Chapter 3

Web-Related Implementation Issues

This chapter concentrates on the design and implementation issues of the WWVM not addressed in Chapter 2. Among these can be counted the implementation of Web-based GUIs, configuration of Web (HTTP) servers that will be extended as WWVM servers, WWVM user accounts, access to back-end computational resources, determination of system facilities to be provided to users, and system security. These issues are mostly dependent on the underlying system and targeted use of the software systems and are specific to Web-based software systems.

In order to field-test the WWVM software on a small scale and determine the required system functionality and user preferences, WWVM was configured in a single-execution/single-machine mode (i.e., stand-alone mode). This limited version of WWVM with more emphasis on the Web-based user interface and system facilities is called a *Web-based parallel programming environment* (*WPPE*).

The next section discusses the role of HTML, frames, JavaScript, and Java in the realization of Web-based GUIs. Section 2 summarizes the distinctive features of three prototype WPPEs and describes their system architecture. Section 3 reviews the advantages due to the client-server-based architecture of the WWVM. Section 4 lays out several design alternatives and discusses the tradeoffs associated with various design decisions. Section 5 presents the system security measures that are crucial in order to avoid comprising the security of the whole system.

3.1 Using Web Technologies for Graphical User Interface Design

It is expected that in the future the Web will be the standard user interface in many organizations for accessing computational resources. The Web browser and other Web technologies such as HTML, JavaScript, Java, and VRML provide a uniform user interface platform on every type of computer in the world. As a result, X-windows on Unix platforms, or Windows environments on personal computers may be replaced by Web browser-based user interfaces. This ubiquitous way of resource delivery imposes a novel set of constraints on user interface design [RFP+ 96]. The next section discusses the role of portable and platform-independent Web technologies in the realization of the design and details of the lessons learned from this process.

3.1.1 Using HTML Forms and Frames for GUI Development

There are a number of problems associated with using Web technologies such as HTML and frames for designing GUIs. First of all, there is a fundamental difference between the type of interaction supported by HTML and the forms of interaction to which we are all accustomed in graphical user interfaces. In GUIs, operations typically take the form of the user selecting an

operand or operands through direct manipulation and then applying an operator by means of a menu selection or keyboard accelerator. In HTML-based user interfaces, there is no notion of selecting objects per se. Instead, the page the user is on is viewed as an implicit operand, and therefore the user can select a command to apply to that operand. In effect, an HTML interface can allow the user to apply a number of different commands to a single object, or a single command to one of a number of different objects. Commands that take multiple operands are much harder to implement.

The widget set available through HTML's forms capability is limited to submit buttons, radio buttons, checkboxes, pop-up menus, single/multiple selection scrolling lists, text fields and areas, mapped images, and text type-in widgets. It is not possible to combine a submit button or anchor behavior with a pop-up menu, or to include icons in menus or scrollable lists to provide the sort of command selection model that is present in so many user interfaces. There is no way to provide constraints on selection elements (e.g., toggling "list Java files" causes a file list menu to filter out non-Java files). GUI designs sometimes coincide with tradeoffs such as using custom-made image buttons versus the native buttons provided by browsers. Web pages with custom image buttons instead of native ones look more uniform across platforms, but users must learn anew to recognize buttons for each idiosyncratic application.

The behavior of traditional GUIs can be approximated by using multiple frames, keeping the operands and operators in separate frames, and filtering all the submit and select actions through JavaScript functions before submitting them to the server. One common use of frames and JavaScript functions together is to present HTML pages that are multiple screens in height. These pages can be replaced with multiple frames, one of which contains the global buttons that are always visible while manipulating the document with the others. The introduction of JavaScript functions provides better support for more sophisticated applications, making user interaction

more satisfactory and implementation cheaper and simpler. Yet Java, in addition to handling everything supported by HTML, forms, and JavaScript, also provides more advanced tools and utilities for developing GUIs. In the next two sections, some advantages of using JavaScript and Java will be reviewed in regard to user interface design.

3.1.2 Advantages of Using JavaScript

*Interactivity***.** There are only two ways for the browser to transmit information to the server from an HTML document. Pressing a submit button transmits the widget state, while selecting an anchor transmits a request to follow a hypertext link. The action associated with a form is restricted to only one URL, which inhibits having different actions with different submit buttons. Until a submit button is pushed, it is not possible for the server to determine anything about intermediate activities that a user might perform, such as typing text into an input field, toggling radio buttons or check boxes, selecting items from menus, moving the mouse, and so on. JavaScript adds dynamism to the HTML pages through its ability to check widget states as soon as they are entered, and by being able to "submit" values explicitly without needing to use buttons or links. Moreover, JavaScript 1.1 made it possible to change dynamically the target of the action of a page or button. Consequently, it is possible to implement many features of sophisticated user interfaces, such as immediate feedback to the user over the Web.

*Asynchronous Communication with Server***.** It is not possible for an application to preempt the browser's activity or provide any asynchronous communication using only HTML. For example, it is not possible to notify the user asynchronously about the results of a background task or remind the user to save work. JavaScript's time-out mechanism and alert dialog capability can be used to do these kinds of things. Furthermore, JavaScript modules can open HTTP connections asynchronously.

*Transmission granularity***.** Each user action (submit or select) causes an entire new page or frame to be transmitted back to the browser over the network. There is no way for the application to cause an incremental update of a portion of the display. Even on high-bandwidth local area networks, transmitting and rendering large pages is time consuming; for distant browsers it becomes the dominant cost. Hidden frames manipulated by JavaScript functions can be used to manipulate only the required field values and send only small frames to the server when required.

Viewport positioning. Another problem is that HTML and HTTP do not provide effective control over viewport positioning. Because browsers cannot inform the server of the browser's window size or exact viewport position within a document, a server cannot cause the browser to scroll to a particular location. This means that it is often necessary for the server to refresh a whole page by forcing a redirection on the browser that would otherwise not be necessary simply in order to scroll to the desired position on the page. Although it is possible by using named anchors to tell the browser where to scroll to on the new page, this is very coarse-grained control.

Graying out invalid options. HTML does not support a built-in mechanism to gray-out either graphics or menu options that would have the obvious benefits of alerting the user of the existence of inapplicable commands and preserving registration. This means that the interface designer is forced to use JavaScript to disable or enable options dynamically according to users' actions and to generate two sets of graphics: one for active commands, and one for any inapplicable commands.

Pop-up menus. The WWVM user interface has a large number of buttons that will invoke system utilities and functions. The number of buttons will only increase, as the system becomes more sophisticated. Because HTML does not support pop-up submit buttons, the obvious and familiar behavior of a menu bar cannot be implemented. The irony of this is that associating an

action with a menu selection is probably the most common form of interaction that users have with menus. The only "correct" model for command menus in HTML is the exhaustive enumeration of the commands as submit buttons. When the number of commands becomes too large to support in this way, an alternative way is to partition the commands into broad classes and put the commands on menus and the name of each menu in front of it as a submit button. This means that the user must select an option from the menu and then click on the submit button to execute the selected operation. This is non-standard, but an applicable solution. In contrast, Javascript allows actions to be dynamically associated with menu items.

3.1.3 Advantages of Using Java

Java is an excellent resource for building graphical user interfaces that we are used to seeing in many commercial software packages. Java functionality includes that of HTML, frames, and JavaScript. Therefore, Java goes one step further in solving the problems that were discussed in the previous two sections. Some additional properties not mentioned before follow:

Keyboard Accelerators. Binding the "`Enter`"or "`Return`" key to submit is an interesting special case of binding the default selection. As in most GUI design tools, Java has utilities for assigning actions to selected keyboard combinations.

Control over page appearance and registration. HTML explicitly yields the rendering of decisions to browsers. This has many advantages for browsing hypertext documents, but it proves awkward for interface design. An application has very little control over the location of displayed objects on the finally rendered HTML page, or on how and to where the browser will scroll the page if it does not fit into a single screen. Java allows explicit registration of user interface items on a page.

Scrollbars in textareas. It is not possible to control the position or presence of scroll bars on text area widgets in HTML forms. This is a particular problem, because the default of most widget sets is to put the vertical scroll bar on the right of the text widget and the horizontal scroll bar on the bottom. For large text areas, this often results in the vertical scroll bar being scrolled off the right of the user's viewport, and the horizontal scroll bar being scrolled off the bottom of the viewport. For some applications it may be desirable to have scroll bars on all four sides of a viewport. Java gives full control on the placement of scroll bars.

3.2 Web-Based Parallel Programming Environments

The World-Wide Web (WWW) has emerged as an exciting and innovative front-end to the Internet. It provides users with a uniform and convenient means of accessing the wide variety of information resources (pictures, text, data, sound, and video) on the Internet. On the other hand, Web browsers make the Internet a more user-friendly environment by integrating mechanisms to access all these resources within a single tool.

A *Web-Based Parallel Environment* (*WPPE)* is a software infrastructure that provides a Web-based interoperable user interface to access user accounts and allows the use of high-performance parallel computing platforms and software on remote WWVM computational servers. It also provides the tools and facilities required to develop parallel programs without leaving the WPPE. Users do not need to log into a Unix account or type Unix commands. Once they supply the required username and password, they are logged into their accounts and can use the computational facilities through the Web interface. Even anonymous users may be given a limited set of privileges to access these resources.

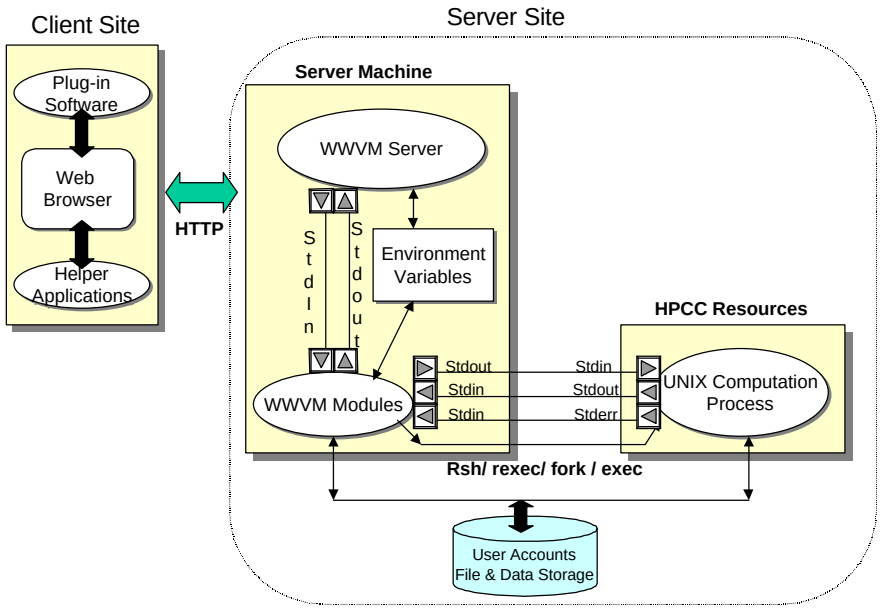

Figure 3 1. Illustration of the WPPE architecture.

In WPPEs, a WWVM server does not connect several network resources as a message-passing or dataflow machine, but only provides access to selected back-end computational resources, generally a single parallel computer or a workstation farm. Three different prototype WPPEs were implemented and were primarily used for educational purposes and for software demonstrations at Syracuse University and the Cornell Theory Center. Their use and evaluation by a large number of people helped to improve the functionality provided by the WPPEs, and indirectly by the WWVM, and to identify critical issues in the implementation.

The WPPE prototypes presented in this chapter were developed using standard Web technologies. The interactive Web, high-performance computation back-ends controlled by a central WWVM server, and HTTP-based communication between the browser and server represent critical enabling technologies in this framework. HTML is used mostly for static components, while the use of Java and JavaScript ensures interactivity and visual animation at the client-site.

3.2.1 WPPE Architecture

The WPPE architecture is built upon a three-tier, client-server paradigm that separates the presentation of services and results to the user (i.e., user interface) from its internal command processing, data storage, and computing. The three layers of this architecture are the Web browser user interface, the WWVM server, and the back-end high-performance computing resources, as shown in Figure 3-1.

It should be noted that there does not exist a virtual machine layer, because WPPEs only provide access to individual computational resources; they do not try to connect the resources together as the full version of WWVM does. The client demands access to remote parallel high-performance computing resources, and a central WWVM server coordinates the accesses to the local computational resources and helps to maintain the user accounts on the server-site. The server site carries the heavy load of all the computation and file storage.

Three tiers of the WPPE architecture can be summarized as follows:

- *The Web-Based User Interface*: Since client site requirements for using WPPEs are minimal, only a JavaScript- and Java-enabled Web browser needs to be available on the client machine. The Web browser software is responsible for displaying HTML documents and Java applets that are loaded from the server site. The Java language helps to provide an

interactive user-interface and well-developed graphical utilities for visualizing the results and behavior of programs.

- *The WWVM Server:* The WWVM server is the core of the system. It accepts requests from the browser and activates the CGI modules responsible for requested operations. The requests may be as simple as fetching a file from a disk or as complicated as compiling a code and showing the results graphically using some kind of visualization tool. CGI modules in this layer provide the most common method of intervening between user interface and high-performance computing resources and the user accounts in the third layer. CGI modules fork UNIX processes on selected high-performance computing machines in order to perform requested services, and they collect results or error messages from these forked processes. Some of these processes conduct file and directory operations, while others activate compilers, linkers, loaders, and other services on the back-end parallel machines. Since CGI modules are written in Perl, a Perl interpreter should also be installed on the server machine. The entire set of related HTML files with JavaScript extensions, Java class libraries, and temporary configuration files are stored on the server site.
- *Back-End High-Performance Computing Resources.* The back-end, high-performance computing platforms own huge computing resources (CPU time, memory, bandwidth, disk space). Upon a user's request, the server maps the computation (possibly involving parallel or distributed subtasks) onto the available computing resources. In most cases, WPPEs use the compilers of the sequential and parallel languages installed on these machines. In some systems, a front-end machine of the same type as the nodes of the parallel machine may be able to compile the programs separately. Web/HPF provides access to IBM's and PGI's HPF compilers on the IBM SP-2 platform, while VPL allows compilation of Fortran 77, C, Fortran

90, and HPF codes on a DEC Alphafarm workstation cluster. MPI and other runtime support or message-passing libraries can be used similarly.

3.2.2 Use of WPPEs in a Research and Education Setting

More and more colleges, universities, schools, companies, and private citizens connect to the Internet either through affiliations with regional not-for-profit networks or by subscribing to information services provided by for-profit companies. All the improvements in Internet and Web technologies have opened many ways for educators to overcome time and distance in order to reach students. For educators, the WWW provides an exciting new opportunity for distance teaching and learning. The WWW and its digital libraries offer a powerful and continuously growing reservoir of educational material. Electronic mail, computer conferencing, and electronic bulletin boards facilitate communication among class members. Built-in, server-site extensibility mechanisms such as CGI and client-site support with Java, JavaScript, and helper applications open the way to worldwide distributed collaborative learning/teaching environments. The computer industry adds value in terms of quality browsers and multimedia virtual reality front-ends.

The WPPEs supplement the distance-teaching efforts by providing virtual programming laboratory functions using Web browsers. They can be used for distant education as a part of collaborative distance teaching environments.

The original motivation for developing WPPEs was to use them in a research and education setting. The intention was to build a generic, Web-based front-end that could be used for supervised and unsupervised demonstrations of developed software, as well as on-site and virtual training workshops with little or no modification. Table 3-1 summarizes some of the features and

constraints of WPPEs targeted for different areas. The requirements and constraints affect the design choices. Below, the main features of each of these four areas of usage are summarized:

- **Supervised demonstrations of developed software.** Typical examples include demonstrations of software products that are outcomes of an institution's research projects. The system should allow a bit of programming for the sake of demonstrating the flexibility of the software.
- **Unsupervised demonstrations of developed software.** Examples of software tools developed at a research center are posted on the WWW for trial and evaluation by interested parties from all over the world. Anonymous users are bound to running a predefined set of demonstration programs.
- **On-site training workshops.** The purpose of these workshops is to teach advanced concepts in programming or the use of parallel machines and tools. Users can access the computational facilities as well as educational material from the same interface, the Web browser. The lifetime of the user accounts is limited to the duration of the training program, which is typically just a few days. Users mostly view the example codes and practice the newly taught concepts by executing slightly modified versions of those codes. Setting up a full UNIX account for such a short duration would be cumbersome and would require superuser privileges.
- **Virtual training workshops or semester-long parallel programming labs.** These usually take several months to complete, and some or all of the registered attendees may be on remote locations. They usually require more involved programming on different HPCC systems.

Usage	Amount of Programming	Users	Location of Users
Supervised Demos	A little bit	Known & trusted	On site or not
Unsupervised Demos	None	Not known	Anywhere
Onsite Training	Considerable	Known	On site
Virtual Training	Considerable	Known	On site or not

Table 3 1 Potential WPPEs and their associated constraints.

WPPEs recently began being used in evaluating newly developed software among collaborating research institutions. Several commercial companies already have "try-and-buy" programs that involve distributing their software on CDs with temporary licenses for potential customers. A similar need arises when a research institution plans to adopt a public domain software to use as part of its development cycle. Since many not-for-profit public domain software products do not satisfy all the criteria demanded by a commercial package, it may be crucial to test it under different circumstances before adopting and installing it on local computing platforms. This installation process sometimes takes several days, depending on the tools provided. It would be nice to assess the quality of the software before adopting it. Furthermore, a tool like VPL can help to post new patches to the current software on the Web, along with a running copy of the latest version of the software. For example, VPL users can follow the day-to-day status of NPAC's "Java+MPI" project targeted towards writing SPMD-style Java code with calls to MPI message-passing routines.

3.2.3 Completed Prototypes

Implementation of three different WPPEs was completed by this time. Each of the following systems represents a different design alternative, as will be explained in the upcoming sections.

The first one is targeted towards supervised and unsupervised software demonstrations, while the second and third are mostly used in on-site and virtual workshops and in semester-long parallel programming labs.

3.2.3.1 HPF and Parallel C++ on the Web

A Web-based demo prototype called “PCRC Web Demo” [CDL 95] was used to display the status of the PCRC [PCRC 95] project at the Supercomputing’95 conference. NPAC’s F90D/HPF compiler and Cooperating Systems Corporation’s parallel C++ compiler, which share the same common runtime system, were demonstrated through a Web interface. The user interface for this system was implemented using a standard HTML and Web form interface. This single-user system was later used for other supervised demos.

3.2.3.2 The Web/HPF

With C. Hecht, K. Barbieri, and A. Trefethen, the WPPE used for the PCRC Web demo was tailored for the Cornell Theory Center (CTC) environment, and named as the Web/HPF module [MehB 97]. The Web/HPF has been being used as a front-end to an IBM SP-2 parallel machine on the Andrew file system in CTC’s Virtual Workshops since the beginning of 1997.

The Web/HPF, using an Apache-based WWVM Web server [Apac 96], provides access to users’ UNIX accounts and allows users to compile and execute HPF codes on the SP-2 via an HTML, forms, and JavaScript-based Web browser interface. Users also have access to the traditional UNIX text editors such as vi or emacs, and to other graphical tools on the server machine.

3.2.3.3 The Virtual Programming Laboratory (VPL)

VPL [DinF 97] is an integrated parallel programming environment. It consists of a visual file manager for manipulating files and directories in a user's account, laboratory modules for compiling and executing message-passing MPI [MPIF 94] programs (written in Fortran, C, and Java) and data-parallel programs (written in Fortran 90 and HPF), as shown in Figure 3-2. It also has a performance analysis and visualization subsystem to depict executed programs' performance behavior as animated or static displays, and a graphic plotting component to materialize output data as two-dimensional plots.

VPL targets the teaching of parallel programming using high-level parallel languages and low-level message-passing libraries. VPL was first used in a graduate level computational science course at Syracuse University during the Fall 1996 semester. Students used VPL to do their Fortran 90, HPF, and MPI programming assignments and accessed relevant resources on a DEC Alphafarm cluster via standard Web browsers. VPL users have virtual accounts that are constructed by partitioning the file space provided for a dedicated UNIX account into subdirectories for each user. VPL supplies users with a restricted set of UNIX shell utilities. The accounts are password-protected. NCSA's Web server is used, and built-in basic Basic Web authentication provides access to the accounts.

Key user interface items of the VPL are illustrated in Figure 3-3. They are the file manager, text editor, and the HPF and MPI programming laboratories. Each of these items is briefly explained below:

File Manager

The file manager is a simple visual file-browsing tool similar to its Windows-based PC counterparts that perform directory and file operations. It is made up of multiple frames: one for

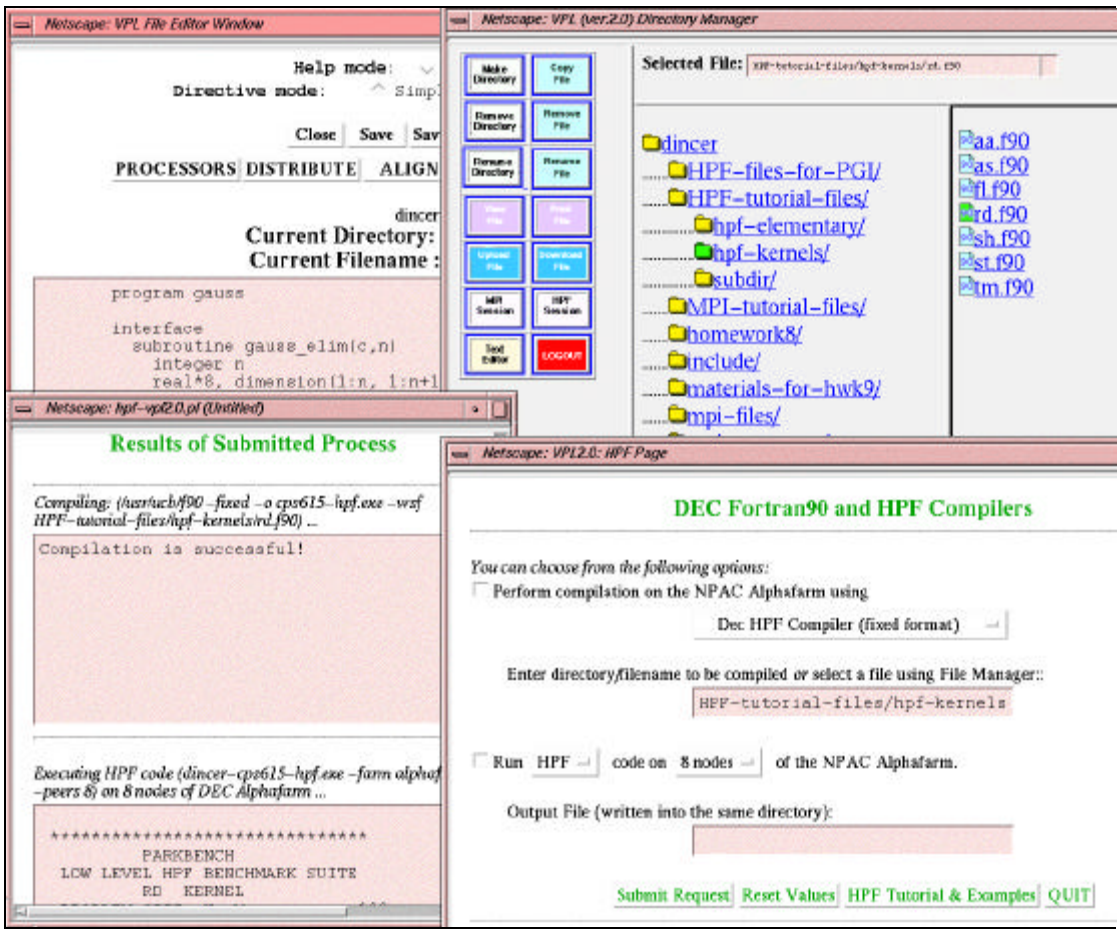

Figure 3 2. Snapshots of VPL v2.0 s Web-based user interface. On the left, HPF compilation screen and HPF-extended text editor is shown.

operation buttons, two for listing the users' files and directories, and another for displaying the currently selected file and directory. Most of the actions selected by pressing buttons are applied to this current directory/file pair.

`Directory` buttons are available for opening a new directory, and for removing or renaming the currently selected directory. `File` buttons are used to copy a currently selected file into the same or another directory, and to remove or rename the currently selected file.

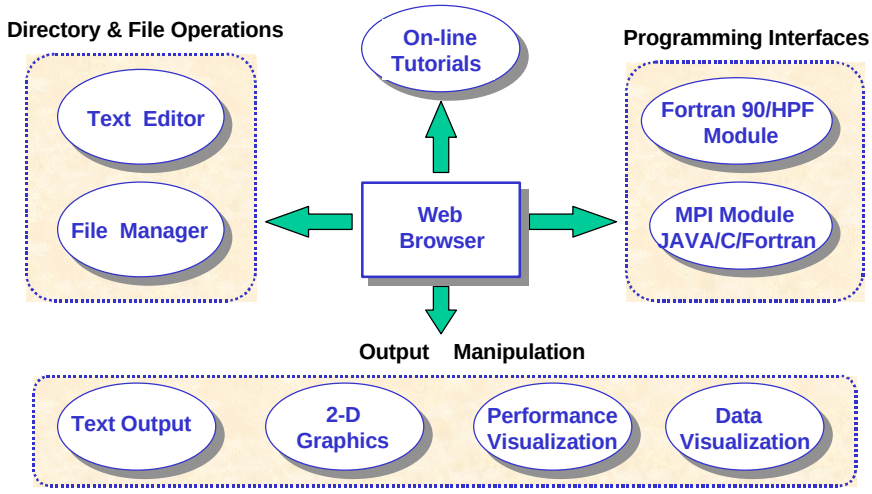

Figure 3 3. VPL User Interface Items.

`View/Print` buttons open a separate window to display the contents of the currently selected file. The user can print the contents of a file by using the browser's `print` function.

In addition to all of the above, buttons for activating the text editor or HPF and MPI laboratory sessions on a separate window are provided for users.

Text Editor

The CGI-based text editor with extended functions written in JavaScript allows the user to create new files or edit existing files without leaving the virtual lab environment. The editor functions are supported/complemented by CGI scripts on the server site.

The use of a platform-independent, Java-based editor or traditional UNIX editors such as vi, emacs, or pico could also be allowed. VPL provides some integrated "on-line help" with the editor for novice High Performance Fortran (HPF) users. When the user switches the help mode on, a hint window appears for each selected HPF directive. The supplied hint may be as simple as

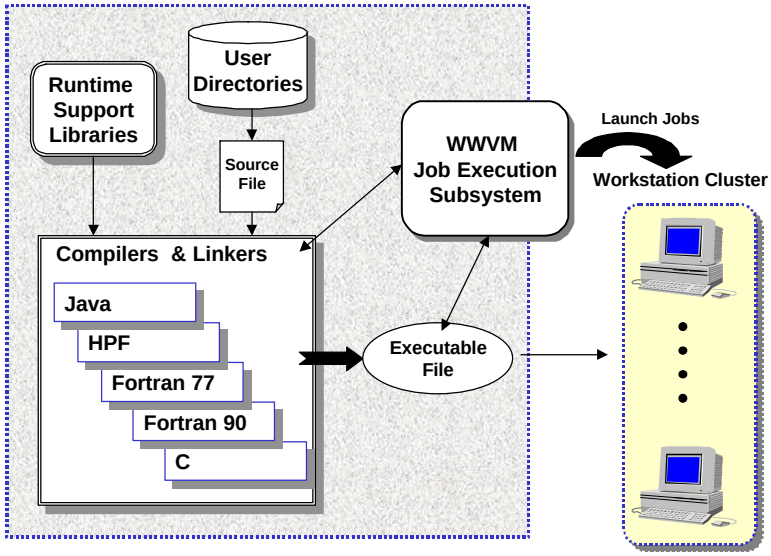

Figure 3 4. VPL server-side compilation items.

typing an example directive of the chosen type directly into the editor window (simple mode) to prompting the user to enter variable parts of the directive while the other parts are filled out automatically (prompt mode).

HPF and MPI Programming Laboratories

VPL users are presented with a "form" interface where they select the services that they require. VPL supports HPF/Fortran 90 programming and MPI programming in C/Fortran 77/Java on parallel machines. Users may choose a file for compilation and an appropriate compiler, along with the number of processors needed for execution on the target (Figure 3-4). The compilation is achieved either by directly activating related compilers on the target machine or by using makefiles that indirectly activate the compilers (only if the necessary object file is not already in the user space). Using makefiles prevents redundant re-compilations. Specifying input files or

output files for redirection of program input and output is also possible. The output can be written in a specified file in the same directory.

3.2.4 Related Work

A few similar tools, built using Web and HPCC technologies, exist that target goals similar to those of WPPEs. They all provide some kind of access to high-performance computational resources via the Web-browser interface.

The WebSubmit [KPL+ 96] system of NIST Information Technology Laboratory is another Web-based system for submitting jobs to an IBM SP-2 Load Leveler. It is more of a Web-based interface to Load Leveler than a programming environment as WPPEs are. Its functionality can be easily added to the WPPEs as a program execution option on similar platforms. The Northwest Alliance for Computational Science and Engineering uses a Web-based terminal emulator [WebT 96] written in Java and other Web tools for interactive teaching of UNIX and computational science and engineering from an integrated Web-based environment. Such a Web-based terminal emulator can add value to the WPPEs for certain applications that require frequent arbitrary interactions with the user.

The EPIC (EPCC Interactive Courseware) [EPIC 96] system, developed by the Edinburgh Computing Centre, is an on-line interactive education that combines both on-line exercises and hypertext course materials. It allows users to read through the course notes on the Web at their own pace, thus making it possible for the user to assimilate information at a self-determined speed. EPIC also contains an on-line interactive exercise component that helps users test and make use of their newly acquired skills.

3.2.5 Comparison of Client Site vs. Server Site WPPEs: VPL vs. EPIC

Having the exercises accessible directly within the framework of one single courseware package allows a much smoother method of working. In many instances this will remove the need for configuring some machines to run the programs during the courses. EPIC can be used to automatically configure the machine with the appropriate software and allow the users to spend their time studying the course materials rather than setting up the system.

EPIC addresses goals similar to those of the WPPEs, but it is essentially a pure client-side solution and thus different from the WPPEs which keep editing and visualization on the client but use the server to run large-scale simulations and host the HPF and other compilers. Other properties of these two systems are compared below:

- *Ease of installation.* In order to use the EPIC package, the user must first prepare the local system for EPIC, which involves downloading, uncompressing, and untarring the shell and Perl scripts that control the execution. In addition, the user needs to transfer the client-side of the EPIC tutorials and add EPIC Mime type to the. mailcap file (i.e., file `applications/x-epic` entry) and specify the EPIC control master as the corresponding helper application. This causes the browser to launch the specified helper application that is capable of understanding the information sent from an EPIC server, rather than trying to decode EPIC Mime types. This setup is required of all new users.

  Although automatic ways to download and install these software packages to the user's system have been included in the EPIC package, the user still needs to make decisions about where to install the new software and how environment variables should be set up. In addition, several potential problems may be encountered during the setup of the scripts. VPL,

on the other hand, requires no initial setup by the user. Anybody having a Java- and JavaScript-enabled browser can easily access the VPL.

- *Portability and client-site expectations*. EPIC needs several software packages (Perl, xterm, and make facilities) to be installed on the user's system. In addition, each exercise needs its own local applications. For example, to run an MPI application, the CHIMP version of MPI developed by the Edinburgh Parallel Computing Centre should be installed on the user's machine. To run an HPF application, either PGI's or DEC's HPF compilers should exist on the local machine. EPIC is designed to support users on a set of selected Unix platforms.

  These requirements of EPIC are expensive requirements for sites with limited funds and computational power. Many of the licensed software packages, such as the commercial HPF compilers, are not within the reach of small institutions. In addition, an individual user may find the disk space required for installing all these software packages a limiting factor.

  In contrast, a VPL client may have a Mac, a PC running Windows or Linux, or any Unix workstation as long as there is a browser. Every type of software package that is expensive or that requires a lot of disk space is already installed on the VPL server site.

- *Usability Areas.* EPIC is customized only for teaching. On the other hand, VPL is also targeted towards being a generic interface for parallel computer platforms. We discussed various uses of VPL earlier.

- *Software updates.* VPL makes it easy to update required educational software easy. It can be extended to include new scheduling policies, or new parallel machines at the back-end. All these changes can be made transparently to the user. EPIC users, on the other hand, have already installed the necessary client-site files and would need to download the new copy of the software in order to take advantage of the improved version of EPIC.

- Conventional approaches to software development and distribution are often frustrating. The cost of making a software cross-platform portable often dominates efforts to develop and test new functionality. Users of the software often complain about installation difficulties and problems with patch distribution for new releases. One way to address this problem would be to offer only shrink-wrap software releases to users. This is usually not a tenable option, because users often employ versions of operating systems, platforms, or compilers to which the institution developing the original software had no access. In order to make software portable, it needs to be tested on a number of platforms and with a number of different compilers from various vendors.

  The client-server model of keeping control of the software and shipping computational services on the Web seems to be the best way to overcome the difficulties of shipping software. Furthermore, distributed access eliminates the need for users to have high-end hardware systems or expensive licenses for the proprietary software systems with which the original application runs efficiently. A centralized server model also means that we can make changes and upgrades to the server at a single site (or a small number of controlled sites), and the new improved software is instantly accessible to all users, including PC and Mac users.

  Using a central computation server (i.e., WWVM server) also makes it easier for the WWVM system administrator to change access restrictions and management policies in a quick and convenient manner without needing super-user privileges. A further advantage of client-server architecture is that quick implementations are possible, since there is no need to modify the server and browser codes.
- *Adaptability.* VPL can be configured to offer more capabilities to Unix clients. For example, the user who has the X-term facility will be able to choose an editor of choice, such as vi, emacs, or pico, and may use xv or debuggers for programming.

3.3 Portability of the WPPE and WWVM Software

Platform-independent components are readily supported in other systems without requiring any further processing. *Portable components* can easily be transported to other platforms, a process that requires at most a recompilation. Actual binaries may differ from system to system, but they all behave similarly, regardless of the platform they run on. A WWVM system configuration file is provided to ease the porting process. It consists of the locations of user directories in the new system, that are used by the VPPE, and the URLs of the WWVM servers. The WPPEs, therefore the WWVM software, discussed here are built upon platform-independent and portable components:

- The user interface is implemented using HTML documents with JavaScript functions and Java classes that are all platform-independent. The Web server sends them to the client browser as ASCII or byte-code files, and the browser interprets them in order to shape the user interface to perform any necessary actions.
- The CGI scripts that control the actions of the server in response to user requests from the client browser are written in Perl and kept as ASCII files. The Perl interpreters process them in the same way. The pathname for the Perl interpreter may be the only required change in the new system. Items forming the back-end computation engine of the server (such as the compilers of various languages, and their loaders and linkers) and runtime support libraries may have a different command format and arguments. These are not part of the WPPE software, but of the underlying system. Depending on the UNIX shell[1] installed on the new system, the *system* and *exec* commands may need to be modified.

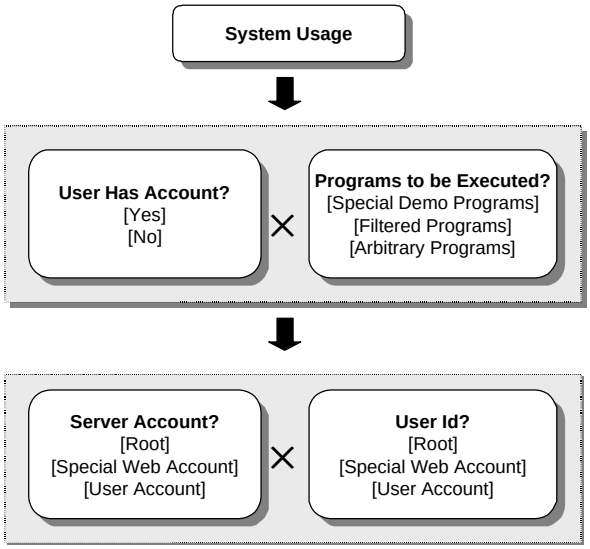

Figure 3 5. Relationship between intended usage and configuration of server account and user ids.

- The required Web (HTTP) server software may be acquired from different sources. It is also possible to find free Web servers for any platform, many of which are compatible with their commercial counterparts.
- Similarly, Web browsers are available for all UNIX machines as well as personal computers. In order to use WPPEs, the user's browser should support JavaScript 1.1 and Java.

3.4 Implementation Issues

Depending on the intended usage and the underlying system properties, the requirements and implementation strategies for the WWVM vary significantly. In this section we will discuss some

[1] There are different shells (i.e., UNIX command interpreters) available. The most common ones are sh, csh, jsh, ksh, bash, and tcsh.

of the important issues and implications of several design choices. The following issues play important roles in the design:

- Providing user accounts and the associated Web server configuration options.
- Ability to use server-site utilities.
- Taking advantage of the client-site software.
- Running Web servers on parallel platforms.

In the following sections WWVM and WPPE terms will be used interchangeably.

3.4.1 Providing User Accounts and Associated Server Configuration

The intended use of the WWVM system determines the type of programs to be executed by system users and whether the users of the system will have accounts on the system. In turn, these two factors determine the server account and the user id options in the server configuration, as shown in Figure 3-5. The *server account* is the account where the original server daemon process is started and run. *User id* is the id that the server will use to answer requests. This may be the same as the server account user id, or if the server is run as root, it may be the user id of the children processes. CGI scripts and computational processes are activated using this user id. Table 3-2 will be referred to throughout this section when the configuration of the Web server is to be explained. The table also associates those configuration options to the potential usage of the WWVM system in building a WPPE.

The users that access the WWVM servers may or may not have an account on the system. Anonymous users can access the resources of a WWVM that is configured as an unsupervised demo WPPE without requiring a user id or password in the system. In this case, the server is run from a dedicated Web account and as soon as a connection is made the user id is switched to nobody, which has minimum privileges in the system. Read-only and execute-only file access

permissions given to those users should be adequate for running demo programs. In order to prevent anonymous users from overwriting each other's temporary files created as a result of the compilation and execution operations, each anonymous user is assigned a session id. This session id is a small tag derived from the connecting host id of the anonymous user.

The WWVM provides two types of system accounts for its users: a real UNIX account or a virtual user account.

3.4.1.1 Real UNIX Accounts

The WWVM provides a convenient front-end to users' UNIX accounts and a GUI for accessing high-performance computational resources. Since the underlying operating system utilities readily provide most of the protection, file access, file space, and security mechanisms, the implementation of WWVM mainly concentrates in providing a user interface, and on providing authentication and access control mechanisms for accessing user accounts through a Web interface.

Usage	User has account?	Server Account	Process ID
Supervised Demos	Yes (virtual)	Web	Web
	Yes (real)	Root	User
	No	Web	Web
Unsupervised Demos	Yes (virtual)	Web	Web
	No	Web	Nobody
On-site and Virtual Workshops	Yes (virtual)	Web	Web
	Yes (real)	Root	User

Table 3 2. Appropriate server account and user id choices for various WPPEs.

As shown in Table 3-2, the server may provide access to real UNIX accounts by initially running under the *root* account and promptly switching to the user's UNIX account id by executing a *setuid* command as soon as the authorization is completed. A few lines of CGI code

take care of this process. Notice that this is not the same as configuring the Web servers to switch to another user id when serving the requests. The "user id" server configuration option that is used in such a case is static and the server should be restarted to change the configuration settings. This option could be useful if a separate Web server having a different access port for each user of the system were to be employed. However, such a solution would not be scalable. Although the required disk space for the WWVM server software would be minimal (less than 2 MB), a number of servers whose numbers reach to the number of users in the system would stay sleeping in the background and would consume system resources.[2] An alternative would be to start those servers via the UNIX *inet*. Even then their startup is much slower, since the binaries should be loaded on the machine first and remain sleeping in the background once they have been started.

3.4.1.2 Virtual Accounts

Providing virtual accounts for WWVM users is easy and does not require superuser privileges when setting up the WWVM system. The implementation of the required mechanisms to provide virtual account is, in some ways, similar to providing a multi-user operating system on top of a single-user operating system: file protection and security mechanisms need to be implemented. The physical account's file space is partitioned among the users. CGI scripts and JavaScript functions handle the coordination and partition of disk space among users.

Providing virtual accounts should not be confused by the user directories or areas provided by conventional Web servers. In such a scheme, all user directories are under direct control of the Web server, and anybody in the world can access any document in a specific user directory by specifying its URL address. Even if some password mechanisms are used to limit the accesses to

---

Probably not the processing time, but file allocation table entries and other system resources are wasted.

the user directories, this may not be sufficient to ensure protection of the system from hackers in the outside world, because any user may unknowingly leave some executable or interpreted module in his or her directory that may give access to the whole system. However, the WWVM virtual account directories are neither provided in a built-in fashion nor directly managed by the Web server. In the WWVM, middle layer software routines handle the manipulation of user accounts.

One convenient way of providing virtual accounts is to activate the HTTP daemon from a dedicated account and to keep running with the same user id while serving further user requests. Running the server daemon as a specially created, non-privileged user is important, because an intruder who finds vulnerability in the server will have access privileges for only this unprivileged user. These dedicated accounts can be used to run supervised demo programs in WPPEs.

3.4.2 Using Server-Side or Client-Side Software

As one of the major goals of this project was to make the WWVM accessible from personal computers as well as from UNIX workstations, providing platform-independent versions of every kind of software that will be required in providing an interaction with the user was important. For example, VPL provided a platform-independent text editor that was implemented by using Web forms, JavaScript, CGI, and another one based on Java with compatible functions to traditional text editors.

Depending on the underlying system implementation and target platforms, it may sometimes be advantageous to exploit some server-site UNIX utilities. For example, when the client machine is a UNIX workstation, users may be given the option of using the X-windows-based system utilities installed on the server machine. Users of the Web/HPF module may choose to use a

traditional X-windows-based editor, such as emacs, vi, or pico, that directly opens an external window on the user's display.

Allowing clients to use some of the standard UNIX system utilities, such as the X-windows-based ones located on the server site, has both pros and cons:

- Server-site utilities actually run on the server machine and have a tendency to overload the server machine artificially, since every keystroke is sent to and processed on the server machine. On the other hand, client-site utilities such as the ones implemented using form/CGI or Java run on the client's machine; interaction with the server is required only for certain operations.
- Using server-site utilities has the advantage of bringing users a full-featured, well-tested software package with which most of them are familiar. However, Web-aware software products are not widespread yet, and therefore only custom-made utilities with restricted capabilities are available.
- Many X-windows-based utilities provide a way to access the user-account directory structure, especially when loading a file in or saving a file to the disk. The WWVM does not give the user the option of using those standard utilities in a system with virtual accounts, since the user would be able to access other users' accounts by using this mechanism.

If a component is to be used solely on the server (e.g., CGI modules), it does not need to be written in a platform-independent language like Perl. For the sake of faster execution, it can be written in a compiled language and its binary executable can be used in the server. Similarly, programs that need to be executed may times on the server can be written in standard portable languages and can be recompiled conveniently if the WWVM needs to be ported onto another platform. Also, it is not necessary to write every client-side software facility from scratch in Java to make it platform independent. Newer technologies like Active-X allow the convenient use of

utility software on the personal computers, and this looks like a reasonable alternative way in augmenting the WWVM system functionality. A user who connects to the WWVM from a UNIX platform may be presented an option to use X-windows-based utilities on the server-side when using a personal computer, the software resident on its disk may be used for some operations.

In order to keep the transition from Unix to Web + Unix, or Windows to Windows + Web, the existing software should be reused while adopting the new Web technology. Until recently, WWVM architecture was bound to using only server-site utilities for all processing. JavaScript and Java[3], and file upload-support in recent browsers has brought in the possibility of being able to do some processing and computation on the client site or upload a file to the server site only when requested. *JavaScript* is a simple, interpreted scripting language that is tightly-coupled with the browser. It can be used to control many features of the browser, as well as the elements on a page. Recent advances in browser technology have made it possible to extend the functionality of browsers by attaching helper applications and plug-ins to them. *Plug-ins* are software modules that can be written using traditional languages such as C or C++. Programmers can use the browser's plug-in mechanism to extend the types of data the browser can display. Users can install plug-ins on their disks to upgrade the browser without the necessity of modifying or updating the browser itself. One disadvantage is that although the Netscape version of the plug-in protocol is relatively straightforward, consisting of approximately 30 functions, plug-ins generally need to be written for multiple programming and execution environments, including Macintosh, Windows, and various Unix environments.

Netscape's LiveConnect mechanism makes it possible to integrate Java, JavaScript, and plug-ins into an HTML document, thus creating a powerful programming environment that can be

[3] Current browsers supporting Java still do not allow applets to access the local file system or to use network communication freely on the client site.

used to add complex behavior to a Web page. LiveConnect makes it possible for each of these technologies to interact with each other, taking advantage of the strengths and capabilities of each. JavaScript can call functions provided by plug-ins embedded on a page, as well as Java applets, in order to change the behavior of the plug-in or applet.

A Web browser can pass data from the server to a helper application that can interpret the incoming data. The mechanism that allows the browser to determine data types and activate the appropriate helper application is based on MIME types. MIME is an acronym that stands for Multi-purpose Internet Mail Extensions [BorF 93]. A MIME specification is a string that has the form "`type/subtype`," where *type* and *subtype* are arbitrary strings that correspond to a specific data type. In this way, standard utilities that many people are accustomed to using on personal computers can be exploited with little modification. We expect a technology trend in this direction. For example, many text editors can be customized with little effort to access files across the network at URL addresses and to interact with Web browsers. This may also allow part or all of the user's file space to be kept on the client-machine, and this approach seems to be more in line with the current trend toward extending Web browsers with helper applications.

3.4.3 Running WWVM Web Servers on Parallel Platforms

There are a few issues that originate from running WWVM HTTP servers on parallel platforms. If the system WWVM Web server that is installed does not permit the use of `rsh` or `rexec` to execute programs on a parallel computational platform on the same domain, then the Web server should be run on a node of the parallel machine. This has the disadvantage of consuming the resources on the parallel machine. Furthermore, if users have their own servers running, the accounting process is affected. Parallel computer users are usually assigned a certain amount of computation quota and are charged for their process time usage. If WWVM users log

into their real accounts, calculating the charges is trivial. Otherwise, the Web system must cooperate with the accounting system in order to reflect the usage of the resources to the user's actual account.

Another issue is the need to automate the process of starting the server during the reboot of the machine in order to keep the system functional, because parallel machines usually have a shorter mean time between failures. Servers may need to be started automatically by using *crontab* files after every crash of the machine on which they run. A related subject is the detection of broken links to the high-performance machines in advance. Since the WWVM server acts as a gateway to the back-end computational resources, it checks the status of the systems at regular intervals and disables links to crashed or unavailable machines.

3.5 System Security Measures

A *security mechanism* is the means or technology of enforcing a security policy. This section discusses the mechanisms that are crucial in protecting the WWVM against security threats. Only the services related to Web-based implementation of the WWVM are elaborated upon, and individual machines on which the Web clients and servers of the WWVM run are assumed to be protected using traditional security means. A detailed discussion of traditional computer and Internet security services can be found in [RusG 91, ChaZ 95, CheB 94, and GarS 96].

Threats inherent on the Web fall into the categories of disclosure (loss of confidentiality or privacy), modification (loss of integrity), fabrication (loss of authenticity), repudiation (loss of attribution) and interruption (loss of availability) [MSB 95].

A combination of various techniques such as identification and authentication, access control, auditing, and encryption are used to protect the data and resources of the clients and servers and

the data transmitted between clients and servers. These techniques will be discussed in the next section.

3.5.1 Protecting In-Transit Data

Data on a network is susceptible to interception by a malicious attacker. There is an entire set of tools for listening on and for monitoring the flow of data packets as they traverse the network. When a user logs into a computer across a network (for example, using `telnet`) the typed password is transmitted in clear-text across the network and can easily be read by a network monitor. Similarly, sniffing is of interest to Web servers, since many Web servers use a simple password authentication mechanism for restricting access to documents. The best defense against *sniffer attacks* is to use encryption to create passwords that cannot be read if sniffed.

Encryption is a technique used to protect information and involves encoding a readable message into a *cipher text*. The cipher text can only be decoded by someone who possesses a secret piece of information known as a *key*. Encryption is used to assure the confidentiality of information, both in storage and in transmission. *Link encryption* refers to the encoding of all data that passes over a particular medium, such as a wire. When data is transmitted on a private link between two networks, link encryption may be used to protect the data. The link encryptor at the sending end encodes the data and transmits it to the link encryptor at the receiving end, which decodes it into a readable form. Link encryption provides reasonable assurance that no user can obtain data as it traverses the open environment. However, link encryption does not provide any protection for data in the LAN environment, nor can it be used for data being transmitted over a public WAN. Two forms of encryption in common use are *conventional* (symmetric) and *public-key* (asymmetric). In general, conventional encryption is used to protect the confidentiality and

integrity of large amounts of data, while public-key encryption, which imposes a greater processing burden, is often used for user authentication purposes [Stal 95].

3.5.2 Protecting Data on Server

A combination of identification and authentication, access control, and auditing mechanisms can be used to protect the data on the server. Through user-level identification and authentication and host-based access control, legitimate users' access rights are established and the resources of the WWVM server are protected from unauthorized access. Besides these three major protection mechanisms, a number of other additional precautions need to be taken in Web-based environments, as will be discussed in this section.

3.5.2.1 User Identification and authentication

User identification and authentication is the process of identifying a user and corroborating that identity. Authentication of user identity is usually based on a password and allows a computer system to associate a specific individual with particular actions. On each login attempt the system compares the password typed by the user to the one stored earlier in the system. If a password matches, the user's identity is confirmed, and access to the server-side resources is granted.

3.5.2.2 Access Control

Access control means granting or denying a user or group of users the right to access a host, or a file or other resource on a host. Access control relies on information from the system's identification and authentication mechanisms, and must be closely integrated with them.

3.5.2.3 Auditing and Logging

Auditing is the process of collecting and recording security-relevant activities on a system. The auditing and logging of access attempts are important mechanisms for tracking security on any Internet host. The most important log to monitor in a Web server is the record of who has attempted to access, or authenticate in order to gain access to, the server's documents. The Web server's log files should be saved and routinely scanned for security breaches. If intermediate proxies or Web firewalls are used, it may also be desirable to provide auditing at those points. If properly placed, this would allow an administrator to determine which types of Web requests were made into and out of a protected network.

Auditing provides the ability to track requests on a *per-user* basis. For example, it could provide a detailed list of all actions taken by a particular suspicious user. Auditing also provides the ability to track accesses on a *per-resource* basis. For example, log records would make it possible to determine who is accessing particular Web documents on a server, and which documents on the server are accessed most frequently.

The access permissions to the directory in which WWVM keeps the logs are set securely so that only the owner of the physically dedicated Web account can access this directory. Users of the system should be prevented from being able to modify the log. This is similar to the filtering of arbitrary user programs, which is explained in the next section. The WWVM HTTP server should never be run as *nobody*, since in that case any user would be able to modify logs that keep the accesses to the system.

3.5.2.4 Web (HTTP) Server Setup

The first potential security threat may come from the HTTP server software itself. The size and complexity of Web servers makes them vulnerable to attacks. It is crucial to use one of the

servers known to have no security bugs [WebS 96]. Referring to security-related Usenet newsgroups is a good way to keep informed of potential security problems. Once an appropriate Web server is chosen it has to be configured in a very careful manner in order not to hinder the security of the entire system. Many of the threats that occur may be the result of a careless configuration of the HTTP server and the poor organization of the server document directories [UnixS 96]. When configuring the Web servers used mainly to serve static documents, it is recommended that extra protection be given to sensitive directories by turning off the *server-side includes* (*ssi*) option. This option allows static HTML documents to be enhanced at run-time (e.g., when delivered to a client by the server.) The *ssi* can be abused by hostile users who prey on scripts that directly output things that have been sent to them. However, in the WWVM environment it was almost always necessary to use the *ssi* option to create dynamic documents on the fly.

Besides the negative aspects of security, the use of the *ssi* option may slow down the serving of documents. Performance is an issue, since the server is not just sending some HTML to a client, but parsing every HTML file for *ssi* directives before sending it to the client and building that HTML on the fly, or actually executing commands on the server. Therefore, the *ssi* option was enabled on a per-file basis in order to prevent the parsing of all files for *ssi* directives. The *ssi* option also inhibits the caching of documents on the client-side, since the content of each shipped document is created anew.

3.5.2.5 Required Authorization Checks and Execution Rights

Running the server as *root* or *superuser* is always dangerous; therefore, if the server is run under the “root” account, the script interpreted with the *root* privileges should not be longer than

a few statements. If the authorization system breaks after authentication, but before switching to the user's ID, the intruder may gain access to the *root* account.[4]

On the other hand, a specific WWVM server, such as the one used in unsupervised demos, may give access to anonymous users without needing password authentication. Anonymous users have no write or modify access rights on the system. They are able to run the special demo programs, but are not allowed to type in or upload their own programs As a precaution, the server logs the IP address of the client machine and put limits on the stack size and execution time of the anonymous user processes.

3.5.2.6 CGI Scripts

The CGI scripts compute information to be returned to users and are often driven by input from the remote user, who may be hostile. If these programs are not carefully constructed, remote users may be able to subvert them to execute arbitrary commands on the server system. Almost all vulnerabilities arise from these issues. The contents, permissions, and ownership of files in the CGI script directories should be carefully set. It is a good idea to provide CGI modules as statically linked binaries rather than as interpreted scripts. This will remove the need for a command inside the *chroot*-protected environment. The `chroot` system call changes the root directory for the process and all of its children. This means that the directory in which the Web server daemon executes appears as the system root directory to the Web server and all the processes it initializes. Other higher level directories become not inaccessible. The disadvantage of this approach is that it can be very difficult to configure a system, and all system programs, and any libraries and files that the Web server uses must be copied into the *chroot*-protected environment.

[4] In an earlier version of the NCSA server, a bug was detected that caused the server to crash with a page

3.5.2.7 Filtering User Input for Metacharacters

Some input fields typed by users may be used as input to command interpreters such as Perl, AWK, UNIX shells, or programs that allow commands to be embedded in outgoing messages such as `/usr/ucb/mail`. Any characters that have special meaning to the underlying shell in a query string may cause a CGI script to misinterpret those characters. A mischievous client may use special characters to confuse your script and gain unauthorized access. Languages like Perl and the Bourne shell provide an `eval` command that permits the execution of a string constructed by another statement on the fly. If the server script is going to use any data from the user to construct a command line for a call to `popen` or `system`, backslashes should be put before any characters that have special meaning to the shell before calling the function. Using JavaScript functions at the client site, and CGI functions at the server site the input should be checked against all kinds of special Unix meta-characters[5] [CERT 95] and rejected if determined to be potentially dangerous. This additional filtering at the server-site may seem to be redundant, yet we have determined that it is critical in order to prevent mischievous attackers from interpreting manually supplied command strings.

3.5.2.8 Protection Against Other Users

When the WWVM supports virtual user accounts, another concern is the possibility of attacks by other legitimate users of the system. Several measures can be taken to prevent users of virtual Web accounts from being able to access other users' account. One way to prevent this is to set up a special account for running the server and allocate user file spaces under this account. This brings the flexibility of using UNIX file protection mechanisms to protect WWVM user

---

fault when very long query strings were given, which made the system vulnerable to hostile attacks.

[5] Some of them are `\n, \r,(,.,comma, /, ;, ~, !, ), &gt, |, ^, &amp, ; $,`, and `&lt.`

directories from other users having regular UNIX accounts on the same system. Furthermore, the diagnostics and error messages never indicate the actual location of the Web account and directory pathnames to the users.

In addition to the above measures, it can be ensured that WWVM users have no way to go out of their directories by using any of the utilities provided by the WWVM system. VPL is the most capable system in this respect and approximates the Unix shell's functionality. In VPL, the commands such as `delete/make/rename` directory or `delete/remove/rename/copy` file always manipulate items in the user's home account. Using JavaScript for validity checks at the client site, and more advanced CGI script-based checks on the server site guarantees this result. Other systems need only a subset of these precautions to ensure privacy of accounts.

3.5.2.9 Filtering Arbitrary User Programs

It was a challenge to provide access to more powerful, higher-level computing services without compromising security. When the server is running with users' UNIX account IDs, users are able to run any arbitrary programs, as they are executing in their own accounts. However, when virtual accounts are used, the WWVM should bring further restrictions to the programs to be executed by the system. Programs supplied by the user for execution may also contain possible system calls. The user programs are filtered for system calls, and reject the execution if one is found. The user may gain some access to other directories of the same account by using input/output statements.

Confining untrusted programs is technically simple when the code is being interpreted. Since every single instruction is executed under the control of the interpreter, the interpreter can check the instruction arguments to ensure that all accesses are legal before executing the instruction. For example, Netscape [Nets 96] prohibits all local file accesses and allows network connections only

to the server from which the applet was downloaded. Something similar could have been done in the WWVM, but instead, WWVM restricts the input/output syntax/semantics only a little and allows user input/output. The input/output statements are filtered against any files the users open outside their own directories. WWVM does not allow the use of variables as filenames in open statements, since this would elaborate the filtering process and might force us to adopt Java's way of thinking. The user must always put the filename in quotes in the open statement so that the system can validate it.

On one occasion following the Supercomputing '95 demonstration, the possibility of allowing anonymous users to run arbitrary programs in a WPPE without hindering the security of the system was evaluated. One obvious security concern was the inevitability of giving Web clients the ability to compile and execute arbitrary programs on local hardware with whatever privileges the Web server might have. Refusing programs that contained system calls can preclude some mischief, but users can easily trick the server into giving out, for example, password data using only standard I/O. Therefore, the WWVM takes the safest choice of completely prohibiting the execution of arbitrary programs by anonymous users.

3.5.3 Client Protection

Web clients store and process information that may have been obtained from local or remote Web servers. This causes the protection role of the client to be twofold. First, the client must protect data intended for one user from being modified or disclosed to other users. Second, the client must protect its own data from being modified or disclosed as a result of executing remotely obtained software.

To protect remotely-obtained data, the client must store and process it in a manner that prevents it from being accessible to users who do not have legitimate access to it. Furthermore,

many Web browsers make extensive use of caching, in volatile memory as well as on disk. This introduces a significant performance gain for browser users, but it also makes it possible to retrace a user's entire session--even after the client has been disconnected from the network. Alternatively, if access control is implemented only in the server and there is no local access control policy such as is in a personal computer cluster for public use, there must be a way to ensure that cache files and other data objects are deleted between user sessions. This may be handled by modifying the client application, or through procedural means. In addition to this application-level object-reuse issue, there are OS-level object-reuse issues. If authentication information was passed from the client application to the server, that information must be cleared from memory so that it is not accessible to other users. When cache and other data objects are deleted on the client, the OS must clear the corresponding disk sectors prior to allocating those sectors to a new user.

The largest risk in the Web environment is due to the fact that browser tools enable a user to download data from any location on the Internet and use a local application to process that data [Dalva 94]. An optional feature of HTTP protocol is the *content negotiation* of MIME types. In the client's request to server, the `Accept header` specifies which MIME types it is prepared to accept. Unfortunately, most Web servers do not implement content negotiation and totally ignore the client's `Accept` specification. Therefore, it is the responsibility of the client to reject any unwanted data before it is displayed or stored. Fortunately, the server tells the client what to expect about the data it is sending back as part of the header information. The client computer cannot rely on the Web server to filter out hazardous information on its behalf. Firewalls are often used to perform this filtering operation for Web clients.

When the Web browser receives data of a specific MIME type, it needs to know how to handle or display that file to the user. Each Web browser contains its own set of display functions

that will work for certain MIME content types, or the image can be displayed by an external viewer program designed specifically for that MIME Content-type. There is a special file called the *mailcap file* that describes the rules for displaying incoming documents. There is a default mailcap file that defines general rules, a system-configurable mailcap *file* that defines site-specific rules, and a user-configurable mai1cap file for defining personal rules. The mai1cap file mechanism accounts for much of the browser's openness and flexibility [DKF 94].

If the data is simple text or graphics, downloading and displaying it may involve relatively little risk. However, even a word processing document obtained over the Web may include macros that are executed by the word processing application. Some users of Microsoft Word have already reported such a virus. Many Web browsers have support for documents that are essentially executable programs, such as those written in Postscript. Such programs, when downloaded and executed with a click of a button, may be able to directly access files or memory. The dangerous scenario for a Web browser is that a malicious Web server could send Postscript files that contain instructions to execute dangerous actions on the user's Web browser. The browser would hand off the file containing these instructions to the viewer (thinking it was a harmless document to be displayed) and the viewer would kindly execute the dangerous or destructive instructions.

3.5.4 Implementation of User Identification and Access Control Mechanisms

3.5.4.1 Basic Web Authentication Facility

The *Basic Authentication facility* [LPJ+ 94, NCSA 95, LuoB 94] is defined in the HTTP protocol standard and is built into most Web browsers. It is a simple password authentication

scheme that restricts access to documents according to user name, group membership, or membership of the client computer in a specific network domain.

The Basic Authentication facility uses *access control lists* to enforce authorization. Access control lists are the most common type of authorization mechanism and are also used by the UNIX system [Bach 86, LMK+ 88], the Andrew File System [Howard 88], and the Distributed File System (DFS) [Chap 94]. An access control list enumerates each individual, or group of users, that is granted or denied access to a resource. *Authorization* is the process of enforcing the rules or policies defined in the access control lists.

With Basic Authentication, the WWVM administrator places access control specifications in a central system configuration file or in a special file in each protected directory. In the latter case, the access control specifications apply to all the files in the directory. Most servers can be configured to restrict access to documents according to what domain the user connects from.

The Web server enforces the rules specified in the access control list as follows: For each document request, the server checks to see if that document resides in a directory protected by an access control list. If so, then an access control rule must be parsed and possibly enforced. The server checks the network address of the client computer against the relevant rules to see if it should be denied or permitted access. If the directory is limited to specific users, the user's identity must be authenticated with the password mechanism. The Web server daemon will examine the `Authorization header` (if any) of the request sent by the client. If the authorization field is missing or empty, the Web server knows that the user has not yet authenticated and sends a response with the `status code 401 (Unauthorized)`. The response will also have a header, `WWW-Authenticate`, which indicates the type of authentication required and the domain the document is protected by. In response, the browser will ask the user to enter a password. When the user types the password, the browser scrambles it using the

`uuencode` algorithm and sends the original request again by adding the scrambled password in the `Authorization header`. This time, the Web unscrambles the password information using the *uudecode* algorithm, and confirms the identity of the user by comparing the password sent by the user to the one stored in the Web server's password file for that user. If the password is incorrect or the user is not allowed access to the document, the server will return an error with `status code 403 (Forbidden)`.

For restricted access documents a Web server must authenticate each request, since the HTTP is stateless. The Web browser may reuse the password several times by sending the same `Authorization header` with each new request to the Web server. It is not necessary for users to type their passwords again. However, the server must still do the same amount of work authenticating the user for each protected document. So, for a document with multiple inline images, the server would have to decode the authorization information multiple times, once for each request.

3.5.4.2 CGI-Based Authentication and Access Control

Alternatively, the access control mechanism may be completely implemented by using CGI programs. A specially configured server is not required. The environment variables `REMOTE_ADDRESS` and `REMOTE_HOST` return the IP numerical address and IP alphanumeric name for the remote user, respectively. The advantage of using a CGI script is that the Web server does turn away the domains that are not allowed to access the system, but may send them different documents. The disadvantage is that access-control enforcement implemented at the server configuration level is far more robust than the CGI-based implementation.

Similarly, a CGI-based solution is possible for user authentication. The password that was entered into the HTML logon form is encrypted and sent to the server. The server passes it

through a routine to encrypt the password in the same way in which it appears in the password file and checks it against the stored password file. A legitimate user is allowed access to the protected files, and the server maintains the user's name and passes it to any subsequent CGI programs that are called.

The first time the user contacts the server, the *session id* is created (either a cookie or a hidden variable). The program creates a random and unique session id by using the crypt function to encrypt a string that is based on the current time and the remote address. The server passes it to the client, which sends it back to the server with the next request. From then on, the server manipulates the existing cookie or hidden variable instead of creating a new one.

Using HTML Forms and Hidden Fields

Using hidden fields, dynamic CGI scripts on the server-side can embed some session or password information into an HTML form that will be sent back to the CGI program when the form is submitted. The main disadvantage of using hidden fields is that they are not secure, since a client can view them by selecting the "view source" option in the browser.

Using Persistent Cookies

Persistent cookies help the server to initiate a session with the client by sending a `Set-Cookie` response header[6] that sets one cookie on the client-side where a key is equal to a value. The cookie header requires the key/value information to be encoded. The header contains information about the path of the URLs on a particular server for which the cookie is valid, a timeout value indicating the expiration time for the cookie, and a domain name for which the cookie can be used. If the client accesses a URL from another domain that tries the retrieve a cookie, it will be unable to do so. The state information preserved in the cookies is a server-side

protocol assertion that ensures that a client will include the response on all subsequent accesses to that resource until an expiration time specified by the server. The response includes an opaque cookie, path, and timeout. The client accepts a new `Set-Cookie` header from CGI programs, and when the user accesses a certain document, the browser will send back the cookie information as the environment variable `HTTP_COOKIE`.

There are a couple of disadvantages with this client-side approach to storing information. First, the technique works only for certain browsers at the present time, namely, Netscape Navigator and Microsoft Internet Explorer. Furthermore, there are certain restrictions on the number and size of cookies. Finally, the browser is capable of storing information on the client side, which may not be considered as secure for authentication information.

3.5.4.3 Java-Based Authentication and Access Control

The Web/HPF uses an Apache SWeb server that runs on a dedicated IBM RS6000 workstation with interactive logins disabled for security. The only way to access the machine is through the Apache server. For supporting a Java-based authentication and access control mechanism, the user interface of the Web/HPF is completely written in Java. As soon as the Java applet manipulating the user interface is activated at the client's browser it makes a persistent port connection to the Sweb user agent. If the Java application closes for any reason the agent shuts down and kills all the user’s processes. As long as the user interface Java applet holds the port, others cannot send commands to the agent on the same port. This also means that users don't have to send authentication information for every command, but authenticate only once at the start of the Java application. [Lifka 97] The agent also does signal handling for added security.

---

[6] Set-Cookie: $key=$value; expires=$date; path=$path; domain=$domain.

3.5.5 Future of Web Security

There are currently two leading commercial specifications for crypto-enhanced secure Web services: Netscape Communications' Secure Socket Layer (SSL) [HeaE 95] and EIT's Secure HTTP (SHTTP) [Wong 95a, Wong 95b].

SSL and S-HTTP can be added to any environment and may be useful for adding security to Web transactions. However, neither of them appears to be complete or general enough to address the fundamental computer security issues that underlie the Web application environment. S-HTTP was developed for CommerceNet, a digital commerce testbed in California [ComN 95]; S-HTTP headers encapsulate HTTP messages that are composed of HTTP headers and documents. SSL is a communication protocol implemented as a layer between the TCP/IP transport layer and applications that send and receive messages. It provides for message encryption and certification of public key authenticity, data integrity, and server authentication. A side effect of the SSL and S-HTTP technologies is the ability to create a *digital signature* that can identify a document or message as actually coming from a client whose identity is known to the server. Both of them use certificates to verify the authenticity of the data between the browser and the server. A *certificate* is a means of determining whether a client who sends its public key is really authorized to do so by the server that is being connected. The certificate contains the server's public key and is signed by a third party, the certificate issuer. The *certificate authority* is some entity whom everybody trusts and whose public key is known. The *signing* process is simply encryption with the third party's private key. Anyone can decrypt a certificate using that third party's published public key and thus validate the certificate. When SHTTP and SSL are used together, SHTTP encrypts data at the application layer and passes it to SSL, which encrypts it a second time. As with SHTTP, SSL performs security services independently of the operating system, and access control for individual data objects is not provided. SSL has not fully been accepted but is rapidly

becoming a de facto standard. A number of secure Web servers and clients using the above mentioned technologies have recently emerged and they can seamlessly be used in the WWVM environment.