly with the flow manager in order to dynamically add and remove modules. Regridding can be represented in terms of dataflow as shown in Fig.X2B.

All modules involved in the regridding have critical importance for both efficiency and stability of the PDE integration. The optimal choice of parameters such as the threshold value of the truncation error or clustering efficiency as well as particular algorithms for clustering or interpolation is application dependent. More, initialization of the fine grid includes imposing boundary conditions on the newly created grids. This is in obvious way correlated with the integration method adopted in the update module. Users are expected to choose the modules which best fits their application and sometimes replace modules by their own version.

4.4 WebVM/WebFlow Testbed for Adaptive PDE

It is the subject of our proposed research to select the best suited technology to implement the idea of WebFlow. Moreover, we anticipate that the current rapid development of Web technologies, driven both by academia and industry will sustain during the proposed lifetime of this project. Therefore is it crucial that the dataflow implementation of AMR is independent of particular implementation of the WebFlow on WebVM.

The primary target of our implementation will be a collection of UNIX workstations or personal computers in a Local Area Network (LAN) possibly using ATM or vBNS. Even though the World Wide Web allows for an attempt to create a literally World-Wide Web Virtual Machine, we do not see any benefits of running AMR

PDE solvers in such environment. The experience gained running this very demanding application over LAN will enable us, or others, to design a mature implementation of geographically distributed WebFlow. Another interesting target of the WebFlow is networks of shared memory SMPs, and/or networks of dedicated clusters of workstations interconnected by high performance, low latency switches.

In each case, there is a heterogeneous environment, and therefore portability of codes is an issue. The computational engine, the unigrid PDE solver, will be implemented in Fortran 90 and/or C++. An investigation will take place into the possibility of implementation of at least some of the proposed modules in Java, JavaScript or other emerging platform neutral languages. Inclusion of multiprocessor machines or low latency networks will make it possible to develop HPF or MPI based Fortran 90/C++ SPMD modules. It is important that conforming to the current practice to develop codes in Fortran and C++ will allow for easy migration of codes between WebVM and MPP environments such as DAGH.

The nodes of the WebVM will communicate with each other using the industry standard http protocol (here we treat SMPs and the dedicated clusters as a single node), exchanging data of custom defined MIME types. For example, we may define data type GRID which will include a grid id, bounding box, spatial and temporal resolution, and actual functions defined on the grid. Design of the MIME types for AMR, and guidelines for defining appropriate MIME types for other applications are part of the research we propose here.

HTTP connections are stateless, and nodes of WebVM are only loosely coupled. In fact we expect that typically all nodes of the WebVM will be used in a time-sharing mode, and synchronization of the nodes will be an issue. It might then be necessary to first store all data to be sent to another node on the disk with a file name extension corresponding to their MIME type. Then, instead of sending actual data, their owner will post their URL. The receiver will import them using CGI GET request, Java socket connection, or any other method proper to the actual implementation of the WebFlow. Admittedly, the frequent I/O operations will add to the latency of communications. However, this feature of the proposed implementation has important advantages; namely, supporting fault tolerance and visualization.

LAN based computations suffer from unpredictable load of nodes, and variable size of available resources. A failure of a single workstation, for example turned off by its owner, results in the loss of entire computation. Since the applications we target, the AMR PDE solvers, are very computationally extensive, the probability of a failure is relatively high. Typically, to defend against such a situation, an expensive checkpointing is employed that involves a periodical complete coredump. The modular structure of AMR makes it unnecessary. Thanks to storing data on disk before internode communication, we risk only lost of a few iterations on a grid. The flow manager, extended with timer capabilities, can reschedule the grid updates to another node, if the results have not been returned on time. Thus, by keeping the input data on disk in a platform independent MIME conforming format we can easily implement a cheap substitute of checkpointing and process migration.

The data stored on disk can also be used for visualizations. As long as the data format includes enough information to identify spatial and temporal position of the grid, the data can be used directly for making visualizations. Some of the data will also be used to facilitate internode communications. The visualization routines can process the data concurrently with their generation, or they can be run after the evolution is completed as a stand alone Web based application. To make the latter possible, a record of which nodes wrote the data must be maintained which mirrors the actual dataflow.

The structure of data dependencies (order of grid updating) suggests several ways of optimizing the overall implementation of the AMR. For example, a grid on a coarser level cannot be updated before restrictions from finer levels are completed. The fine levels, even though they span smaller physical domains, may be comparable in size to the parent grid, because of their greater density of mesh points. Moreover, since the refinement is made in both spatial and temporal dimensions, the actual computational task on a fine grid may be larger than that on the coarse grid. All this indicates that a naive distribution of one grid per node may be in practice very inefficient. Some load balancing will be certainly necessary to increase parallel efficiency of the system and increase throughput. Therefore, it may be necessary to assign processing of more than one grid to a single processor. It can then be possible to reuse ideas implemented in DAGH for load balancing imposing the constraint that no grid can be splitted among processors.

4.5 Webflow testbed for AMR data visualization

AMR data visualization will directly benefit from our scripting language. Here AVS-like visualization modules and computational grids will be created and destroyed on the fly. Since we are looking at evolutions, data must be written at every several time steps. This then has the potential of creating Terabytes of data output. We will need database support from Illustra/Oracle in order to manipulate the enormous amount of data that will be generated by these codes.

The AMR application has several distinct properties that make visualization from Webflow enticing. One such property is the direct access to visualization packages such as VRML 2.0. Here 3D data of each grid can be visualized and HTTP links can be generated to go from one grid to another. Since VRML visualizes data in only 3D, we propose to build an interface where one can click on the 3D image to obtain a series two dimensionally slices, via Java[VH,NCSA]. These two dimensional slices can be further decomposed into one dimension line plots that can also be viewed with Java. Once the visualizations are on Java and VRML, collaboration can instantly occur between various researchers, since all the data will be accessible on the WWW. This also provides us with an unique way to publish results. Calculations are now accessible to the public, and can easily be refreshed via an HTTP address. An important part of scientific visualization is collaboration, and VRML 2.0 will allow various researchers to interact with the AMR visualization at the same time. In this way researchers can view and manipulate the same image to help the scientific process of extracting meaning from the data.

Data visualization for AMR can be viewed as another module that will be used in the Berger & Oliger routine. This module will be composed of a WWW page. This module will have dual functionality. Its first function will be to write the data for each grid. In order to allow for machines to go down, we will periodically dump the state of the grid functions using this module. This can be used to restart the calculation. Data output will be readable by any Web browser, supporting per-grid visualization. The second function of this module will be to interactively display the data as the AMR process is running.

Currently we are actively involved in this area of research. Our current implementations write a new HDF (binary compatible from machine to machine) file for each grid that is used in the calculation. For one dimensional AMR, Choptuik has developed a client server data visualization package. Visualization is only one part of the I/O of AMR. Storing the data from various runs in a coherent manner is another task which will require extensive database support. We will use Web linked relational or object DBMS as already explored in other data intensive problems [].

5 Statement of Work

This proposal concentrates on designing and prototyping new technologies to support scientific and engineering computations using resources available on World Wide Web. We will demonstrate the feasibility of implementing a Web Virtual Machine to carry adaptive dataflow computations. Since the focus of our research is to select the best implementation strategy, a testbed environment will be created which will allow for easy experimentation with different versions of the WebVM, including validation of the whole concept and benchmarking. The testbed will comprise three elements: the WebVM, a suite of benchmark applications, and API ,making possible the implemention of applications in the WebVM environment. It is crucial that the benchmark suite applications are included since it is of vital importance for potential users and suficiently challenging to convince communities of users and developers about practical importance of WebVM/WebFlow ideas, in addition to Computer Science interests.

Consequently, we will follow a three-prong implementation plan, including: (a) system component focused on base WebVM/WebFlow design and engineering; (b) application component developing the benchmark suite in the PDE/AMR domain, and (c) the testbed component, coordinating the system and application thrusts.

Month	System	Applications	Testbed
1-2	Design of a portable module API. Implementation within the miniml HTTP+CGI model. Develop language binders for C/C++, Fortran90, Perl, Java and VRML.	Analysis of the AMR and MLAT algorithms in order to define an API. Feedback to the systems group based on algorithms requirements.	Extend the current proof-of-the-concept WebVM layer operational at NPAC to support API design testing.
3-5	Design of the module publication framework and the file format for compute-web fonfiguration files. Prototype implementation in the WebTools model. Publication of the base system modules: Flow Executive, Link Manager, Configuration Manager.	Select a representative suite of PDE solvers for the testbed. The suite must contain also simple codes, such as wave equations (1d and 3d) and Laplace's equation for early tests.	Develop more modules according to the API specifications to test early implementation of WebVM. Develop simple visualization modules based on VRML 1.0
6-7	Deliver the minimal WebVM layer to the testbed team. Develop minimal Resource Manager module and establish linkage with other CMS modules in the MetaWeb project.	Collect selected codes and generate reference results which will be used for validation of the testbed. Collect elements of the MLAT/AMR support library (interpolators, clustering routines, etc)	Enhance visualization modules (in Java and VRML) in order to test static WebFlow.
8-10	Prototype Java applet for WebFlow visual editor and couple it with the WebVM layer.	Select and test components of the MLAT support library that will be converted to the WebVM modules.	Start WebFlow implementation of MLAT applications, as they do not require support for adaptive data flow.
11-12	Extend module API by scripted language interface. Design and implement Script Executive module ("little language" interpreter).	Select and test components of the AMR support library that will be converted to the WebVM modules.	Test static DataFlow using MLAT. Test and improve visualization modules. Implement AMR modules.
13-16	Develop high performance version of WebVM based on Java server technology.	Work on convergence and stability of codes. Exchange experience with other groups.	Develop scripts to control dynamic DataFlow as required by AMR.
17-20	Develop support module library for collaborative interactive visualization using integrated Java and VRML 2.0.	Test and validate MLAT and AMR implementation.	Benchmark different variants of the WebVM. Implement new visualization features.
21-24	Package, document and disseminate the WebVM software.	Publish AMR results using visualization tools on the WWW	Package, document and disseminate the testbed software.

REFERENCES

[Baker95] M. Baker, “Cluster Computing Review”, November 1995
`http://www.npac.syr.edu/techreports/html/0700/abs-0748.html`

[ChaRi96] Mani Chandy, Adam Rifkin, “The Caltech Infospheres Project”
`http://www.cs.caltech.edu/˜adam/CALTECH/infospheres.html`

[Cowie96] J. Cowie et al., “RSA130 Factoring by Web”
`http://www.npac.syr.edu/factoring.html`

[DoCa96] J. Dongarra and H. Casanova, “Netsolve”
`http://www.cs.utk.edu/netsolve/`

[FoKe96a] Ian Foster and Carl Kesselman, “The Nexus Multithreaded Runtime System”,
`http://www.mcs.anl.gov/nexus/index.html`

[FoKe96b] Ian Foster and Carl Kesselman, “Globus: Infrastructure for Internet Computations”,
`http://www.mcs.anl.gov/globus`

[FoKe96c] Ian Foster and Carl Kesselman, “I-WAY Software Infrastructure andProgramming Tools”
`http://www.iway.org`

[FFCRC95] G.C. Fox, W. Furmanski, M. Chen, C. Rebbi and J. Cowie, “WebWork: Integrated Programming Environment Tools for National and Grand Challenges”, June 1995
`http://www.npac.syr.edu/techreports/html/0700/abs-0715.html`

[FoFu95] G.C. Fox and W. Furmanski, “Use of the National Information Infrastructure and High Performance Computers in Industry”, July 1995
`http://www.npac.syr.edu/techreports/html/0700/abs-0732.html`

[FuFo96] W. Furmanski and G. Fox, “Prototyping a WebVM based WebFlow Environment for Distributed Computing and Visual Programming”, March 1996
`http://www.npac.syr.edu/projects/webspace/doc/webflow/dual96.html`

[NPAC96] NPAC Team, “Web based HPCC at NPAC” – List of Documents and Demonstrations
`http://www.npac.syr.edu/projects/webbasedhpcc.html`

BIOGRAPHICAL SKETCHES

SUMMARY PROPOSAL BUDGET

CURRENT AND PENDING SUPPORT

FACILITIES, EQUIPMENT AND OTHER RESOURCES